

Noah Bjerkli Aanonli
Ferdinand Wèden Snapa
Erlend Thingnes
Åsulv Vaa

Digitalt system for utlån av utstyr

Bacheloroppgave i Programmering - Bachelor
Veileder: Peter Stefan Nussbaum
Mai 2024

Noah Bjerkli Aanonli
Ferdinand Wèden Snapa
Erlend Thingnes
Åsulv Vaa

Digitalt system for utlån av utstyr

Bacheloroppgave i Programmering - Bachelor
Veileder: Peter Stefan Nussbaum
Mai 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk



Kunnskap for en bedre verden

Sammendrag

Tittel:	Digitalt system for utlån av utstyr
Dato:	21/05/2024
Deltakere:	Noah Bjerkli Aanonli Ferdinand Wøden Snapa Erlend Thingnes Åsulv Vaa
Veileder:	Peter Stefan Nussbaum
Oppdragsgiver:	Utstyrbanken Vestre Toten
Kontaktperson:	Gunnhild Røisli Andersen
Nøkkelord:	programmering, webapplikasjon, fullstack, Utstyrbanken, REST API
Antall sider:	79
Antall vedlegg:	11
Tilgjengelighet:	Åpen

Sammendrag: Utstyrbanken Vestre Toten er en tjeneste som tilbyr gratis utlån av utstyr til ulike sporter og fritidsaktiviteter. I lang tid har de brukt papirskjemaer for å holde orden på det som til enhver tid er utlånt og for å få en oversikt over tilgjengelig utstyr må de fysisk lete igjennom i lokalene. Papirløsningen er tidkrevende, derfor ønsket oppdragsgiver en mer moderne, digital løsning. Denne bacheloroppgaven tar for seg å designe og utvikle denne løsningen, som endte opp en fullstack webapplikasjon med en frontend i React og en backend i form av en RESTful API.

Abstract

Title:	Digital system for lending equipment
Date:	21/05/2024
Authors:	Noah Bjerkli Aanonli Ferdinand Wøden Snapa Erlend Thingnes Åsulv Vaa
Supervisor:	Peter Stefan Nussbaum
Employer:	Utstyrskassen Vestre Toten
Contact Person:	Gunnhild Røisli Andersen
Keywords:	programming, web application, fullstack, Utstyrskassen, REST API
Pages:	79
Attachments:	11
Availability:	Open

Abstract:

Utstyrskassen Vestre Toten is an organization who provide a free service which lends out a variety of equipment for sports and outdoor activities. For a long time they have used paper forms in order to manage their loans, and to keep inventory they have to search their physical location often. The paper solution is time consuming, and therefore the employer wanted a more modern, digital solution. This bachelor thesis covers our design and development for this solution, which ended up as a fullstack web application with a frontend written in React, and a backend powered by a RESTful API.

0.1 Forord

Denne rapporten er skrevet av Erlend Thingnes, Ferdinand Wedén Snapa, Noah Bjerkli Aanonli og Åsulv Vaa ved institutt for datateknologi og informatikk ved NTNU i Gjøvik. Vi ønsker å takke Peter Stefan Nussbaum, veileder ved vårt prosjekt, for rådene og veiledningen han har gitt oss underveis i prosjektet. Vi ønsker også å takke Gunnhild Andersen, representant fra vår oppdragsgiver Utstyrsbanken ved Vestre Toten Frivilligsentral, for et godt samarbeid under hele prosjektperioden. Videre ønsker vi å takke familie og venner som gav tilbakemeldinger på rapporten som hjalp oss å gjøre den bedre.

0.2 Ordforklaring

base64 enkoding Base64 enkoding konverterer binærdata til tekst med bruk av 64 ulike tegn, dvs. hvert tegn representerer 6-bits. Base64 enkoding er definert i RFC 4648 [35]. 47

bibliotek I programmeringssammenheng er et bibliotek en samling med ferdiglagd kode som man kan importere og bruke i prosjektet sitt. 14, 16, 22, 45, 49, 55, 62, 63

branch En branch er et konsept med versjonskontroll hvor kode oppbevares basert på rekkefølgen de ble committet i. iv, 64

brute force-angrep Et brute force-angrep [3] er en teknikk der man prøver hver eneste kombinasjon helt til man finner noe som fungerer. I passord-sammenheng vil det bety å prøve alle mulige passordkombinasjoner til man finner det rette. 10, 53

CI/CD CI/CD som står for Continuous Integration & Continuous Deployment, altså kontinuerlig integrasjon og kontinuerlig utrulling, er en metode for å automatisere utrullingsprosessen og settes vanligvis opp til å kjøre ved endringer i kodebasen for eksempel i Git når endringer merges inn i hovedbranchen. v, 3, 17, 18, 66, 67

commit En commit er en dokumentert endring i versjonskontroll. Denne består vanligvis av informasjon om hvordan filer endret seg, og inkluderer en melding rundt endringene. iii

cookie Se [informasjonskapsel](#).

CSRF-angrep Et CSRF-angrep [9] er når ondsinnede aktører får brukerens nettleser til å gjennomføre uønskede HTTP-forespørsler uten at brukeren vet om det da fra en annen nettside. 10, 49

DELETE DELETE er en av HTTP-metodene [10] som brukes i forespørsler over HTTP-protokollen. DELETE brukes for å slette en ressurs. iv, 20, 30, 46, 56

endepunkt Endepunkt betyr i HTTP-basert API sammenheng adressen (URL) man sender HTTP-forespørsler til for å utføre en bestemt handling som å hente data om en ressurs eller oppdater en eksisterende ressurs. 29, 30, 46, 56, 57, 59, 60, 64, 70, 71, 79

fragmentering I en magnetisk harddisk må et fysisk lese-skrive hode flytte seg for å finne spesifikk data som ligger fysisk et sted på en av platene i harddisken. Fragmentering referer til når data ligger spredt rundt på platene fremfor at de ligger sekvensielt, dette betyr at harddisken må bruke mer tid på å flytte lese-skrive hodet og mindre tid på å faktisk lese dataene. Fragmentering fører dermed til lavere ytelse og er derfor uønsket [17]. 15

fullstack-utvikling Fullstack-utvikling [30] er en prosess som involverer å jobbe med begge endene av en applikasjon - klienten/brukeren, og serveren. Disse endene blir vanligvis representert gjennom frontend, som håndterer utseende til applikasjonen, og backend, som håndterer applikasjonslogikk.. 2, 3, 79

- GET** GET er en av HTTP-metodene [10] som brukes i forespørsler over HTTP-protokollen. GET brukes for å hente ressurser. En GET-forespørsel skal aldri føre til endringer i dataene og regnes derfor som en trygg metode. Det vil si, alle forespørsler med metoden skal være "idempotent" [11] som vil si at flere identiske forespørsler vil ikke påvirke dataene lagret serverside. iv, 20, 30, 60
- goroutine** En goroutine [37] er en lettvekts-tråd styrt av Go sin runtime. 35, 54, 71
- hamburger-meny** En hamburger-meny er en meny som brukeren må bringe frem ved å trykke på menyets ikon. Vanligvis er ikonet tre vannrette streker oppå hverandre, dette kan ligne på en hamburger derav navnet. 22
- hash-algoritme** En hash-algoritme er en algoritme som tar inn vilkårlig data og produserer en string også kalt en hash basert på dataene. Denne hashen har to viktige karakteristikker, den ene er at det skal være bortimot umulig å reversere prosessen og gjenopprette dataene som ble brukt som inndata i algoritmen, samt at selv små endringer i inndataen skal føre til ulike hasher. 10, 49
- HTTP-metode** En HTTP-metode beskriver typen handling som ønskes gjennomført med en forespørsel. Se GET, POST, PATCH eller DELETE. 46
- IDE** En IDE er et verktøy som er laget for å utvikle dataprogrammer. Vanlige komponenter er en editor for kildekode, valg for oppsett av kompilering og bygging til en kjørbare fil samt en debugger. 19
- informasjonskapsel** En informasjonskapsel [13] er en måte for HTTP-servere å lagre små mengder med data hos klient ved å sende den i respons til en HTTP-forespørsel. Klientens nettleser vil da lagre kapslen i en tidsperiode spesifisert av serveren. Videre vil klient sende med kapslen ved fremtidige forespørsler mot samme server. En informasjonskapsel brukes blant annet til autentisering. iii
- klient** I websammenheng er klient den som sender forespørsler til en server og mottar informasjon fra server. Sluttbrukerens nettleser er et eksempel på en klient som kommuniserer med servere, men også servere kan ha rolle som klient i kommunikasjon med en annen server. En klient er enkelt fortalt maskinen som starter forespørsler. 69
- lån** Et lån er utlån av utstyr til låner for en gitt tidsperiode. Et lån kan ha to statuser, aktivt og fullført. Et aktivt lån betyr at brukeren låner utstyret for øyeblikket mens et fullført lån betyr at bruker har levert tilbake lånt utstyr. Lån er forskjellig fra en reservasjon ved at utstyret har faktisk blitt utlevert til låner. 1, 14, 30
- merge request** En merge request [23] er en forespørsel for å slå sammen kode mellom to branches. 17, 64
- metadata** Metadata er data som beskriver data.
- miljøvariabel** En miljøvariabel er en variabel av typen string som defineres utenfor programmet, men kan leses av programmet. Miljøvariabler brukes i hovedsak til konfigurering.
- mutex-lås** En mutex-lås er en måte å regulere tilgangen til delte ressurser mellom tråder som sørger for at kun en tråd har tilgang av gangen. Det gjøres ved at før tråden skal bruke delte ressurser må den "holde" mutex-låsen som er satt der og deretter "slippe" den når tråden er ferdig med å bruke de delte ressursene. Dersom en annen tråd "holder" låsen må man vente da kun en tråd kan "holde" låsen av gangen. 54
- navigasjonsbar** En navigasjonsbar er en meny for å navigere mellom en nettside sine ulike sider der valgalternativene er presentert vannrett på en linje, vanligvis adskilt fra resten av siden på toppen eller på bunn. 22
- PATCH** PATCH er en av HTTP-metodene [10] som brukes i forespørsler over HTTP-protokollen. PATCH brukes for å oppdatere en eksisterende ressurs med ny data. iv, 20, 30, 46, 57

- path parameter** Parameter som ligger i URLen/adressen f.eks. dersom man har et endepunkt som forventer at URLen som blir forespurt er på formatet `/api/loans/loan/{loan-id}` og forespørselen er på adressen `/api/loans/loan/7` så vil 7 i dette tilfellet være en "path parameter". 29, 30, 46, 64
- POST** POST er en av HTTP-metodene [10] som brukes i forespørsler over HTTP-protokollen. POST brukes for å lage nye ressurser som ikke allerede eksisterer. iv, 20, 29, 30, 57
- prop** I React er en Prop, eller Property, variabler som blir passert til klasse når den skapes. Props blir ofte brukt til å gi komponenter tilgang til felt eller funksjoner til komponenten over seg i hierarkiet. Det som skiller en property og state er at en property ikke kan bli direkte endret.
- query parameter** En query parameter er en parameter som ligger i query-komponenten av en URL, disse ligger som regel på enden av URL-en og består av nøkkel-verdi par. Query-komponenten startes med et spørsmåltegn og hver parameter skiller med og-tegn, et eksempel kan være: `/api/stats?from=2024-05-01&to=2024-05-31`. I dette eksempelet har vi to query parametre, en kalt "from" med verdien "2024-05-01" og en kalt "to" med verdien "2024-05-31". 29, 46, 52, 60, 64
- race condition** Dersom det finnes en race condition i programvare betyr det at man kan få ulike uønskede situasjoner der man kjører kode parallelt avhengig av hvilken kode som kjører først og/eller kjører ferdig først. 43, 54
- Requests for Comments** Alle de viktigste standardene for internett- og webteknologier er dokumenterte via såkalte Requests for Comments eller RFCer. De blir utviklet av organisasjonen Internet Engineering Task Force (IETF) som er den fremste organisasjonen for utviklingen av standarder for internett [15]. vii
- reservasjon** En reservasjon er en måte for en bruker å holde av utstyr for en gitt tidsperiode. Idet låneren henter utstyret ved Utstyrsbanken, er det et aktivt lån istedetfor en reservasjon. 30
- routing** Routing er en måte å fortelle en web applikasjon hvilken URL som tilsvarer til hvilken kode eller funksjon. I konteksten av en React applikasjon vil dette bestemme hvilken komponent som vises basert på .
- runner** En datamaskin, eller **virtuell maskin**, med hensikten på å kjøre oppgaver. Plattformer som GitHub[21] og GitLab[22] bruker disse for CI/CD systemene sine, spesifikt GitHub Actions og GitLab Pipelines. 66, 67
- server** En server er, i websammenheng, en maskin som kan kontaktes for å hente informasjon. En server er enkelt fortalt maskinen som svarer på forespørsler. iv, 17, 69
- skytenester** Skytjenester[6] er tjenesteytere som tilbyr ulike løsninger gjennom datasentre eller serverparker. Dette kan inkludere alt innenfor programvare (Ofte kalt "Software as a Service", eller SaaS), plattformer (Ofte referert til som "Platform as a Service", eller PaaS), eller infrastruktur (Ofte kalt "Infrastructure as a service", eller IaaS) til en kunde. 3
- soft deletion** Soft deletion er en type sletting der slettede objekter forblir lagret, men med et felt som sier at objektet er slettet. Det som tradisjonelt tenkes på som sletting kalles "hard deletion" der objektene som slettes fjernes permanent. 33, 65
- SQL-transaksjon** En SQL-transaksjon er en måte å kjøre flere SQL-spøringer på [41]. Man starter opp en transaksjon og kjører SQL-spøringer, dersom en feil skjer kan transaksjonen avbrytes. Da vil alle endringene som har skjedd under transaksjonen ruller tilbake. Dersom ingen feil skjer avsluttes transaksjonen og eventuelle endringer blir permanente. 57, 59
- state** State er en måte å lagre variabler på, som brukes for å bestemme hvordan noe skal se ut. Når man endrer state I frontend blir utseende til siden endret på med de nye variablene..

statuskode En statuskode er en tallverdi sendes med i respons til HTTP-forespørsler. Statuskoden indikerer resultatet av forespørselen. Statuskoder er inndelt i kategorier, eksempelvis er kodene som begynner på 2 en indikasjon på at forespørselen var suksessful mens kodene som begynner på 4 indikerer at det var en feil med selve forespørselen. Betydningen av statuskodene er standardiserte og definert i kapittel 15 av RFC 9110 [14]. 55

tokenbøtte-algoritmen Tokenbøtte-algoritmen [18] er en algoritme som kan brukes for *rate limiting*. Den kan best beskrives dersom man ser for seg en bøtte med tokens. For hver forespørsel tas en token fra bøtta, dersom bøtta er tom vil ikke forespørselen gå igjennom. Bøtta fylles opp med en og en token ved et fast intervall. Bøtta har en maks kapasitet for antall tokens, den vil ikke fylles opp mer enn dette. For å kontrollere rate limitinga kan man dermed justere hvor ofte bøtta fylles opp samt dens kapasitet. xv, 53

Transport Layer Security Teknologien som sørger for kryptering over HTTP-protokollen.

utstyrstype Med utstyrstype menes det en noe som Utstyrsbanken låner ut. Noen eksempler på dette kan være fotballer, langrennsski og soveposer. 3, 7–9, 14, 32, 61, 69

versjonskontroll Med versjonskontroll menes et system som sporer endringer og gir muligheten til å sette sammen endringer fra ulike versjoner samt rulle tilbake endringer om ønskelig. iii, 17

virtualisering Virtualisering vil si å etterligne maskinvare i programvare. Et eksempel er virtualiseringsprogramvare som lar deg lage *virtuelle maskiner* der man kjører operativsystem og programmer som tror de kjører på en egen maskin, men alle maskinvarekomponentene som prosessor og harddisk etterlignes av virtualiseringsprogramvaren. En annen variant er OS-nivå virtualisering der man kun etterligner operativsystemet og programmene vil her kjøre direkte på den reelle maskinvaren. vi, 17

virtuell maskin En virtuell maskin er en *virtualisert* datamaskin. I praksis er dette en maskin lagd fra programvare [42]. En slik maskin kan kjøre et eget operativsystem, egen programvare, og kan ha egne nettverkskonfigurasjoner og maskinen vil oppføre seg som om den var en egen selvstendig maskin. v, vi

Web Content Accessibility Guidelines Web Content Accessibility Guidelines er et sett med retningslinjer utgitt av World Wide Web Consortium (W3C) som beskriver hvordan man kan gjøre webinnhold mer tilgjengelig for personer med nedsatt funksjonsevne [4]. vii

XSS-angrep Et XSS-angrep [8] er en måte for ondsinnede aktører klarer å injisere ondsinnet kode som kjøres på brukerens nettleser. Dette kan f.eks. brukes til å sende informasjonkapsler til avsender av koden. 10, 49

0.3 Akronymer

API Application Programming Interface. 2, 16, 21, 29, 30, 45, 56, 64,

CRUD Create, Read, Update and Delete (lag ny, les, oppdater og slett). 28, 30, 32, 46, 47, 59,

ER-Diagram Entity Relations diagram. 144,

HTTP Hypertext Transfer Protocol. 10, 16, 20, 29, 30, 36, 44, 46, 49, 55

IaaS Infrastructure as a Service. v,

IDI Institutt for datateknologi og informatikk. 17,

JSON JavaScript Object Notation [31]. 20, 44–47, 55–57, 59, 62, 64

JWT JSON Web Token [34]. 47,

KI Kunstig intelligens. xvi, 75–78,

MVP Minimum Viable Product. 11, 70,

PaaS Platform as a Service. v,

PWA Progressiv Web App. 37

RFC Requests for Comments. iii,

SaaS Software as a Service. v,

SQL Structured Query Language. 20, 30, 32, 33, 35, 44, 46, 51, 52, 59, 60,

TLS Transport Layer Security. 10, 16, 45, 49,

URL Uniform Resource Locator[39]. 37

WCAG Web Content Accessibility Guidelines. 10,

Innhold

0.1	Forord	iii
0.2	Ordforklaring	iii
0.3	Akronymer	vi
1	Introduksjon	1
1.1	Litt om Utstyrsbanken	1
1.2	Problemområde	1
1.3	Gruppens bakgrunn	1
1.3.1	Motivasjon	2
1.4	Mål	2
1.4.1	Effektmål	2
1.4.2	Resultatmål	3
1.4.3	Læringsmål	3
1.5	Rammer og avgrensning	3
1.6	Inspirasjoner	4
1.7	Målgruppe	4
1.8	Prosjektorganisering	4
1.8.1	Roller	4
1.8.2	Rutiner og gruppereregler	5
1.8.3	Faste møter	5
1.9	Om rapporten	6
1.9.1	Om språket i rapporten	6
1.9.2	Rapportens struktur	6
2	Kravspesifikasjon	7
2.1	Funksjonelle krav	7
2.1.1	Utstyrstyper	7
2.1.2	Lån og reservasjoner	7
2.1.3	Levering	8
2.1.4	Brukere	8

2.1.5	Administrative funksjoner	8
2.1.6	Konfigurerbar verdier	9
2.1.7	Åpningstider	9
2.1.8	Statistikk	9
2.1.9	Hendelseslogg	9
2.1.10	Notifikasjoner	10
2.2	Ikke-funksjonelle krav	10
2.2.1	Operasjonelle krav	10
2.2.2	Sikkerhetskrav	10
2.2.3	Grensesnittskrav	10
3	Metodikk og prosess	11
3.1	Prosjektrammeverk	11
3.2	Framdriftsplan	11
3.3	Wireframes	13
4	Teknologivalg	14
4.1	Valg av databaser	14
4.1.1	Bildelagring	14
4.2	Valg av frontend	15
4.3	Valg av backend	16
4.3.1	Tredjeparts-bibliotek og tjenester i backend	16
4.4	Andre teknologier og verktøy	17
4.4.1	GitLab	17
4.4.2	Docker	17
4.4.3	Docker Compose	18
4.4.4	SkyHigh	18
4.4.5	Discord	18
4.4.6	Google Drive	18
4.4.7	IDE	19
4.4.8	phpMyAdmin	19

4.4.9	Figma	19
4.4.10	Draw.io	19
5	Design	20
5.1	Programvarens komponenter	20
5.2	Systemarkitektur	21
5.3	Design av frontend	21
5.3.1	React	22
5.3.2	Inventarside	22
5.3.3	Reservasjonslogg	23
5.3.4	Utstyrside	25
5.3.5	Adminside	26
5.4	Bruker- og innloggingssystem	26
5.4.1	Sletting av brukere	27
5.4.2	Sesjoner	28
5.5	Tilgangskontroll	28
5.6	Åpningstider	28
5.7	Design av REST API	29
5.8	Databasedesign	30
5.8.1	Normalisering	30
5.8.2	Lån og reservasjoner	30
5.8.3	Utstyrstyper og størrelser	31
5.8.4	Brukere, roller og ressurser	32
5.8.5	Historikk og statistikk	33
5.8.6	Hendelseslogg	34
5.8.7	SQL events	35
6	Implementasjon	36
6.1	Frontend-spesifikke temaer	36
6.1.1	Kodestruktur	36
6.1.2	Datastrukturer og Zod	36

6.1.3	Routing	37
6.1.4	Håndtering av state	38
6.1.5	Asynkron programmering	39
6.1.6	Kopiering av verdier	43
6.2	Backend-spesifikke temaer	44
6.2.1	Inndeling av kildekode	44
6.2.2	Konfigurasjon	45
6.2.3	Standardprosess	46
6.2.4	Autentisering og autorisering	47
6.2.5	Om passord	49
6.2.6	Sikring av informasjonskapsler	49
6.2.7	Godkjenning av datoer	50
6.2.8	Bruk av prepared statements i SQL-spørringene	51
6.2.9	SQL-transaksjoner	52
6.2.10	Rate limiting	53
6.2.11	Feilhåndtering	55
6.2.12	Håndtering av bilder	56
6.3	Registrering av brukere	57
6.4	Oppretting av reserverasjoner og lån	57
6.5	Oppdatering av inventaret	59
6.6	Statistikk	60
6.7	Datohåndtering	61
6.7.1	I frontend	61
6.7.2	I backend	62
7	Kvalitetssikring	64
7.1	Code review og merge requests	64
7.2	Dokumentasjon	64
7.3	Testing	64
7.3.1	Brukertesting	64
7.3.2	Stresstesting	65

7.3.3	Automatiserte tester i backend	65
7.4	Andre krav til kildekoden	65
8	Utrulling	66
8.1	CI/CD	66
8.2	GitLab runneren	67
9	Diskusjon	69
9.1	Valg av dato	69
9.2	Utlånssystem	69
9.3	Overholding av personopplysninglover	69
9.4	Avvik fra framdriftsplan	70
9.4.1	Rutiner	70
9.4.2	Kutt av funksjonalitet	71
9.5	Videre utvikling	71
9.5.1	Av frontend	71
9.5.2	Av backend	71
9.6	Tidsbruk	72
9.7	Grupesamarbeid	73
9.8	Bærekraft	74
9.9	KI og oppgaven	75
9.9.1	KI og fremtiden	78
10	Avslutning	79
10.1	Egen vurdering av resultat	79
10.2	Konklusjon	79
	Referanser	80
A	Prosjektplan	83
B	Standardavtale	102
C	Use cases	109

D	Guide for Brukbarhetstest	141
D.1	Testing	141
D.2	Hvorfor brukbarhetstest	141
D.3	Hva vi ønsker å måle	141
D.4	Hvem vi ønsker skal teste	141
D.5	Test data	141
D.6	Introduksjon	142
D.7	Hoveddel	142
D.8	Avslutning	142
D.9	Eksempler på Oppgaver	142
D.10	Iterasjoner	143
D.11	Analyse i ettertid	143
E	ER-diagram	144
F	Detaljert tidsbruk	146
F.1	Per gruppemedlem	148
F.1.1	Erlend	148
F.1.2	Noah	151
F.1.3	Åsulv	155
F.1.4	Ferdinand	159
F.2	Per uke	163
G	Kodeutdrag: Backend	165
G.1	Kodeeksempel: Handler-funksjon	165
G.2	Kodeutdrag: Finn gyldige datoer	166
G.3	Kodeutdrag: Parsing av en multipart form	167
H	SQL-kode: Events	169
H.1	Fjerning av ugyldige sesjoner	169
H.2	Fjerning av gamle unntaksdatoer	169
H.3	Automatisk kansellering av reserverasjoner	170

I	Utkast til API design	171
J	API dokumentasjon	181
J.1	Autentisering	182
J.2	Inventar og kategorier	184
J.3	Lån og reserverasjoner	189
J.4	Brukerhåndtering	197
J.5	Diverse	202
J.6	Bildehåndtering	208
K	Oppgavebeskrivelse	211

Figurer

1	Rollefordeling, viser både gruppelemmene og eksterne aktører. Figuren tar utgangspunkt i rammeverket Scrum, mer om dette i kapittel 3	4
2	Gantt-diagram som viser plan for fremdrift, hentet fra prosjektplan (se vedlegg A)	12
3	Wireframen viser det første utkastet for inventarsiden	13
4	Et skjermbilde av hvordan issueboardet vårt så ut mot slutten av prosjektet	17
5	Arkitekturskisse (overblikk over servere)	21
6	Wireframen til venstre er første utkastet for inventarsiden, og høyre viser hvordan det ser ut i dag	23
7	Wireframe til venstre, hvordan admin ser siden i midten og hvordan bruker ser siden til høyre	24
8	Sammenlikning mellom wireframen og endelig utstyrsside	25
9	Admin knappene på profilsiden. Dukker bare opp om du er administrator	26
10	Alternativ struktur for lagring av lån	31
11	Valgt design av lån	31
12	Lagring av utstyrstyper	32
13	Hvordan data om brukere, roller og ressurser er lagret	33
14	Denne delen av ER diagrammet viser hvordan hendelser for hendelsesloggen er strukturert etter normalisering	34
15	Et eksempel på et standard interface og type i TypeScript	36
16	Et eksempel på definisjon og bruk av Zod i programmet vårt	37

17	Struktur av Router element	38
18	Et kodeeksempel deklarasjon av state	39
19	Diagram som viser initialiseringen av state, og stegene som skjer når det endres	39
20	Et kodeeksempel på en komponent som bruker både state og props	40
21	Modell som demonstrerer forskjellen mellom asynkron og synkron kode	42
22	Asynkron som ikke venter på svar	42
23	Asynkron kode med Promise.all i brukerliste-side for administratorer	43
24	Bruk av en spread operator	44
25	Sekvensdiagram som viser autentisering (med innlogging og evt. henting av ny AccessToken om nødvendig)	48
26	Flytskjema som beskriver godkjeningsprosessen av hentedato for reservasjoner.	50
27	Bruk av "prepared statement" i Go for å hente en sesjon fra databasen. (?) representerer parameteren som brukes når spørringen kjøres med QueryRow-funksjonen.	51
28	Deler av koden som henter data om flere brukere fra databasen. Koden er noe modifisert fra opprinnelig kildekode.	52
29	Figur som beskriver tokenbøtte-algoritmen. Hentet fra [46]	53
30	Rate limiting med rate-pakken i sin enkleste form.	54
31	Typisk feilhåndtering i Go	55
32	LogError, brukes til å logge de fleste feilmeldinger som oppstår både til konsoll og til klient	56
33	Flytskjema som beskriver sjekkene for å lage en ny reservasjon, for nærmere beskrivelse av sjekkene på datoene se kapittel 6.2.7	58
34	Utdrag fra hjelpefunksjonen som finner nye og slettede elementer i en array med positive heltall (uint). Disse representerer unike verdier gitt til hver kategori	59
35	SQL-spørring som brukes for å hente antall lån og antall lånt utstyr i perioden. ? representerer start- og sluttdato, disse blir satt inn når spørringen kjøres.	60
36	Når datoene inkluderte et månedskifte, ville datovelgeren alltid ha feil måned på sluttdato	62
37	Henting av TIME-verdier fra databasen. Eksemplet her er innlesning av data om åpnings-tidene.	63
38	Sekvensdiagram for pipelines til frontend (venstre) og backend (høyre)	67
39	Kopi av konfigurasjonsfilen for vår GitLab runner	68
40	GitLab runneren var klar til bruk	68
41	Tidsbruk fordelt på de ulike kategoriene, hentet fra regnearket kl. 14 16/05	72
42	Tidsbruk fordelt på ukene, hentet fra regnearket kl. 14 16/05. Merk: Det mangler data for uke 2 og uke 21	73

43	De fundamentale byggesteinene av Norges Frivilligsentraler	74
44	ChatGPT 3.5 eksempelkode om Promise.all	75
45	ChatGPT 3.5 overkompliserer en simpel løsning, der man vil lese inndata fra bruker . . .	76
46	Eksempel på en lettere løsning som leser inn telefonnummer og søker når man trykker enter	77
47	KI for repetitive oppgaver	77

1 Introduksjon

Dette kapitlet introduserer oppgaven, hvilke mål som settes for prosjektet samt organiseringen av gruppen.

1.1 Litt om Utstyrskassen

En Utstyrskasse er en ordning som tilbyr gratis utlån av utstyr til ulike sporter og friluftsliv omtrent som et bibliotek. Tilbudet er ment som et velferdstilbud og gir alle muligheten til å delta på ulike aktiviteter uavhengig av om man har økonomien til å kjøpe inn nødvendig utstyr eller ikke. Ordningen organiseres ofte av en av frivilligsentralene ofte i tett samarbeid med den lokale kommunen.

Norges frivilligsentraler er en interesseorganisasjon for landets frivilligsentraler og definerer begrepet slik ”Frivilligsentraler er mangfoldige møteplasser bygget på frivillighetens premisser og lokale ønsker, ivaretatt av ansatte i samhandling med frivillige, lag & foreninger og kommune.” [16]. Frivilligsentraler tilbyr en rekke ordninger til gode for lokalsamfunnet og skal eksistere som et supplement til kommunens tilbud og ordninger. Ordningen med en Utstyrskasse faller naturlig innenfor hva frivilligsentralene kan tilby da det er til god hjelp for lokalsamfunnet.

En slik Utstyrskasse finnes i Raufoss i Innlandet fylke og driftes av Vestre Toten Frivilligsentral. Tilbudet her blir særlig brukt av de lokale skolene som ligger like i nærheten av Utstyrskassens lokaler.

*Videre i rapporten er det Utstyrskassen i Raufoss/Vestre Toten som menes når begrepet **Ustyrskassen** brukes.*

1.2 Problemområde

I lang tid har Utstyrskassen benyttet papirbaserte skjemaer for å holde orden på hvem som låner hvilket utstyr, dette er veldig upraktisk og mye tid brukes for å holde orden på dette. Det er også vanskelig å ha oversikt over hvilket utstyr som til enhver tid er tilgjengelig for utlån. Dette gjør at dersom for eksempel en gruppe ringer inn og ønsker å låne utstyr må de ansatte se igjennom utstyret fysisk i lokalet for å finne ut om det er nok utstyr til gruppa. Det betyr også at dersom man ønsker å låne spesifikt utstyr må man enten ringe Utstyrskassen og få de til å sjekke eller møte opp fysisk og da potensielt ende opp tomhendt.

Det har derfor oppstått et behov for en ny løsning som kan gjøre det enklere å få oversikt over inventar og utlån, da helst en digital nettbasert løsning. Dette er noe som andre tilsvarende utlånsordninger som BUA allerede har, men av ulike grunner har det ikke vært ønskelig for Utstyrskassen å bli med i BUA. Dermed tok de kontakt med NTNU og vi endte opp med oppgaven om å utvikle en egen løsning for Utstyrskassen.

Løsningen skal kunne gi en oversikt over utstyret som er tilgjengelig i Utstyrskassen for øyeblikket, la folk reservere utstyr i en bestemt periode og ansatte må kunne få en oversikt over hva som skal utleveres og hva som skal leveres tilbake. I tillegg må de ansatte kunne få en oversikt over lån som er på overtid. Løsningen skal være fleksibel nok slik at administrator kan oppdatere inventaret etter hvert som utstyr kommer inn i Utstyrskassen og gammelt utslitt utstyr skrutes.

1.3 Gruppens bakgrunn

Gruppemedlemene har kompetanse om en del fagområder som er relevant for oppgaven, disse kommer hovedsaklig fra emner i tidligere semestre ved studieprogrammet Bachelor i Programmering. Relevante

emner er blant annet Cloud Technologies ¹ for kunnskap om API-er og utviklingen av disse, sentralt for en moderne webapplikasjon, Software Security ² for kunnskap om vanlige sikkerhetsutfordringer som webapplikasjoner møter samt Datamodellering og databasesystemer ³ for kunnskap om ulike typer databaser og hvordan man designer en databasestruktur som løser datalagringsbehovene for applikasjonen.

Å bygge en moderne webapplikasjon vil kreve mer kunnskap om frontendutvikling enn det gruppe medlemmene fikk i emnet Webteknologier ⁴, dette må derfor læres underveis.

1.3.1 Motivasjon

Når vi skulle velge hvilken oppgave vi ville ta for oss blant de mange ulike oppgaveforslagene vi kunne velge så var flere på gruppen i utgangspunktet interessert i en mer praktisk utviklingsoppgave. Mer spesifikt noe som involverer en **fullstack-utvikling**-applikasjon. Dette var fordi vi ønsket en oppgave som tilnærmet seg et prosjekt man kunne ha fått i arbeidslivet. I tillegg var det ønskelig for gruppen å lære mer om **fullstack-utvikling** og dets vanlige komponenter. Det var derfor liten interesse for de mer teoretiske og forskningsbaserte oppgaveforslagene. Dermed, fant gruppen blant oppgaveforslagene som passet kriteriene, utviklingen av et utlånssystem til Utstyrskontoret.

1.4 Mål

I prosjektplanen (se vedlegg A) skrev vi ned effektmål, resultatmål, og læringsmål. Målene satt her er basert på målene fra prosjektplanen. Effektmålene viser hva slags endringer vi ønsker å oppnå med lansering av tjenesten. Resultatmål er hvilke konkrete resultater vi ønsker å oppnå med selve prosjektet. Læringsmål viser til hva vi som gruppe ønsker å lære som resultat av prosjektjobbingen

1.4.1 Effektmål

Det overordnede effektmålet med tjenesten er å gjøre det lettere for folk i nærheten av Vestre Toten til å låne utstyr. Ved å gjøre det mer synlig på hva slags utstyr Utstyrskontoret tilbyr, er det forventninger at dette fører til økt bruk av tilbudet enn når den gamle analoge løsningen var i bruk.

Tallene her er kun estimeringer av hvor store effekter applikasjonen kan ha og det bør ikke legges for mye vekt på nøyaktigheten av tallene. Effektmålene i seg selv er dog reelle.

- Med tilgang på statistikk vil det bli enklere for Utstyrskontoret å prioritere hvilket utstyr som bør kjøpes inn.
- Seks måneder etter den nye løsningen er tatt i bruk skal antall lån per måned ha gått opp med 30% sammenlignet med tiden med analog løsning, som følge av at tilbudet har blitt enklere å benytte
- Antallet brukere av Utstyrskontoret øker med 25% tre måneder etter at løsningen er tatt i bruk
- De ansatte ved Utstyrskontoret skal etter den nye løsningen er satt i drift bruke 70% mindre tid på å få oversikt over tilgjengelig utstyr og hva som til enhver tid er utlånt enn da de måtte gå gjennom papirskjemaene.

¹PROG2005 - Cloud Technologies

²IIKG2001 - Software Security

³IDATG2204 - Datamodellering og databasesystemer

⁴PROG2053 - Webteknologier

1.4.2 Resultatmål

- Ferdig produkt etter endt prosjekt skal være en webapplikasjon som oppfyller kravene satt i kravspesifikasjonen.
- Tjenesten skal gjøre det enklere for personalet til Utstyrskontrollbanken å få en oversikt over hvilket utstyr de har på lager, og hvilke lån som har gått over fristen.
- Applikasjonen skal inneholde et minimum av bugs og følge relevant standard for brukergrensesnitt.
- Kildekode ved prosjektets slutt skal være organisert og dokumentert på en slik måte å gjøre det enkelt å videreutvikle ferdig produkt.
- Utrulling av applikasjonen bør være fleksibel nok til å støttes på de fleste plattformer, spesielt de som tilbys av [skytjenester](#)

1.4.3 Læringsmål

- Lære mer om [fullstack-utvikling](#), og hvordan å utvikle en fullstack applikasjon (frontend, backend, database)
- Øke erfaring med smidig metodikk, som Scrum
- Få mer erfaringer med versjonskontrollsystemer som Git for mer profesjonell utvikling
- Mestre prosessen som involverer bruk av kontinuerlig integrasjon og kontinuerlig utrulling (CI/CD)
- Forbedre produktet gjennom brukertester
- Lære om hvordan å samarbeide og utvikle en programvare med en ekstern oppdragsgiver og kunne definere funksjonelle krav utifra kommunikasjon med oppdragsgiver

1.5 Rammer og avgrensning

En av de største rammene til prosjektet er fristene til bacheloroppgaven satt av universitetet. Med planleggingen i januar betydde det at vi hadde i praksis tre måneder altså perioden februar-april til å utvikle programvaren.

En annen ramme er at applikasjonen skal overholde relevant lovverk knyttet til blant annet brukergrensesnitt (WCAG) og legge til rette for at Utstyrskontrollbanken skal overholde lovene rundt personvern.

Etter ønske fra oppdragsgiver må programvaren være en progressiv web app, men vi sto fritt til å velge programmeringsspråk og andre teknologier.

Det er ikke nødvendig for applikasjonen å kunne identifisere individuelle gjenstander, applikasjonen skal forholde seg til [utstyrstyper](#) som for eksempel langrennsski og tursekker istedetfor.

Etter samtale med oppdragsgiver skulle vi ikke være ansvarlig for å sette applikasjonen i drift på en av de mange skytilbyderne. Vi skulle heller ikke ha ansvaret for å legge inn dagens inventar i Utstyrskontrollbanken inn i databasen.

1.6 Inspirasjoner

Applikasjonens hovedinspirasjon er løsningen til BUA [2]. Dette er naturlig siden BUA er en svært lignende ordning med utlån av utstyr og deres løsning dekker derfor mye av det samme som vår applikasjon skal gjøre. Den største forskjellen er at vår applikasjon ikke trenger å forholde seg til mer en ett utlanssted da BUA er et nettverk av flere *Utstyrsentraler* som de har valgt å kalle det. Ellers har vi også tatt inspirasjon fra andre utlånstjenester som biblioteker.

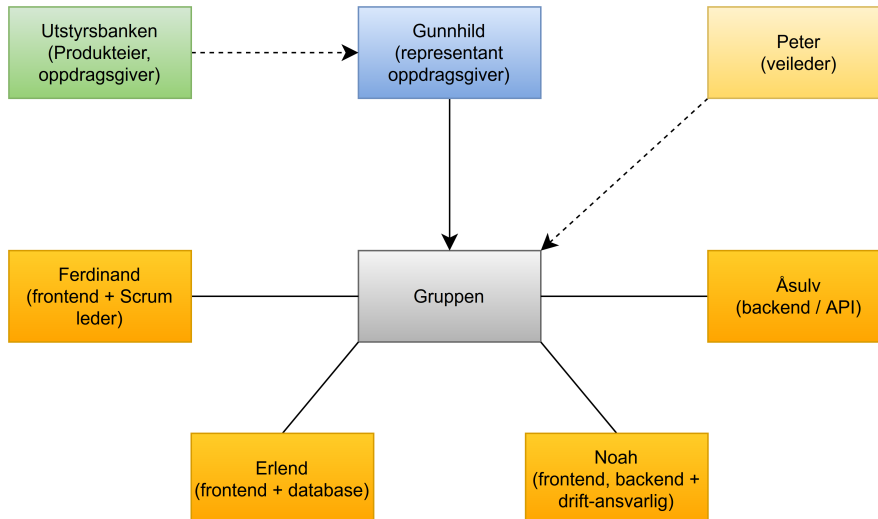
1.7 Målgruppe

Målgruppen for endelig produkt er brukere av utlånstjenesten til Utstyrsbanken samt ansatte ved Utstyrsbanken. Brukerne vil i hovedsak være innbyggerne i Vestre Toten kommune som er interesserte i å låne utstyr. De ansatte vil måtte registrere utlånsaktiviteten ved Utstyrsbanken via tjenesten.

Målgruppen for selve rapporten er først og fremst sensorene som skal sensurere bacheloroppgaven vår, men også alle som er interesserte i utviklingsprosessen av en fullstack-applikasjon. Det forventes at man har noen grunnkunnskaper om IT og datateknologi inkludert kunnskap om relasjonelle databaser samt engelsk for å kunne forstå alle aspekter ved denne rapporten.

1.8 Prosjektorganisering

1.8.1 Roller



Figur 1: Rollefordeling, viser både gruppemedlemmene og eksterne aktører. Figuren tar utgangspunkt i rammeverket Scrum, mer om dette i kapittel 3

Rollefordelinga mellom gruppemedlemmene endte opp med å bli slik:

- **Ferdinand:** Utvikling av frontend, Scrum-leder, fast møtereferent
- **Erlend:** Design av database, utvikling av frontend, vara møtereferent

- **Noah:** Cloud/deployment-ansvarlig, ansvarlig for kontakt med oppdragsgiver og rombestilling til møter, hjelpe til med utvikling av backend og frontend
- **Åsulv:** Utvikling av backend og API

Dette betyr at hvert gruppelem har hvert sitt ansvarsområde av applikasjonen som blir utviklet som de kunne fokusere på. Det legger også opp til at applikasjonens ulike deler utvikles parallelt.

Andre roller utenfor selve gruppen:

- **Peter:** Veileder gjennom hele prosjektet, faste møter hver uke for å få råd til videre jobbing med prosjektet
- **Gunnhild:** Vår representant fra Utstyrskomiteen, altså oppdragsgiver. Gunnhild er daglig leder for Vestre Toten frivilligsentral som drifter Utstyrskomiteen.

1.8.2 Rutiner og grupperegler

Rutinene og gruppereglene er hentet fra prosjektplan, se vedlegg A

Mål for tidsbruk til prosjektet er 30 timer i uka per person. Tidsbruken skal loggføres, og hvert gruppelem har selv ansvar for å loggføre sin egen timebruk. Det forventes at hver person jobber med oppgavene de er tildelt. Dersom man trenger hjelp bør man ta kontakt med de andre gruppelemmene.

Dersom fravær planlegges, meld ifra til resten av gruppa minst 1 dag før. Dersom eventuelle forsinkelser oppstår på vei til møte (f.eks. sen buss/tog), meld ifra via Discord eller telefon så fort som mulig.

Dersom en person ikke møter opp, men er forventet å møte opp, går det greit dersom det er snakk om engangstilfellet eller særlige gode grunner. Dersom det skjer gjentatte ganger over tid blir det et problem som må tas opp med veileder.

Det forventes at gruppelemmene skal være tilgjengelig til å svare på meldinger på Discord eller mobil mellom kl. 09:00-18:00 hver ukedag. De faste møtene har møteplikt.

Når det gjelder uenigheter i gruppen er det i utgangspunktet flertallet som bestemmer. Dersom vi fortsatt ikke kommer til en beslutning er det den personen som har hovedansvar ved området som har siste ord (eksempelvis front-end ansvarlig har siste ord i front-end uenigheter). Dersom det oppstår større problemer tas det opp med veileder.

1.8.3 Faste møter

I løpet av prosjektets varighet er det planlagt faste møter. Noen av møtene kjøres fysisk på campus mens andre tas digitalt over nett. Grunnen til at vi har valgt å gjøre det slik er på grunn av at en av gruppelemmene har en lang reisevei til Gjøvik.

Hver mandag har vi faste møter internt i gruppa, dette er for å oppdatere hverandre på hva som har blitt gjort siden sist samt ta opp og diskutere temaer vi er usikre på. Mandagsmøtene brukes også annenhver uke for Sprint Review og Sprint Planning, mer om det i kapittel 3.

Hver tirsdag og onsdag har vi faste standupmøter der vi oppdaterer hverandre på hva som jobbes med for øyeblikket. Siden disse møtene er mindre viktigere og er av kortere varighet, tas de digitalt.

Hver torsdag har vi møter med veileder der vi tar opp eventuelle spørsmål vedrørende prosjektet som dukker opp underveis.

Hver fredag har vi et lignende møte som på mandag, men annenhver uke har vi møte med oppdragsgiver istedetfor. Ved møter med oppdragsgiver handler det om å holde oppdragsgiver oppdatert på fremgangen på prosjektet samt få tilbakemeldinger og stille spørsmål underveis.

De faste møtene skal sikre jevn progresjon og at alle på gruppa er på samme side, spesielt viktig er møtene med oppdragsgiver slik at vi kan sikre at applikasjonen som utvikles passer med behovene og ønskene til oppdragsgiver.

1.9 Om rapporten

I digital PDF utgave av rapporten vil man kunne klikke på kapitler i innholdslisten, referanser til ulike kapitler og vedlegg, kildehenvisninger og ordforklaringer for å navigere direkte dit.

Enkelte temaer og konsepter dukker opp i flere hovedkapitler, dette er fordi disse har særlig relevanse i flere hovedkapitler.

1.9.1 Om språket i rapporten

Hovedspråket til rapporten er norsk bokmål. Deler av rapporten vil være på engelsk. Dette gjelder blant annet funksjonsnavn, variabelnavn og kommentarer fra utdrag fra kildekoden. Noen begreper relatert til IT vil også bli direkte satt inn i teksten på engelsk. Dersom man er usikker på enkeltord og forkortelser, vil en forklaring av mange av ordene finnes i kapittel 0.1 - ordforklaring.

1.9.2 Rapportens struktur

Rapporten er inndelt i 10 ulike kapitler, her er en oversikt over de ulike kapitlene:

- **Introduksjon:** Introduserer oppgaven inkludert mål og avgrensninger samt hvordan gruppeprosjektet ble organisert.
- **Kravspesifikasjon:** Beskriver konkrete krav som ferdig utviklet produkt skal oppfylle. Dette inkluderer blant annet funksjonaliteten til programvaren samt krav til drift og sikkerhet.
- **Metodikk og prosess:** Beskriver hvordan gruppen har arbeidet for å oppnå målene som er satt. Det vil si blant annet hvilket rammeverk som er brukt samt en fremdriftsplan.
- **Teknologivalg:** Handler om hvilke teknologier og verktøy som ble valgt for å utvikle produktet samt hvorfor disse teknologiene ble valgt.
- **Design:** Beskriver designet til produktet samt alle dets komponenter. Det inkluderer både teknisk design samt design av brukergrensesnitt.
- **Implementasjon:** Beskriver ulike aspekter ved implementasjonen det vil si den utviklede programvaren og selve utviklingsprosessen.
- **Kvalitetsikring:** Ulike tiltak for å sikre at det ferdige produktet holder den kvaliteten som ønskes.
- **Utrulling:** Beskriver prosessen for å få programvaren i drift og klar til bruk.
- **Diskusjon:** Drøfting og refleksjon over ulike temaer knyttet til bachelorprosjektet.
- **Avslutning:** Gruppens vurdering av rapport og ferdig produkt, konklusjon og siste ord.

2 Kravspesifikasjon

Dette kapittelet handler om kravene som ble satt for endelig programvare. Både funksjonelle krav som omhandler funksjonaliteten til applikasjonen og ikke-funksjonelle krav som krav til grensensitt, drift og sikkerhet.

2.1 Funksjonelle krav

De funksjonelle kravene er basert på use-casene vi har bygd opp i løpet av semesteret i møtene med oppdragsgiver. Selve use casene kan bli funnet i vedlegg C. Vi har valgt å ikke inkludere feilhåndtering som funksjonelle krav. Nedenfor følger en oversikt over de funksjonelle kravene til applikasjonen.

2.1.1 Utstyrstyper

Systemet skal kunne vise **utstyrstypene** utstyrsbanken vanligvis har tilgjengelig for lån, selv om alle utstyr av typen er utilgjengelig. Disse **utstyrstypene** skal kunne bli søkt etter og bli filtrert basert på kategorier.

For hver **utstyrstype** skal systemet vise informasjon som:

- Navn på utstyr
- Størrelser og antallet av dem som er tilgjengelig
- Tekstlig beskrivelse
- Bilder
- Kategorier

Systemet skal vise en forside med tekst skrevet av administrator og åpningstider.

2.1.2 Lån og reserverasjoner

- Ansatte og administrator skal kunne se alle reserverasjoner, nåværende lån og ferdige lån.
- Systemet skal ha lån som består av en enkelt størrelse, eieren av lånet, antallet av størrelsen og frist for levering. Hvis det er en reserverasjon, skal det også ha planlagt dato for henting.
- Utlånt utstyr skal ikke kunne bli reservert før det er levert tilbake igjen, selv hvis det ikke er en konflikt i låneperiode.
- Systemet skal la utstyr bli reservert på to måter:
 - At innlogget bruker reserverer for seg selv.
 - Av ansatte for andre brukere.
- Systemet skal automatisk trekke fra antallet av størrelsen av **utstyrstypen** som er igjen på lager når den blir reservert eller lånt ut.
- Brukere skal ha muligheten til å kansellere egne reserverasjoner og ansatte skal kunne kansellere alle reserverasjoner.
- Brukere skal kunne utsette lån og ansatte skal kunne redigere innleveringsfrist for lånene

For henting av lån skal systemet:

- La ansatte i utstyrsbanken bekrefte utlevering av utstyr.
- Automatisk markere reservasjoner som ikke blir hentet på valgt dag .

2.1.3 Levering

Systemet skal håndtere følgende:

- Bare la ansatte i utstyrsbanken bekrefte leveringer.
- La ansatte oppgi antall av utstyr i lån som er mistet eller ødelagt.
- Automatisk øke antallet av størrelsen til *utstyrstypen* når utstyr blir markert som levert
- Automatisk markere lån som ikke blir levert tilbake på valgt dag

2.1.4 Brukere

For registrering og innlogging skal systemet kunne:

- Registrering av ny bruker skal være tilgjengelig for alle som ikke er logget inn.
- Registreringen skal kreve fornavn, etternavn, unikt telefonnummer, passord og valgfritt postnummer.
- Alle brukerkontoer skal verifiseres med en kode på SMS til registrert telefonnummer
- Ansatte skal kunne opprette gjestebrukere som de skal kunne gi lån og reservasjoner til.
- Systemet skal la alle som er logget inn logge ut.

Videre skal systemet ha følgende funksjonalitet for brukere:

- Alle brukere skal kunne oppdatere egen brukerdata som telefonnummeret, fornavnet, etternavnet, og postnummeret.
- Ved endring av telefonnummer skal brukerkontoen verifiseres på nytt.
- Administrator og ansatte skal kunne skrive kommentarer på brukere
- Administrator skal kunne slette brukere

2.1.5 Administrative funksjoner

Administrator krever tilgang til funksjonaliteter som er nødvendige for drift av systemet

Systemet skal la administrator opprette, slette, og endre på alt fra kategorier til *utstyrstyper*. Den følgende listen bør implementeres:

- Legge til, fjerne, og endre på kategorier.

- Endre navn, beskrivelse, og rekkefølgen på utstyr.
- Endre maks antall av en [utstyrstype](#) som kan bli lånt.
- Legge til og fjerne bilder.
- Legge til, endre og slette størrelser på utstyr.
 - Inkluderer å endre rekkefølgen størrelsene vises

Systemet skal også ha funksjonalitet for å promotere registrerte brukere til ansatte, og motsatt.

2.1.6 Konfigurerbar verdier

Systemet skal la administrator modifisere verdier knyttet til varighet av lån og reserverasjoner. Noen eksempler på disse:

- Lengde utstyr kan bli lånt sammenhengende i et lån
- Hvor lang fram i tid man kan reservere
- Lengde på utvidelse

2.1.7 Åpningstider

Systemet skal gi administrator tilgang til redigering av åpningstider for henting og levering av utstyr. Åpningstiden består av et klokkeslett for åpning og et klokkeslett for stengning. Denne redigeringen skal kunne bli gjort via to metoder:

- Redigering av åpningstider for ukedager fra mandag til søndag som gjelder hver uke.
- Redigering av åpningstider for spesifikke datoer som gjelder i stedet for tidene for ukedagen det er på.

Systemet skal automatisk flytte reserverasjoner og lån som skulle hentes eller leveres på en dag som deretter ble stengt til en ny dato og eier av lån/reserverasjon skal bli varslet.

2.1.8 Statistikk

Systemet skal vise statistikk som viser popularitet av ulike kategorier, [utstyrstyper](#) og størrelser til administrator, selv for de som har blitt slettet

2.1.9 Hendelseslogg

Systemet skal gi oversikt over hendelser i systemet med informasjon om:

- Brukerne som forårsaker dem
- Klokkeslettene og datoene det skjedde på
- Informasjon om hendelsen

2.1.10 Notifikasjoner

Systemet skal gi notifikasjoner om reserveringer er klare for henting og lån når de trengs å leveres

2.2 Ikke-funksjonelle krav

Ikke-funksjonelle krav deles inn i tre underkategorier, operasjonelle krav som beskriver kravene til programvaren i drift, sikkerhetskrav som beskriver krav til applikasjonens sikkerhet hovedsaklig knyttet til håndtering av data samt grensesnittkrav som beskriver krav til brukergrensesnittet samt grensesnittet mellom applikasjonens komponenter.

2.2.1 Operasjonelle krav

Disse operasjonelle kravene er utformet med bakgrunn i at oppdragsgiver ønsket at siden skulle være raskere enn konkurrenten BUA.

- Siden bør ha en opptid på 99% målt månedlig (maksimalt 7 timer nedetid per måned)
- Siden skal kunne håndtere perioder med opptil 200 aktive brukere samtidig uten at responstiden økes med mer enn 2 sekunder sammenlignet med ordinær trafikk (omtrent 50 aktive brukere)
- Total svartid på inventarsiden ved ordinær trafikk skal ikke overstige 2 sekunder

2.2.2 Sikkerhetskrav

- Alle brukerkontoer skal verifiseres slik at det vil kun være en brukerkonto per person. Det er også ønskelig at man får verifisert kontaktinformasjonen til brukeren.
- All kommunikasjon mellom klient og server samt servere seg imellom skal bruke [HTTPS \(HTTP over TLS\)](#) for å sikre at ingen utenforstående kan fange opp og/eller endre kommunikasjonen.
- Alle passord skal lagres med bruk av en [hash-algoritme](#) slik at de ikke er direkte lesbare dersom for eksempel dataene i databasen kommer på avveie.
- Tiltak skal innføres for å beskytte applikasjonen mot vanlige web-angrep som [CSRF-angrep](#), [XSS-angrep](#), og [brute force-angrep](#).

2.2.3 Grensesnittkrav

- Brukergrensesnittet skal følge de 35 [WCAG](#)-kravene som bestemt i lovverket [28].
- Brukergrensesnittet skal være intuitivt og lett å bruke, særlig gjelder dette den delen som rettes mot Utstyrskontos brukere (altså lånerne).
- Det skal brukes et standardisert dataformat for kommunikasjon mellom programvarens komponenter.

3 Metodikk og prosess

3.1 Prosjektrammeverk

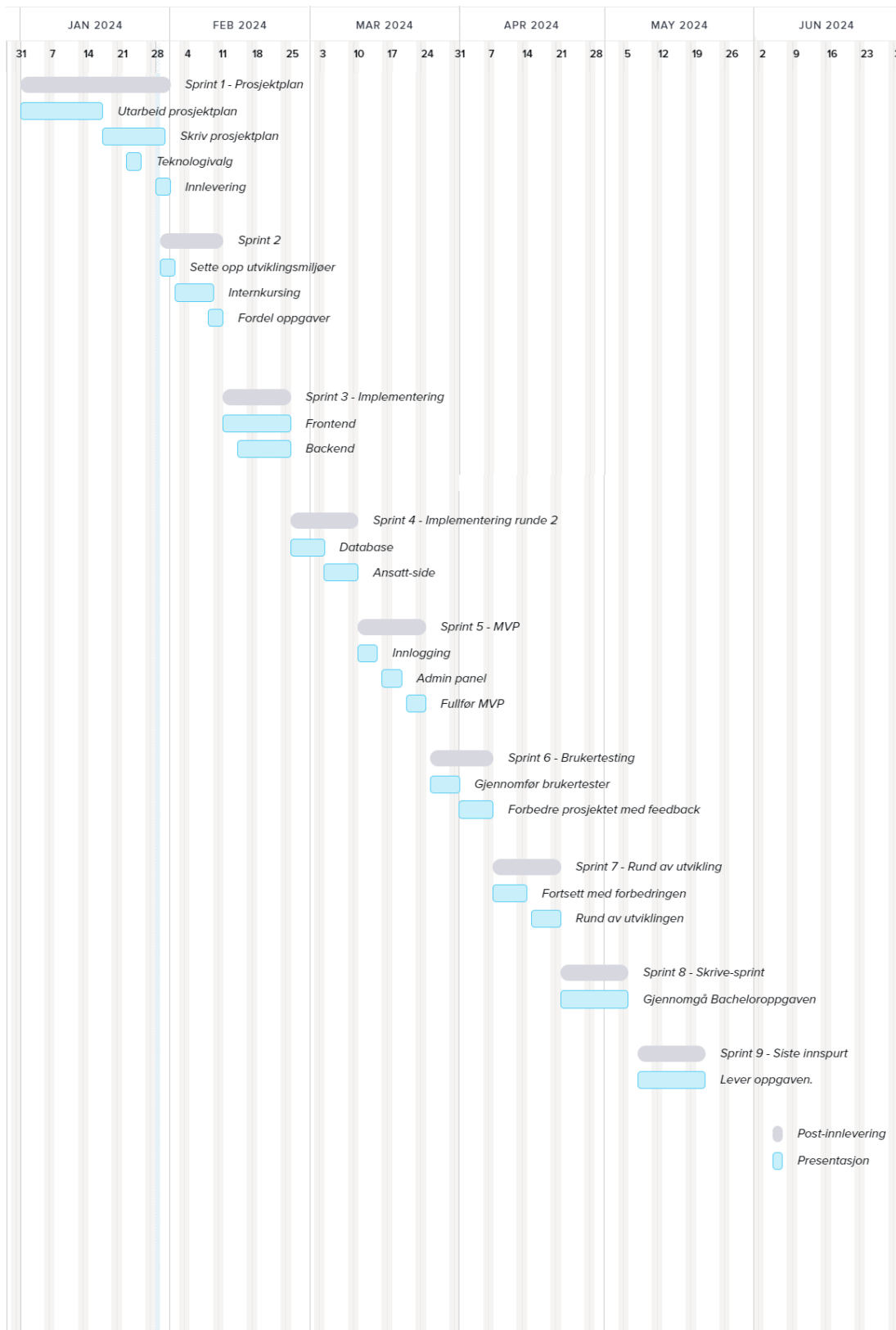
Rammeverket vi valgte for å organisere prosjektet er Scrum. Scrum [45] er en smidig metodikk der man deler inn prosjektiden i faste korte perioder kalt sprinter på under en måned i varighet. Hver sprint starter med et møte (kalles *Sprint Planning*) der man velger ut arbeidsoppgaver fra listen over alle oppgaver som må gjøres (kalles *Product Backlog*) og fordeler det mellom gruppemedlemmene. Deretter jobbes det med arbeidsoppgavene utover i resten av sprinten. Ved sprintens slutt møtes gruppemedlemmene på nytt og ser igjennom status på alle oppgavene, her vurderes det hva som skjer med oppgavene som ikke ble fullførte. Skal de nedprioriteres og legges til *Product Backlog* eller skal de videreføres til neste sprint. Slike møter kalles *Sprint Review*.

Som nevnt i kapittel 1, påvirker Scrum-rammeverket også rollene i gruppen da når man benytter dette rammeverket er det vanlig å ha en leder innad i gruppen kalt *Scrum Master* og en *Product Owner* som da i praksis er oppdragsgiver.

Når det gjelder sprintlengde på vårt prosjekt valgte vi to uker da det gav en fin balanse mellom muligheter til å omprioritere arbeidsoppgavene og hvor mye tid som brukes på møter og organisering i forhold til tid som brukes på utvikling. En kortere sprintlengde ville ført til at vi lettere kunne omprioritere arbeidsoppgavene, det er også lettere å estimere hvor mye man rekker på en kortere tidsperiode. Hovedulempen med kortere sprintlengde er at man må bruke mer tid på å avholde *Sprint Planning*- og *Sprint Review*-møtene som tar tid bort fra å faktisk jobbe med arbeidsoppgavene.

3.2 Framdriftsplan

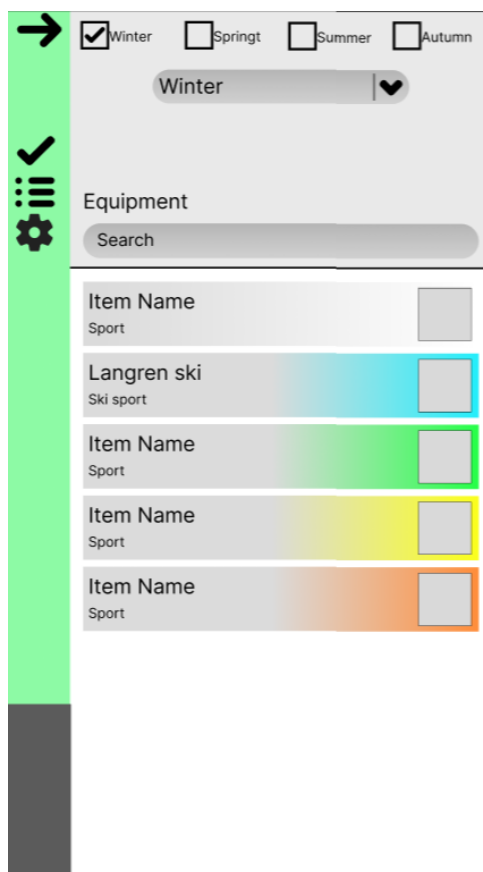
I prosjektplanen lagde vi et Gantt-diagram (se figur 2), som vi brukte som en pekepinne gjennom hele prosjektet. I Gantt-diagrammet gir vi en oversikt over hvordan vi ønsker at progresjonen skal være i løpet av prosjektet ved å gi en rask beskrivelse av hva som skal gjøres hver sprint. Noen viktige milepæler er fullføring av MVP (ved slutten av uke 12), brukertest (rundt påsketider, ved slutten av uke 14) og overgang fra utvikling til rapportskrivning (ved slutten av uke 16). Siden vi bruker smidig metodikk kan planene endre seg underveis ved behov, men det er likevel nyttig å ha en overordnet plan.



Figur 2: Gantt-diagram som viser plan for fremdrift, hentet fra prosjektplan (se vedlegg A)

3.3 Wireframes

Wireframes var en nyttig redskap for kommunikasjon, både mellom oss på gruppen men mest av alt mellom oss og oppdragsgiver. I starten hadde vi mye problemer med kommunikasjon, de første kravspesifikasjonene vi fikk fra oppdragsgiver var at de hadde lyst på en side som BUA bare raskere og mindre tomrom mellom objekter på siden. Dette var vanskelig for oss å tolke, men etter vi fikk fikset et første utkast av hvordan siden kunne se ut ble det lettere å komme til enighet mellom både funksjonalitet og design. Senere i kapitlet for frontend design (kapittel 5.3) vil vi se på endringene vi gjorde som følge av wireframen.



Figur 3: Wireframen viser det første utkastet for inventarsiden

4 Teknologivalg

Dette kapitlet tar for seg hvilke teknologier (programvare, programmeringsspråk samt biblioteker) vi har valgt å bruke for programvaren samt utviklingen av den. Kapitlet omhandler også alternativer til teknologien vi valgte og hvorfor vi endte opp med å velge den teknologien vi gjorde.

4.1 Valg av databaser

Ett av de større valgene vi har gjort i løpet av semesteret er hvilken database-teknologi som er aktuell å bruke for prosjektet. Det finnes svært mange ulike typer databaser, i hovedsak deles de i blant annet SQL-baserte databaser (også kjent som relasjonelle databaser), nøkkel-verdipar databaser, dokument-baserte databaser og graf-databaser. Det tradisjonelle valget er en SQL-basert database, slike databaser fungerer godt for data som er knyttet sammen. I vårt prosjekt kan dette illustreres med at et lån er knyttet til en bruker og til en størrelse av en *utstyrstype*. SQL-baserte databaser har også et språk for spørringer. Det gir oss muligheten til å blant annet enkelt kombinere og hente relatert data som å kunne hente navnet til låner sammen med data om selve lånet. På bakgrunn av dette virker en SQL-basert database som det rette valget for vårt prosjekt.

Innenfor SQL-baserte databaser finnes det flere ulike programvarer. En av disse vi vurderte å bruke var SQLite. SQLite har en del unike elementer ved seg sammenliknet med andre SQL-baserte relasjonsdatabaser som gjorde at SQLite virket interessant. Blant disse er:

- SQLite lagrer hele databasen i en fil, for øvrig ikke det samme som dokument-baserte databaser. Dette gjør det svært enkelt å sikkerhetskopiere databasen.
- Backend kan samhandle direkte med databasefilen via en pakke som bruker SQLite, man trenger altså ikke å kjøre en egen serverprogramvare for å bruke databasen. Dette er i teorien ressursbesparende, da man i utgangspunktet kjører slik serverprogramvare på en egen (virtuell) maskin.
- SQLite skalerer svært dårlig, dersom det blir ønskelig å kjøre flere instanser av backend på grunn av økt trafikkmengde kan man ikke enkelt koble de opp mot en eller flere databaseservere. Dette er på grunn av at SQLite-databaser er en fil og flere backend instanser kan ikke få tilgang på samme fil samtidig. Dermed vil man bli tvunget over på en annen databaseprogramvare så fort det er mulig.
- SQLite har veldig få typer for dataene som skal lagres. Det finnes nok typer til det man i teorien trenger for en database (string/tekst, heltall, flyttall, blob (binærdata)), men typer for dato for eksempel, mangler. Det ville ha betydd at backend må konvertere mellom sin datatype og antageligvis string i databasen. Dermed mister vi muligheten for at databasen kan validere dataene.

På bakgrunn av ulempene til SQLite som skalering og mangel på datatyper gikk vi ikke for SQLite, men heller MySQL. MySQL er en mer avansert SQL-basert databasesystem som kjøres som en egen server. MySQL har støtte for alle datatypene vi trenger og lar flere klienter aksessere databasen samtidig.

Vi kunne også ha vurdert å bruke PostgreSQL som har den fordel at man kan sammenkoble flere instanser for å skalere opp trafikkmengden som kan håndteres, men ulempen er at PostgreSQL er noe mer ressurskrevende sammenliknet med MySQL. Siden vi for øyeblikket ikke så behovet for skalering, valgte vi bort PostgreSQL. [47]

4.1.1 Bildelagring

For å få siden til å bli mer presenterbar er det viktig å kunne inkludere bilder, dette krever at vi har et sted å lagre bildene som blir lastet opp. I tillegg må vi lagre noe metadata knyttet til bildene. Blant

annet slik at rett bildet vises for hver utstyrstype. Hvert bilde har også en beskrivelse samt en verdi som man kan sortere på for å finne ut av hvilken rekkefølge man skal vise bildene i dersom en utstyrstype har flere bilder.

Å finne et sted å lagre bildene på ble en liten utfordring. Det finnes flere alternativer, en er å rett og slett lagre bildene i databasen som binærdata (blob). Dette er ikke spesielt vanlig av to årsaker, det fører til økt belastning på databasen samt at det er en del overhead ved å bruke databasen til å lagre filer som til slutt ender opp på filsystemet uansett.

Når det gjelder ytelsen har det blitt gjort undersøkelser der man sammenligner ytelsen på fillagring direkte i filsystemet og i en SQL-basert database. I 2006 undersøkte Microsoft [25] fillagring i databasen og fant ut at dersom filene var under 256 KB var det faktisk raskere å bruke databasen mens for filer over 1024 KB var det bedre å bruke filsystemet. Mellom 256 KB og 1024 KB var det ingen klar vinner. En årsak til dette var at prosessen av å åpne filene gikk tregere enn databasespørringene og påvirkningen det hadde var merkbar på de minste filene. En lignende undersøkelse ble gjennomført av folkene bak SQLite [48] og der var terskelen på omtrent 100 KB. Begge undersøkelsene er fra over 10 år siden og mye har skjedd både med prosessorkraft og hastigheten på lagringsmedium. Blant annet testet Microsoft ytelsen med magnetiske harddisker der **fragmentering** av filer kan være en reell flaskehals. De fant ut at filsystemet NTFS var bedre på å redusere **fragmenteringen** sammenlignet med databasen. I dag har vi SSD-er som ikke er påvirket av **fragmentering**. Hvordan situasjonen er i dag er derfor noe usikker. Man kan likevel anta at for større filer vil det være mer effektivt å bruke filsystemet; det vil uansett bety ett mindre steg som forespørslene for å hente bildene må gå igjennom.

Det er likevel en fordel man ville fått ved å lagre bildene i databasen og det er en sterkere tilknytning mellom bildene og metadataen, f.eks. ved sletting av et bilde vil man kunne slette både selve bildet og metadataen med en operasjon. Dermed ville scenariet at det finnes metadata i databasen til bilder som ikke eksisterer lengre eller omvendt blitt unngått. Likevel gikk vi bort ifra å bruke databasen for bildelagring på bakgrunn av ytelsen og måtte derfor se etter en annen løsning.

Det finnes flere ulike steder der bildene kan lagres, man kunne satt opp en enkel filserver som bruker f.eks. FTP-protokollen. Men, vi valgte å bruke MinIO istedetfor. MinIO [29] er en objektbasert fillagringsserver basert på Amazon sin S3-protokoll. Fordelen med MinIO over en enkel filserver er at man får en skalerbarhet som gjør det mulig å lettere kunne håndtere økende trafikkmengde i fremtiden.

4.2 Valg av frontend

Vi har valgt å bruke React med Typescript, Axios og Tailwind CSS. Dette er for å lage en bra arbeidsflyt for å lettere designe en dynamisk og effektiv frontend som effektivt poster og husker data fra databasen.

- React er et JavaScript bibliotek som lar utviklere lettere lage en feit klient som er rask og dynamisk. React er også veldig populært blant andre utviklere, og har et aktivt og bra dokumentert utviklings miljø. Et annet javascript bibliotek vi hadde vurdert er Angular. En av Angular sine styrker er at den er bedre på feilhandtering, siden det er et statisk programmeringsspråk. Feil med typer og diverse skjer under kompilering. React er mye mer dynamisk, og har heller flere feilområder når programmet kjører. Problemet med Angular er hvor lang tid det tar å bygge nettsiden.
- Axios er veldig effektiv når det kommer til nettverks forespørsler. Hvor vanlig fetch funksjonen vil først lage en forespørsel og så vente på svar fra API før den fortsetter, er Axios løfte basert, det vil si at den vil lage en forespørsel og la resten av nettsiden fortsette mens forespørselen blir prosessert, når nettsiden kommer til et sted der den trenger data fra forespørselen vil den vente med å laste inn det.
- Tailwind CSS er istedet for tradisjonell CSS kode og består av et stort CSS rammeverk med attributter som du kan gi til HTML objektene i prosjektet. Det er fint fordi det kvitter seg med mye av

de utdaterte systemene fra standard CSS og tillater å ha alle attributtene lett synlig i samme stedet som HTML og TypeScript koden, uten å trenge flere filer. Selv om Tailwind er et stort rammeverk, har den evnen til å slanke ned filstørrelse sin for å optimalisere tiden, dette gir raskere frontend og krever mindre lagringplass. Tailwind oppnår dette med å lese gjennom filene i frontend når du bygger prosjektet, for å lete etter alle Tailwind funksjoner som er brukt og kvitter seg så med alle andre funksjoner som ikke er brukt.

4.3 Valg av backend

Backend i vår applikasjon er en API som frontend benytter til å utføre handlingene sine. Kravene til backend vil være at den skal være effektiv og rask, kjøre som en HTTP-server og kunne kommunisere med frontend og databasen vi har valgt. Det bør også være gode muligheter for å kunne skrive og gjennomføre tester av de ulike funksjonene. I tillegg må den kunne kjøre under Linux og Docker (mer om Docker senere).

Til backend finnes det flere alternativ som vi har vurdert å bruke. Java og .NET (C#) er vanlig å bruke, det er også JavaScript via Node. Et siste alternativ er Go, dette er et språk flere av gruppemedlemene har tidligere erfaring med fra emnet PROG2005 ⁵ samt fra prosjektet i PROG2052 ⁶ til utvikling av APIer. Av hensyn til at det ikke var ønskelig å bruke tid på å eventuelt lære seg et annet programmeringsspråk valgte vi Go som språket for backend.

Men, det er også gode grunner til å velge Go utenom tidligere erfaring, Go sitt standardbibliotek har mye som vi trenger til backend API-en allerede som å kjøre en HTTP-server inkludert støtte for HTTP over TLS (HTTPS) for sikker kommunikasjon med frontend. Det var også enkelt å finne biblioteker dersom det trengtes, for eksempel for å kommunisere med databaseløsningen vår. Go er også et programmeringsspråk som er raskt å skrive og enkelt å lese noe som gjør det effektivt å skrive kode og inspisere den.

4.3.1 Tredjeparts-bibliotek og tjenester i backend

Backend bruker i hovedsak Go sitt standardbibliotek. Dette er fordi mye av det backend gjør som å fungere som en HTTP server og konvertering mellom Go datastrukturer og JSON for kommunikasjon med frontend er innebygd i standardbibliotek. Men, vi trenger likevel noen tredjeparts biblioteker. Kanskje den viktigste er *Go-MySQL-Driver* som hjelper den innebygde *database/sql*-pakken å kunne kommunisere med MySQL databasen vår. *envconfig*- og *yaml.v3*-pakkene brukes for å lese konfigurasjon henholdsvis fra miljøvariabler og yaml-konfigurasjonsfilen. Google sin *uuid*-pakke brukes for å generere unike ID-er til f.eks. sesjoner og bilder. Disse er såpass unike at vi ikke trenger å bekymre oss for å få konflikter med samme ID-er.

Backend kommuniserer også med to andre tjenester. Den ene er *MinIO*-instansen vår, her bruker vi den offisielle Go-pakken, *minio-go*. Den andre tjenesten er *ClickSend* som brukes til å sende SMS-meldinger. Denne kommuniseres direkte med via REST API-en deres, dermed trengs det ikke et eget bibliotek, man trenger kun bruke *http*-pakken i standardbiblioteket for å sende forespørselene.

⁵PROG2005 - Cloud Technologies: <https://www.ntnu.no/studier/emner/PROG2005>

⁶PROG2052 - Integrasjonsprosjekt: <https://www.ntnu.no/studier/emner/PROG2052>

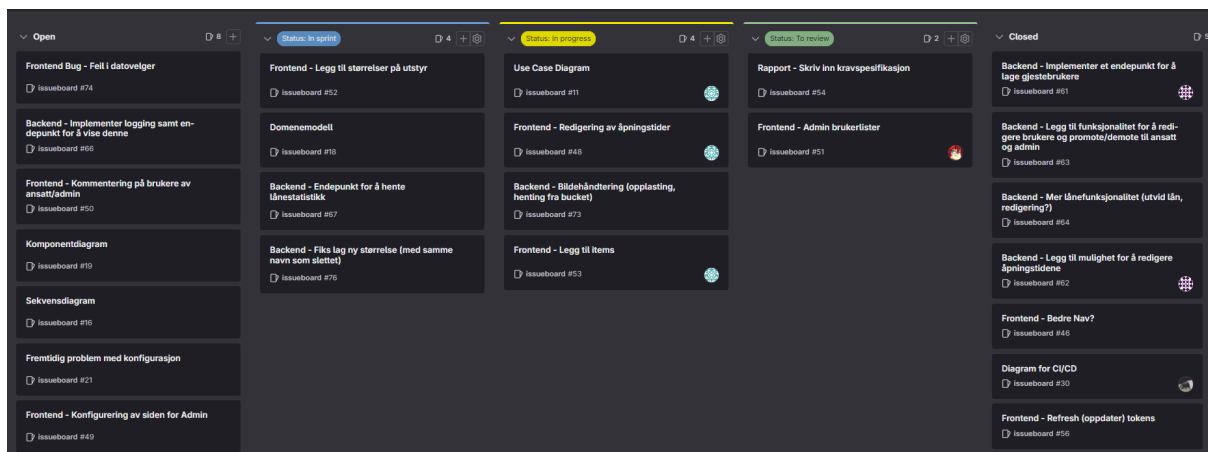
4.4 Andre teknologier og verktøy

4.4.1 GitLab

Under valget av hvilken tjeneste vi skulle bruke for å lagre kildekoden måtte den ha støtte for *versjonskontroll* via Git. I hovedsak så vi på tre alternativer, GitHub og de to instansene av GitLab som tilbys av NTNU. Siden dette er et skoleprosjekt var det naturlig å bruke en av GitLab instansene. Den ene kjøres lokalt i Gjøvik men er plaget av middelmådig ytelse og nedetid, i tillegg var det ingen tilgjengelig *container registry* som vi trenger for Docker-kotainerene våre. Dermed falt valget på den andre instansen som kjøres sentralt av IDI i Trondheim. Dette ble bestemt under planleggingsfasen.

Før vi begynte prosjektet ønsket vi å ha alt av oppgaver, dokumentasjon, og kode samlet på samme sted, og siden vi ønsket å bruke GitLab for å oppbevare koden vår, ga det mest mening å bruke det for å dekke resten av behovene for prosjektet. GitLab er en plattform som tilbyr en rekke ulike tjenester rettet mot utvikling. Noen eksempler på hva de tilbyr er lagring av kildekode gjennom versjonskontroll (Git), automatisk bygging og utrulling av kode (CI/CD), og organisering av prosjekter.

For å få oversikt over de ulike oppgavene våre, delte vi de inn i forskjellige poster i GitLab, ofte kalt "issues". GitLab har et innebygd issueboard som effektiviserte organiseringen vår⁷. I issueboardet kan de ulike oppgavene sorteres etter status, altså planlagt (tilsvarende Product Backlog i Scrum-rammeverket), skal jobbes med denne sprinten, jobbes med for øyeblikket, klar for inspeksjon og til slutt ferdig. Dette er nyttig ved Sprint Planning og Sprint Review-møtene.



Figur 4: Et skjermbilde av hvordan issueboardet vårt så ut mot slutten av prosjektet

Noen andre viktige funksjonaliteter som GitLab tilbyr er lagring av dokumentasjon (Wiki), merge requests, og CI/CD. Merge requests spesifikt lar oss som utviklere se over hverandre sin kode. Dette blir forklart dypere i seksjon 7. CI/CD blir utdypet i kapittel 8.

4.4.2 Docker

Docker er en løsning for å kjøre programvare, i hovedsak serverprogramvare, som bruker OS-nivå *virtualisering*. I Docker kjøres programvarene separert i adskilte kontainere, som kan minne om virtuelle maskiner, men separasjonen er ikke like sterk. I utgangspunktet har programvaren som kjøres ikke tilgang til dataene til programvarene i de andre kontainerene samt resten av filsystemet til maskinen som kjører Docker. Denne isolasjonen gir et lag av sikkerhet og sikrer at data lagret av andre programmer ikke påvirker kjøringen av programvaren.

⁷GitLab tilbyr mye funksjonalitet, som vist på siden deres: <https://about.gitlab.com/features>

Docker-kontainere kjører images som inneholder programvaren som kjøres sammen med alt som trengs for at programvaren skal kjøre inkludert operativsystem. Images blir skapt via en byggeprosess der imaget settes sammen og programvaren kompiles. I CI/CD automatiseres denne byggeprosessen slik at vi alltid har oppdaterte images av programvaren vår.

Dette gjør at å få programvare til å kjøre i Docker er enkelt, man slipper å lete etter annen programvare som trengs og isoleringen gjør at man ikke trenger å bekymre seg for konflikter med programvare som allerede er installert på maskinen. Dette er hovedgrunnen til at vi valgte Docker.

4.4.3 Docker Compose

Docker Compose⁸ er et verktøy som kan brukes til å enkelt kjøre applikasjoner som trenger flere Docker-kontainere samt å konfigurere tilganger til mapper på maskinens filsystem og eksponering av nettverks-porter. Både frontend og backend tjener på å bruke Docker Compose, siden det gjør det enkelt å sette opp infrastrukturen og applikasjonene som trengs for å koble de sammen.

4.4.4 SkyHigh

For å kjøre Docker-kontainerene våre, brukte vi tjenesten SkyHigh. SkyHigh er en av NTNU sine OpenStack-installasjoner som er drevet av Institutt for informasjonssikkerhet og kommunikasjonsteknologi (IIK) lokalt her i Gjøvik. OpenStack er en virtualiseringsplattform der man kan sette opp virtuelle servere og nettverk. Dette gjorde at vi kunne få applikasjonen kjørende i et miljø som vil ligne på det som vil brukes når applikasjonen tas i bruk. SkyHigh, samt de andre Openstack-instansene til NTNU, er riktignok beregnet kun for bruk til utdanning og forskning ved universitetet. Den kan altså ikke brukes når applikasjonen skal tas i bruk, da må man se seg etter en tilsvarende tilbyder, noen av de mest populære er Google Cloud, Microsoft Azure og Amazon Web Services (AWS). Å finne ut av hvilken av disse som er aktuelt å bruke er ikke del av oppgaven, se rammene satt i kapittel 1.

4.4.5 Discord

Kommunikasjonsverktøyet mellom gruppemedlemmene endte opp med å bli Discord da det er en tjeneste de fleste gruppemedlemmene allerede er vant med å bruke. Den har også all funksjonalitet som vi trengte: kommunikasjon både med tekst, lyd, og muligheten til å lage flere kanalertil ulike temaer. Det ble brukt til blant annet avtale av møter og spørsmål til de andre gruppemedlemmene. Discord brukes også til å avholde digitale møter samt kombinerte møter dersom en eller flere av gruppemedlemmene ikke hadde mulighet til å møte opp på et fysisk møte.

4.4.6 Google Drive

Til lagring av dokumenter, diagrammer og andre filer som ikke er kildekode brukes Google Drive. For dokumenter gir det mulighet for redigering og samarbeid i sanntid. Tidsbruket på prosjektet ble også logget i en egenlaget regneark lagret i Google Drive. Hver gruppemedlem hadde hvert sitt ark der man fylte inn øktene med dato, tidspunkt fra og til, kategori samt en valgfri kommentar. Deretter vil regnearket regne total tidsbruk fordelt på uker, kategori og gruppemedlem. Dermed får vi en oversikt over hva vi bruker tid på og hvor mye tid vi bruker.

⁸<https://docs.docker.com/compose/>

4.4.7 IDE

Til utviklingen av frontend ble det brukt Microsoft Virtual Studio Code (VS Code) og JetBrains sin WebStorm IDE. WebStorm er en kraftig IDE tilpasset for webutvikling. Til utviklingen av backend ble det brukt JetBrains sin GoLand IDE. Dette er en variant av JetBrains sin populære IntelliJ IDE tilpasset spesifikt for utvikling i programmeringsspråket Go.

Bakgrunnen av valget for å bruke JetBrains sine IDEr er at studenter fra NTNU kan få en studentlisens for disse. De tilbyr kraftige verktøy for å skrive, refaktorere og debugge kode. VS Code er også et godt og lettvent alternativ som er tilgjengelig helt gratis. I forhold til JetBrains sine IDEr, er ikke VS Code like krevende å kjøre.

4.4.8 phpMyAdmin

Verktøyet phpMyAdmin brukes til å administrere databaser med et grafisk webbasert grensesnitt. Det støtter i hovedsak MySQL og MariaDB, og siden vi brukte MySQL for prosjektet vårt var det et enkelt valg å bruke dette i utviklingen vår. Verktøyet gjør det enklere å få oversikt over databasens struktur og dataene i tabellene som er nyttig under utvikling av databasestrukturen og under testing av applikasjonen for å se hva som faktisk ligger i databasen for øyeblikket. Det kan også brukes til å designe databasens struktur, og i løpet av prosjektet brukte vi verktøyet til å iterere over databasens design.

4.4.9 Figma

Figma er verktøyet vi brukte til å lage wireframes. Figma er et verktøy for å designe brukergrensesnitt og man kan enkelt tegne opp de ulike elementene som et brukergrensesnitt består av. I tillegg er det mulig å koble sammen ulike sider slik at wireframen blir interaktiv. Dette gir en klarere ide over hvordan brukergrensesnittet kan fungere når man presenterer wireframen for andre. Figma tillater også sanntidsredigering slik at flere gruppelemmer kan samarbeide på designet.

4.4.10 Draw.io

Til tegning av diverse diagrammer, figurer og flytskjemaer brukte vi det gratis verktøyet *Draw.io*. Verktøyet har integrasjon med Google Drive som lar oss lagre og dele figurene med hverandre.

5 Design

Dette kapitlet omhandler emner rundt teknisk design av applikasjon, design av databasestruktur samt design av brukergrensesnittet. Det tas utgangspunkt i de funksjonelle kravene nevnt i kapittel 2 og teknologivalgene nevnt i kapittel 4.

5.1 Programvarens komponenter

Programvareløsningen består av to komponenter, frontend og backend. I tillegg til en egenutviklet databasestruktur tilpasset til databasesystemet MySQL. Med dette har applikasjonen en tre-lags arkitektur [27], der man deler inn applikasjonen i tre lag dvs. et presentasjonslag som vil være grensesnittet som brukeren samhandler med, et applikasjonslag der applikasjonens logikk utføres samt et datalag der applikasjonens data lagres og hentes.

Frontend representerer presentasjonslaget i applikasjonen vår, det er dette laget som brukeren samhandler med. I vår applikasjon som er en webapplikasjon kan frontend beskrives som selve nettsiden. Frontend sine oppgaver består av å være et grensesnitt som lar brukeren snakke med applikasjonslaget og å presentere dataene hentet fra datalaget via applikasjonslaget. Frontend sitt grensesnitt er også i utgangspunktet designet slik at den tilpasser seg automatisk til ulike skjermstørrelser som smarttelefoner, nettbrett og datamaskiner.

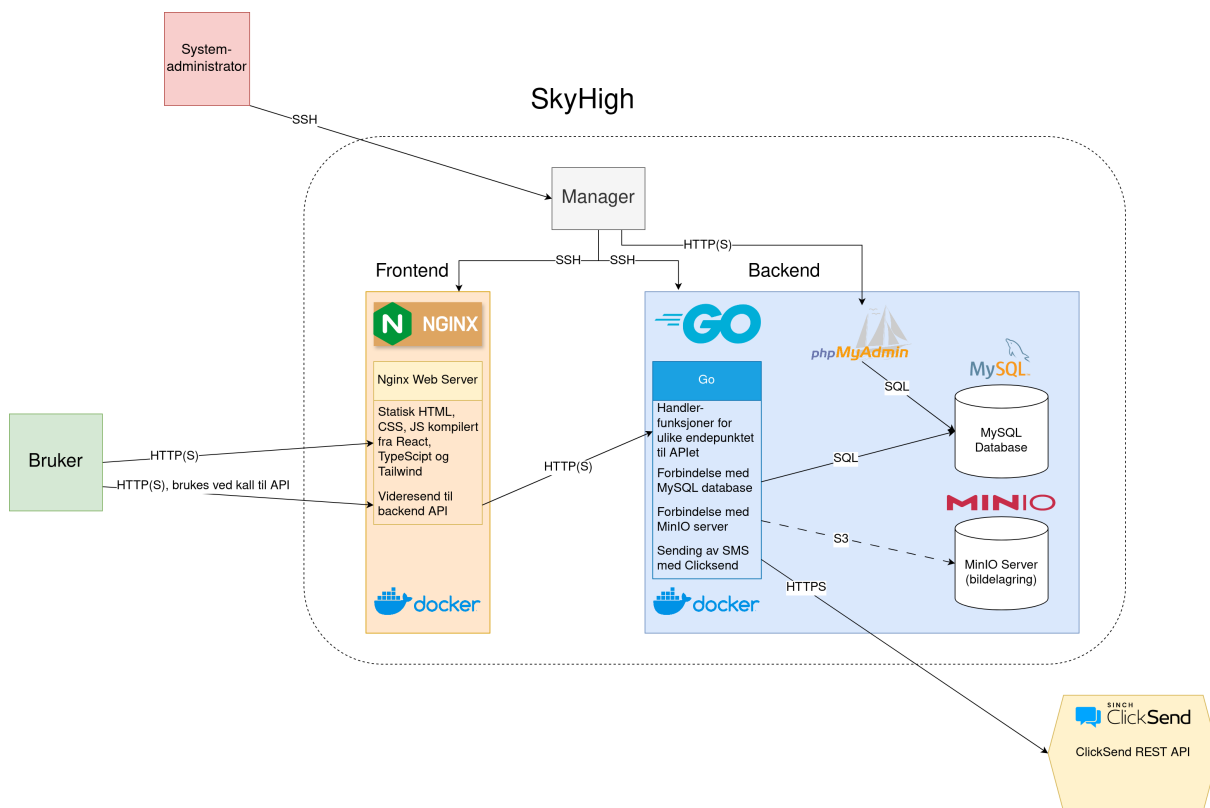
Backend representerer applikasjonslaget i applikasjonen vår, det vil si at backend er mellomlaget som har logikken til å utføre handlingene som brukeren ønsker å gjennomføre. Måten frontend kommuniserer med backend på er via HTTP-forespørsler, ved hjelp av URL, de ulike HTTP-metodene (`GET`, `POST`, `PATCH` og `DELETE`), ulike typer data samt medsendt JSON fortelles det nøyaktig hva som skal gjøres. Dette gjøres i hovedsak ved å hente data fra databasen og/eller oppdatere dataen i databasen som da er datalaget i applikasjonen. Backend er med andre ord frontend sin kommunikasjonskanal med databasen. Det vil da være opp til backend å gjennomføre tillatelsessjekk for å gjennomføre visse handlinger samt å validere inndata som stammer fra brukeren på frontend.

Frontend bør likevel også validere data som skal sendes til backend slik at man unngår unødvendige forespørsler som tar tid og fører til unødvendig belastning på backend. F.eks. dersom under registrering så glemte brukeren å taste inn fornavnet sitt, så bør frontend kunne si ifra istedetfor å sende en forespørsel til backend for så å få feil på grunn av manglende fornavn, det vil ta lengre tid som man kunne spart ved å gjøre sjekken i frontend før man sender forespørselen om å registrere brukeren til backend.

Kommunikasjonen mellom de ulike komponentene bruker ulike protokoller. Backend kommuniserer med frontend over HTTP-protokollen via formatet JSON. JSON [31] er et standardisert format for å utveksle informasjon mellom ulike systemer, da typisk data som kan uttrykkes via tekst eller tall, som er både relativt lett å lese samtidig som det finnes muligheter for å konvertere mellom innebygde datastrukturer og JSON i de fleste programmeringsspråk. Dette gjelder spesielt både i Go som brukes i backend og i JavaScript som brukes i frontend som begge har støtte for dette i standardbibliotekene sine. Dette gjør JSON til et format som er velegnet til denne oppgaven.

Kommunikasjonen mellom backend og databasen skjer via SQL-spøringer. En slik spørring kan blant annet hente data fra databasen eller utføre handlinger som modifierer databasen. Videre for spørringer som henter data kan man bruke visse definerte nøkkelord for å blant annet definere hvilken tabell man skal hente data fra, hvilke data som er av interesse og hvilke kriterier som brukes for å filtrere radene i tabellen. Mer avansert funksjonalitet inkluderer å kunne kombinere tabeller og telle opp antall rader etter at man har filtrert på visse kriterier. Spørringene gir derfor backend stor fleksibilitet til å utføre handlingene som kreves.

5.2 Systemarkitektur



Figur 5: Arkitekturskisse (overblikk over servere)

Arkitekturen vår for å teste applikasjonen vår i drift under utviklingen består av tre virtuelle maskiner i SkyHigh, se skissen i figur 5. Manager-maskinen brukes av administrator for å styre driften av tjenesten. De to andre maskinene er dedikert til henholdsvis frontend- og backend-komponenten av applikasjonen. På frontend-maskinen kjøres Nginx Web Server som tilbyr frontend i form av statiske HTML, CSS og JavaScript-filer generert via React og TypeScript. I tillegg fungerer den som en proxy mot backend API-et slik at brukerens nettleser kan kjøre forespørsler mot API-et.

På backend-maskinen kjøres blant annet backend-programvaren. Den samhandler med flere andre tjenester inkludert MySQL databasen som kjøres på samme maskin og i teorien også MinIO-serveren (ikke satt opp i SkyHigh-miljøet vårt for øyeblikket). phpMyAdmin kjører også på backend-maskinen for å administrere databasen. Backend-programvaren kommuniserer også med den eksterne tredjepartstjenesten ClickSend for sending av SMSer.

Alle serverkomponentene som kjøres som del av applikasjonen vår, altså Nginx, Go-backend, phpMyAdmin, MySQL og MinIO, kjøres som Docker-kontainere.

I fremtiden kan applikasjonen bruke en lignende arkitektur hos en av skytilbyderne, men man bør vurdere å sette opp en dedikert virtuell maskin til databasen fremfor å kjøre det på samme maskin, da får man dedikerte ressurser til en av applikasjonens viktigste komponenter.

5.3 Design av frontend

Designet av frontend er basert på wireframet og tilbakemelding fra oppdragsgiver. Naturligvis har sluttproduktet endret seg betydelig fra wireframet. Mange av endringene er basert på tilbakemelding fra

oppdragsgiver. En av de første endringene vi gjorde var å endre fra en **hamburger-meny** over til en **navigasjonsbar** som ligger på toppen. Dette var for å minimere klikk og kommunisere til brukere alt man kan gjøre på denne siden, ved kun ett blikk. At den ligger på toppen av siden gjør det enkelt å presentere alt brukeren har tilgang til samtidig som at den dynamisk oppdaterer seg basert på statusen til bruker. Når en bruker først åpner siden vil reservasjon og lånoversikten være i en grå farge, siden en gjest ikke vil kunne ha lån eller reservasjoner. Når bruker har logget inn, vil reservasjonsliste bli tilgjengelig, mens brukernavnet under Login-knappen oppdateres til å matche brukerens navn så det aldri er tvil om hvem som er logget inn. Siden navigasjonsbaren alltid er tilgjengelig på toppen av alle sider, vil bruker ha tilgang til å gå ut av en hvilken som helst handling eller side når en de vil, slik at det er mye vanskeligere å gå seg vill på siden.

Feedback blir gitt tilbake til bruker gjennom en såkalt *toast* fra et eksternt **bibliotek**, en toast vil vises over hele siden som en liten boble med tekst i, som forsvinner etter bestemt tid, vanligvis noen sekunder. Meldingene som blir sendt med i toasten er typisk tilbakemelding på at felt har blitt fylt inn feil, backend ikke svarer eller at du må være logget inn for å fortsette.

En annen aspekt av designet er at vi lar vi bruker logge inn fra en hvilken som helst side med hjelp av en pop-opp dialog, bruker kan også åpne feltet manuelt med å klikke på login knappen når de ikke er logget inn. Vi valgte å bruke en pop-opp dialog for login fordi det er flere sider som endrer seg basert på om bruker er logget inn eller ikke, så hvis en bruker vil ønsker å nå en tjeneste som krever å være logget inn så kan de gjøre det uten å måtte gå tilbake igjen eller fylle ut felter en gang til. I tillegg lar en pop opp kommunisere at det ikke krever å logge inn for å bare se på siden og utstyr, siden bruker vil alltid ha mulighet til å klikke ut av dialogen.

5.3.1 React

React er et komponentbasert rammeverk, og siden består da naturligvis av mange forskjellige komponenter. Det er derfor viktig å ta hensyn med hvordan komponentene jobber sammen. Navigasjonsbaren for eksempel er på toppen av nettsiden, og den skal være synlig gjennom hele applikasjonen. Derfor plasseres denne øverst i hierarkiet av komponenter - direkte inn i App.tsx. Noen av gevinstene her er at du får dele attributter (state), som blir sett på dypere i kapittel 6.1.4.

5.3.2 Inventarside

Vår første forsøk på en inventar side, hadde originalt planer om å kunne dele opp utstyr i både kategorier og i sesonger, hvor hver sesong ville ha hver sin farge. Dette førte til logistiske problemer, som hva gjør du med utstyr som kan brukes under flere sesonger og utstyr som er helårs. Et annet problem var å koordinere farge bruk over siden og til bruker.

Inventarsiden ble spesielt forbedret av feedback fra wireframe. Det hjalp oppdragsgiverens komme med forskjellige krav som forbedret siden. Et av disse kravene var å få flere gjenstander inn på inventarsiden samtidig. Dette oppnådde vi med å minimere filtreringsystemet på toppen av wireframet, både med å kutte sesongfiltrering og med å sette tag-filter og søkebar på samme linje, som ga oss mindre formfaktor.

Tag-systemet er en forbedret versjon av sesongfiltrering systemet. Som du kan se i modellen 6, har hver gjenstand et navn i fet tekst og under der har de tags. Tags hjelper bruker med å forstå kontekst til en gjenstand. I tillegg vil det kunne hjelpe en bruker med å finne relaterte gjenstander. For eksempel kommer Slalåmski og Skistaver til å begge ende opp med Ski som tag. Når en bruker søker på tags, vil frontend automatisk gi forslag på tags som matcher hva bruker søker. Etter en bruke velger en tag, vil de få lov til å søke på flere tags eller kvite seg med en tag de har søkt på. Resultatet vist til bruker vil nå kun være gjenstander som har minst alle tags som er søkt på tags.

Søkefeltet derimot er ment for en bruker som vet nøyaktig hva de vil ha. Søkefeltet vil påvirke gjensander

vist til bruker, til kun de som matcher navn som søkefeltet, bruker kan bruke både tags og søkefelt samtidig.



Figur 6: Wireframen til venstre er første utkastet for inventarsiden, og høyre viser hvordan det ser ut i dag

5.3.3 Reservasjonslogg

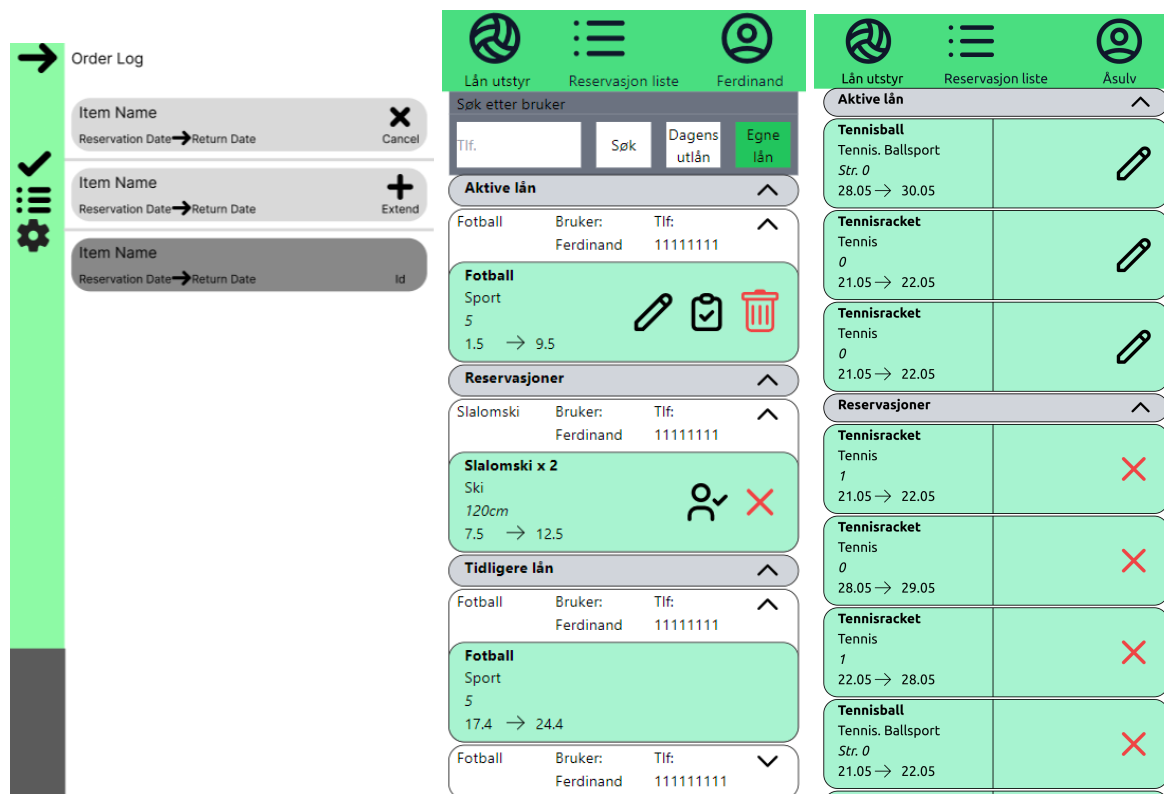
Målet med reservasjonsloggen var å gjøre hvert lån så simpelt som mulig. Slik at selv når siden fyller opp, skal det fortsatt være tydelig hva en kan gjøre med hvert lån. Dette fikk oss til å designe siden rundt hva et lån faktisk er, mens vi sakte introduserer nye funksjoner etter som de ble nødvendige. Vi startet med å se på, hvilke stadier kan et lån være i. For å så designe 3 forskjellige stadier av lån bokser, istedet for en overkompleks en.

Et lån kan være i en av tre stadier som vi baserte designet på:

1. Reservasjon: Når et lån først blir lagd av en bruker så er det en reservasjon, dette vil markere utstyret som tatt ut av lager. På dette tidspunktet er det kun en handling bruker kan gjøre og det er å kansellere lånet (rød X), som vil sette det tilbake på lager igjen. En ansatt har også muligheten til å kansellere et hvert lån (rød X), men har også muligheten til å overlevere reservasjon og gjøre det om til et aktivt lån (bruker ikon med hake tegn til høyre).
2. Aktivt lån: Et aktivt lån symboliserer at gjenstanden er overlevert til bruker og det er de som nå har ansvar for den. På dette tidspunktet vil bruker ikke lenger kunne kansellere lånet selv, men har

nå muligheten til å forlenge lånet en gang for en uke (pen ikon). Dette er for å kommunisere til bruker at. Selv om du ikke skal ha lov til å låne i mere en 2 uker (kan endres av administrator), så tilbyr utstyrsbanken fleksibilitet. Ansatte har to muligheter med et aktivt lån og det er enten ta det tilbake og sette det inn i lager (liste ikon med hake), eller markere lånet som ødelagt (rød søppelbøtte). Begge disse mulighetene vil gjøre lånet om til et tidligere lån.

3. Tidligere lån: Når et lån er markert som tidligere så er det ingen flere handlinger man kan gjøre med det, på dette tidspunktet skal lånet fungere som en historikk og vil ikke bli slettet, selv om en sletter brukeren.



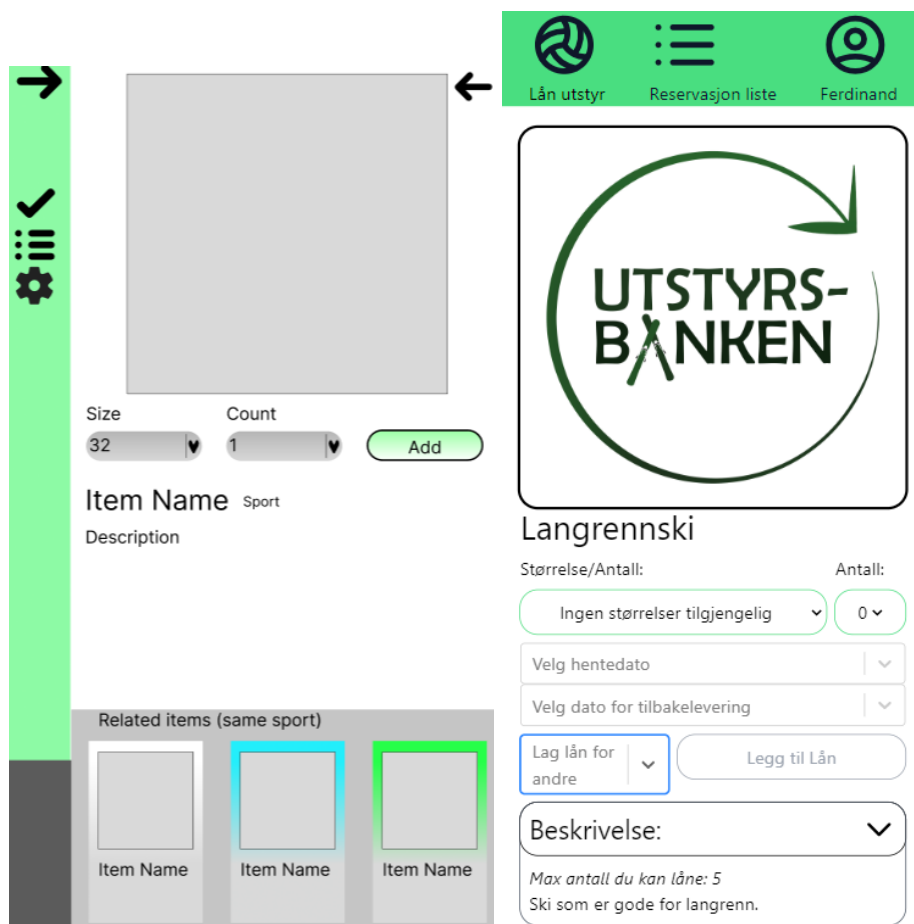
Figur 7: Wireframe til venstre, hvordan admin ser siden i midten og hvordan bruker ser siden til høyre

Disse tre stadiene var det først vi ble enig om når det kom til hva et lån er, og er hva vi ente opp med å basere siden på, dette gjorde det veldig viktig å kunne kommunisere til bruker at disse lånene er forskjellige fra hverandre basert på stadier, og at det er 3 forskjellige grupper, med varierende viktighetsgrad. Derfor har siden muligheten til å kollapse de individuelle seksjonene. I tillegg vil tidligere lån være den eneste av gruppene som er kollapse når bruker først åpner siden, for å kommunisere at den ikke kommer til å være like viktig for bruker.

Ansatte og administrator skal også kunne ha samme tilgang og funksjonalitet, som en vanlig brukere uten å måtte lage en separat bruker. Derfor er ansatt siden basert på bruker siden, med ekstra funksjonalitet som kreves for å kunne godkjenne et lån for en annen person. Som du kan se i 7 så kan du se at ansatt versjonen av siden har mye mere informasjon en bruker siden. Dette er fordi på brukersiden kan bruker kun se sine lån, mens en ansatt side trenger en måte å vise hvem som har lagd hvilke lån, i tillegg så kommer det til å være mange flere lån på ansatt siden en det som er på bruker siden. Så vi møtte fort på et problem hvor det var vanskelig for ansatte å finne en bruker, derfor vil alle reserverasjoner for ansatte være kollapset inn i en liten hvit rektangel som holder data om, navn og tlf på hvem som lagde lånet, og hva reserverasjonen var for. I tillegg har ansatt 3 forskjellige måter å filtrere lån på. Dagens lån, vil gi alle lån som enten skal hentes i dag, eller som skal avleveres i dag, i tillegg til det er det Personlige lån og Søk

etter bruker, disse vil gi alle lån om en bruker, enten brukeren du søker på eller deg selv. Dette er hvis en bruker som har lyst til å levere tilbake reserveringer tidligere / ta ut en reservering tidligere, eller for den ansatte. For å kunne ha en oversikt over sine egne lån.

5.3.4 Utstyrside



Figur 8: Sammenlikning mellom wireframen og endelig utstyrsside

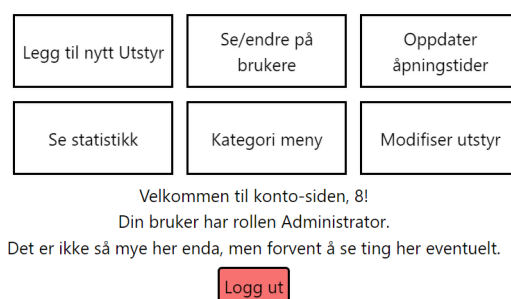
Utstyrside er der en bruker skal kunne lage en reservering, og trenger derfor mange felter og en måte å lede bruker gjennom. I wireframen ønsket vi først å oppnå dette med å la bruker først fylle ut feltene størrelse og antall, før det åpner seg en pop opp, hvor bruker så blir spurt om de vil fortsette å legge til flere utstyr til en kurv, eller gå til kassa. Dette systemet skulle gjøre Gjenstand siden simplere, siden å velge dato skulle bli gjort med en hel handlekurv av utstyr. Samtidig som at det skulle kunne gi bruker en bedre oversikt over hva de er i ferd med å reservere. Dette endte opp med å fort bli alt for komplisert, så vi bestemte oss for kutte ut handlekurv systemet. Du kan lese mere om det under diskusjon. Vi endte derfor opp med å måtte sette inn en datovelger inn i siden.

Siden siden allerede begynte å bli full av felter som bruker måtte fylle ut, var det viktig å få dem så konsise som mulig. Vi oppnådde dette med å unngå en kalender basert løsning for å velge dato som vi først hadde planer om, for å heller ha en dynamisk dropdown meny med en dato liste. Denne listen vil oppdatere seg, slik at den kun kan gi relevante dato. Dette står det mere om under implementasjon om data håndtering. Realistisk kommer det ikke til å være mere en 7 tilgjengelige dager, som lar siden forbli konsis for brukere.

Administratorer har en til mulighet som kan sees på modellen 8. Uthevet i blå er knappen for å lage reserverasjoner for andre. Dette tilater utstyrskontoret å fortsatt la kunder lage reserverasjoner over telefon. I tillegg har ansatte muligheten til å lage og gi andre brukere direkte aktive lån, med å velge dagens dato. I tillegg hvis en kunde kommer innom og ikke har lyst til å lage en bruker, kan ansatte gi dem en gjeste bruker som kan holde et lån, gjennom en liten pop opp som ber om gjestbruker sitt navn og telefon nummer.

5.3.5 Adminside

En del av spesifikasjonen i kravspesifikasjonen (kapittel 2) er eksklusiv til administratoren på siden. For å håndtere dette var det originalt planlagt en dedikert side som eksponerer funksjonaliteten. Mot slutten av utviklingen innså vi at for å holde navigasjonsbaren simpel var den beste løsningen å sette disse knappene på brukersiden. Det fyllte tomrommet på denne siden, som i framtiden kan bli brukt for redigering av egen brukerdata.



Figur 9: Admin knappene på profilsiden. Dukker bare opp om du er administrator

All funksjonaliteten for administrator ble implementert, som inkluderer:

1. Legg til nytt Utstyr
2. Se/endre på brukere
3. Oppdater åpningstider
4. Se statistikk
5. Kategori meny
6. Modifiser utstyr

5.4 Bruker- og innloggingssystem

En sentral del av applikasjonen vår er å ha støtte for brukerkontoer slik at lån og reserverasjoner kan knyttes til en spesifikk person sin brukerkonto. Det er også ønskelig med en form for autentisering for å kunne utføre de fleste handlingene som applikasjonen skal støtte. I tillegg skal man unngå at enkeltpersoner kan opprette flere kontoer samt at kontaktinformasjon skal være verifiserbare og reelle. Grunnen til dette er at dersom det oppstår utfordringer med lån så skal Ustyrskontoret kunne kontakte låneren.

Vi så på tre ulike alternativer for å håndtere innlogging av brukere. En av disse var å benytte såkalt federering, det vil si å bruke noen andre sitt innloggingssystem. Dette er mulig fra flere tilbydere som

Google, Apple og Facebook. Dette gir oss muligheten både til å slippe å lagre passord og gjøre det svært enkelt for brukere å registrere seg, gitt at de har en konto fra en av tilbyderene allerede. Federering ble likevel ikke aktuelt på bakgrunn av at vi ikke fikk informasjonen vi ønsket, det inkluderer telefonnummer slik at Utstyrbanken kan ta kontakt med brukerne om lånene sine. Det vil også fungere dårlig som eneste løsning da man er avhengig av at bruker har konto hos en av tilbyderene som man har valgt å støtte.

En annen løsning vi så på var å spesifikt bruke Vipps som innloggingsmetode, dette i bakgrunn av at man ikke enkelt kan lage flere Vipps-kontoer. Å lage en Vipps-konto krever også at informasjonen som brukes er reell. I tillegg er det noe mange bruker og kjenner til. Likevel har Vipps noen av de samme problemene som federeringsløsningen. For det første så må brukeren ha en Vipps-bruker, det er ikke alle som har og det er dessuten aldersgrense på å ta i bruk Vipps. Etter diskusjon med oppdragsgiver fant vi også ut at Vipps kan gi brukerne et inntrykk at det er noe som skal betales for, mens det å låne fra Utstyrbanken er gratis. I tillegg koster det penger for Utstyrbanken å bruke Vipps som innloggingssystem.

Det siste alternativet er å utvikle en egen løsning, det gir mye større frihet i hvordan den skal fungere og brukerne er ikke avhengig av å ha en bruker på en eksisterende tjeneste. Som regel må man likevel ha noe å knytte kontoen til, vanligst er dette en e-postadresse, men vi valgte å knytte kontoen opp mot et telefonnummer. Det gir Utstyrbanken kontaktinformasjon som kan brukes til å ta kontakt dersom for eksempel noe utstyr ikke blir levert tilbake i tide.

For å sikre at man ikke kan bruke hvilket som helst telefonnummer ved registrering må man også ha en måte å verifisere at det er riktig. Dette kan enkelt gjøres ved å sende en SMS til telefonnummeret med en kode som brukeren så taster inn for å bekrefte at brukeren eier telefonnummeret. Sending av SMSer må gjøres gjennom en tilbyder av slike tjenester, vi fant to tilbydere som kunne brukes til dette. Den ene kostet mye både ved oppstart og hadde en månedlig sum tillegg til selve SMSene, derfor falt valget på ClickSend som kun tar betalt per SMS som sendes. På grunn av kostnader knyttet til SMSer ble det viktig å begrense mulighetene for brukere å få applikasjonen vår til å sende SMSer, mer om dette i kapittel 6.2.10.

I teorien kunne vi hatt vår løsning i tillegg til federering eller Vipps, men det var ikke ønskelig å bruke tid på å implementere både federering/Vipps og vår egen løsning da det ville tatt utviklingstid fra annen viktig funksjonalitet til applikasjonen.

5.4.1 Sletting av brukere

Sletting av brukere er noe som ble diskutert underveis i prosjektet da det har noen implikasjoner. Opprinnelig var planen at dersom en person ønsker at brukerkontoen sin skal bli slettet så må man ta kontakt med Utstyrbanken og selve slettingen ville da blitt en ganske manuell prosess for administrator av tjenesten. Det ville ha innebært at administrator samhandler direkte med databasen enten via kommandolinjen eller et verktøy som PHPMyAdmin. Årsaken til dette er at brukeren er i databasen knyttet til en del annen data det er ønskelig å ta vare på som gamle lån (og i utgangspunktet hendelsesloggen om den hadde blitt implementert). Dataene om gamle lån brukes i hovedsak for statistikk.

Etterhvert fant vi en løsning som vil være at administrator fortsatt har ansvaret for slettingen men at prosessen er mer automatisert. Persondataen (navn og telefonnummer) blir nå anonymisert og brukeren forhindres fra å logge på. Det er enda en komplikasjon ved sletting av brukere og det er eventuelt om brukeren fremdeles har aktive lån. For mens reservasjoner enkelt kan kanselleres må Utstyrbanken få tilbake utstyr som er lånt ut og da må Utstyrbanken kunne vite hvem som har lånt ut utstyret i tilfelle det ikke blir levert tilbake i tide. Da må dette ordnes opp i før brukeren kan slettes, systemet vil bli implementert slik at det blir umulig å slette brukere med aktive lån.

Dette skal gjøre det mulig at applikasjonen er innenfor regelverket knyttet til retten til sletting [5]. Anonymiseringen skal være nok for at slettingen skal være reell, det bør likevel informeres i en personvernerklæring at data om lån bevares for statistikken sin del. Dersom en bruker ønsker å slette kontoen sin

skal det kunne gjøres uten videre men, med unntak dersom brukeren har aktive lån hos Utstyrskontrollen. Da skal Utstyrskontrollen ha muligheten til å beholde kontaktinformasjonen inntil lånene har konkludert, deretter kan brukerkontoen slettes.

5.4.2 Sesjoner

Når en bruker er logget inn har brukeren en aktiv sesjon i applikasjonen. Så lenge sesjonen er aktiv regnes brukeren som logget på og kan utføre alle handlingene som brukeren har tillatelse til (mer om det senere). Sesjonen er aktiv i en viss lengde eller til brukeren velger selv å logge ut. En referanse til sesjonen må eksistere hos frontend slik at brukeren har en aktiv sesjon kan kommuniseres til backend. Denne referansen må lagres på en slik måte at det blir vanskelig og aller helst umulig at uvedkommende får tilgang på den. Dette er fordi at en aktiv sesjon i praksis kan regnes som et midlertidig passord for brukeren. Mer om faktisk implementasjon i applikasjonen av dette i kapittel 6.

5.5 Tilgangskontroll

Enkelte operasjoner skal kun enkelte brukere kunne gjennomføre. Noen av disse er knyttet til daglig drift i Utstyrskontrollen der ansatte registrerer utlån og tilbakeleverte lån mens noen er mer administrative som å oppdatere inventaret og sletting av brukere. Det trengs derfor et system som kan regulere tilgangen på visse handlinger. Noen av oss hadde erfaring med å implementere tilgangskontroll fra prosjektet i emnet Integrasjonsprosjekt ⁹, basert på den erfaringen ønsket vi et mer robust og fleksibelt system.

Løsningen ble et rollebasert system (ofte kalt role-based access control eller RBAC på engelsk) med CRUD-baserte tilganger på ressurser (representerer deler av systemet eller ulike typer objekter som lån og brukere). Dermed kan man sette opp de ulike rollene som ulike brukere har, altså gjest (ikke pålogget), vanlig bruker/låner, ansatt og administrator, med de tilgangene som passer for hver rolle. Eksempelvis har ansatte lese-tilgang på ressursen *User* som gir de muligheten til å se brukerdata om alle brukerne i systemet (selvsagt ekskludert brukerens passord), mens vanlige brukere har kun lese-tilgang på ressursen *UserSelf* som betyr at de kun kan se sin egen brukerdata.

5.6 Åpningstider

For å kunne vise informasjon om når Utstyrskontrollen er åpen på nettsiden samt å unngå at brukere kunne lage reserveringer med hentedato eller leveringsdato der Utstyrskontrollen ikke er åpen, må applikasjonen kunne håndtere informasjon om Utstyrskontrollens åpningstider. Åpningstidene er heller ikke statiske som betyr at man vil også måtte implementere muligheten til å redigere åpningstidene. Et annet poeng er at for enkeltdager vil åpningstidene avvike fra normalen, det kan være at ingen ansatte har mulighet den dagen eller at dagen er en helligdag, dette ble nevnt i kapittel 2.

Dermed må applikasjonen designes rundt dette. Databasestrukturen ble blant annet designet for å ha en tabell som inneholder de ordinære åpningstidene basert på ukedager og en tabell som inneholder dagene som har avvikende åpningstider. Resten av applikasjonen måtte også designes rundt dette, spesielt viktig er at avvikende åpningstider får prioritet over de faste åpningstidene. Dette vil riktignok slik det er nå, begrense enkelte muligheter for faste åpningstider som å ha åpen en dag annenhver uke. For øyeblikket er ikke dette et problem da Utstyrskontrollen har faste åpningstider hver uke.

En annen utfordring med åpningstidene er at dersom de endres kan det skape problemer for aktive lån og reserveringer siden åpningstidene når som helst kan endres. Dette kan føre til at Utstyrskontrollen nå holder stengt når lånet eller reserveringen skal hentes eller leveres. Spesielt problematisk er det for

⁹[PROG2052 - Integrasjonsprosjekt](#)

reservasjonene siden de skal automatisk fjernes dersom utstyret ikke hentes innen hentedatoen. Med lånene kan utstyret i teorien leveres tilbake både før og etter innleveringsdato. En stund ble det vurdert om systemet automatisk skal beregne nye datoer til lånene og reservasjonene det gjelder ved endring av åpningstidene. Det ble dessverre problematisk å kunne informere lånerne om de nye datoene etter at notifikasjoner ikke ble implementert. Derfor ble det valgt at de ansatte ved Utstyrskontoret må manuelt redigere lån og reservasjoner og informere lånerne. Det vil da likevel trenge funksjonalitet for å kunne enkelt få oversikt over de lånene og reservasjonene det gjelder.

5.7 Design av REST API

Designet av APIet vårt, det vil si hvordan man samhandler med HTTP-protokollen følger en del av prinsippene for at APIen skal regnes som REST API. Ifølge artikkelen fra IBM [26] er noen av de sentrale prinsippene til en REST API:

- En ressurs (f.eks. en bruker eller et lån) skal ha en spesifikk URL knyttet til seg.
- Separering av klient og server. En klient kan kun samhandle med en REST API via HTTP-forespørsler på gitte URLer. En server skal heller ikke kunne påvirke en klientapplikasjon utover å sende responser over HTTP på forespørslene til klienten.
- En REST API skal ikke håndtere state. Det vil si at en REST API skal ikke oppbevare data om en klient sine tidligere forespørsler og kunne bruke denne informasjonen for fremtidige forespørsler. Dette kalles at API-et er "stateless". Det vil si at klienten må sende med all nødvendig informasjon for å håndtere hver eneste forespørsel.
- Når det er mulig, bør responser caches enten på klienten eller serveren. Dette øker både ytelse og skalerbarhet.
- Man skal designe systemet slik at man ikke krever at klient og server snakker direkte med hverandre, men at det også kan være mellomledd.

APIet ble strukturert fra starten av til å deles opp i de ulike typene ressurser den skal håndtere altså brukere, inventar, kategorier, lån og reservasjoner. I tillegg kommer autentisering, konfigurering, statistikk og revisjonslogg. Sistnevnte ble dog ikke implementert. Et utkast til designet ble laget og dannet grunnlaget for dagens API (utkastet kan leses i vedlegg I). Endringene fra utkastet var i hovedsak tilpasning til å fungere med endelig databasedesign samt enkelte nye endepunkter ble introdusert underveis da behovet for disse dukket opp under utviklingen av frontend.

Et slikt behov var å kunne hente flere brukere samtidig uten å hente alle for å vise navnet til låner på lånoversiktsidene til ansatte. Dermed slapp man å sende en forespørsel for hver bruker. Et annet var å kunne hente brukerne som delte et telefonnummer (telefonnumrene er unike for registrerte brukere, men det gjelder ikke gjestebrukere) slik at det gikk ann å søke etter brukere basert på telefonnummer.

For å se hvordan APIet endte opp med å bli ved slutt av utvikling, se vedlegg J som dokumenterer de ulike endepunktene til API-et.

Designet avviker noe fra typisk REST API-design da APIet ble blant annet designet med endepunkter der URL inneholder "handlingsord". Et eksempel er endepunktet for å opprette en ny reservasjon `/api/reservations/create`, der `create` er handlingen som blir gjort på ressurstypen `reservations`. Et annet eksempel er endepunktet for å registrere et lån som returnert som er `/api/loans/return?loan={lån-id}`, der `return` er handlingen som blir gjort på ressurstypen `loans`. Hvilket lån som returneres spesifiseres med en `query parameter`, valgst i en REST API er å bruke en `path parameter`. I et typisk REST API ville for eksempel handlingen å lage en ny reservasjon, en POST-forespørsel mot `/api/reservation`. Grunnen til at vi ikke gjorde dette var at "handlingsord" gir muligheten til å enkelt utvide med flere typer handlinger for de ulike

ressurstypene. For standardhandlinger (som redigering og sletting) med eksisterende ressurser brukes som oftest standard REST API-design med [path parameter](#) for å spesifisere ressurs. Et eksempel er dersom man ønsker å redigere et lån med ID 21 vil forespørselen være av typen `PATCH` mot `/api/loans/loan/21`.

Et annet avvik i forhold til REST API-retningslinjene er at API-et skal være "stateless". Med *rate limiting* (mer om dette i kapittel 6.2.10) må backend lagre informasjon om antall forespørsler besøkende til tjenesten har gjort (med besøkende menes ulike IP adresser). Dette gjør at svaret man får kan avhenge av tidligere forespørsler og API-et er derfor ikke "stateless". Det skal sies at informasjonen backend lagrer kan være om flere klienter da flere klienter kan dele samme offentlige IP.

I alle tilfeller brukes de fire HTTP-metodene, `POST`, `GET`, `PATCH` og `DELETE`, til å tilsvare de ulike CRUD-operasjonene som støttes. Det vil si at for eksempel om handlingen er å hente data så er *endepunktet* satt til å bruke `GET`, mens dersom handlingen er å slette data så brukes `DELETE`. Dette er vanlig i en API designet etter REST API-prinsippene.

5.8 Databasedesign

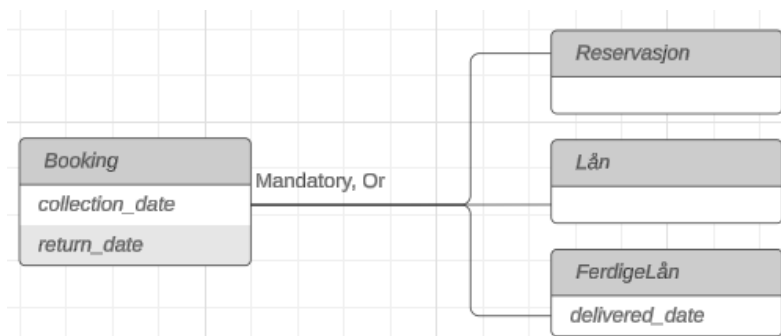
Et viktig aspekt av dette prosjektet er å kunne håndtere nødvendig data på en god måte via å bestemme hvordan dataen skal se ut i databasen. Dette er spesielt viktig med tanke på SQL, siden oppdatering av designet kan være knotete og tungvindt med tanke på å få over eksisterende data over på den nye strukturen.

5.8.1 Normalisering

Normalisering har blitt vurdert i hvert trinn av designet av databasen. Gruppen har diskutert designet og har tatt beslutninger om å nedprioritere normalisering i enkelte tilfeller basert på mulige konsekvenser som kan oppstå senere i utviklingen. Disse mulige konsekvensene angår kompleksitet av SQL-spørringer i backend, hastighet på lasting og innviklet design.

5.8.2 Lån og reserverasjoner

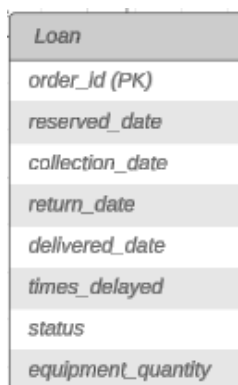
Tabellen for lån (inkludert aktive og ferdige lån) og reserverasjoner besto opprinnelig av fire forskjellige tabeller. En av dem var for de delte attributtene som de alle trengte som id, dato for henting og dato for levering. De tre andre tabellene var for de forskjellige statusene: aktive lån, reserverasjoner og lån som er levert tilbake. Denne strukturen var normalisert slik at det ikke var noe redundans i motsetning til vår nye struktur. I den nye er alle lån og reserverasjoner samlet i en tabell og dermed har alle attributter for alle typer lån selv om de ikke trengs, som for eksempel *delivered_date* (faktisk dato tilbakelevert til Utstyrbanken) for reserverasjoner. Det som gjør at den ikke er normalisert er at attributtet for datoen levert vil også være avhengig av status, som ikke er del av primærnøkkelen. Dette er et av tilfellene hvor vi valgte å prioritere enklere struktur og spørringer i backend.



Figur 10: Alternativ struktur for lagring av lån

En viktig grunn til at vi først ville gå for denne normaliserte versjonen var at vi antok det ville ha vært raskere å hente alle reservasjonene hvis de var i samme tabell og det ikke trengtes å filtrere basert på statusen "reservasjon" i spørringen. Det kommer til å være langt færre reservasjoner enn ferdige lån etter hvert så en "full table scan" hadde vært et problem. Denne antagelsen stemte ikke fordi problemet ville fremdeles vært der ettersom at en slik normalisering hadde krevd sammenslåing med tabellen for de delte attributtene. Uansett er alt som trengs for å unngå en "full table scan" å lage en index på *status*-attributtet slik at databasehåndteringssystemet kan raskere gjøre oppslag basert på *status*.

Ulempen med den normaliserte struktur som gjorde at vi gikk vekk fra den var at den ville potensielt kreve sletting av reservasjoner og oppretting av lån hver gang en reservasjon ble hentet. Og hvis systemet krevde at det ikke ble slettet for tilgang til historikk ville det vært opp til 3 ganger så mange lån og reservasjoner. Vi ville også unngå å måtte gjøre mange *JOIN* operasjoner for å kombinere reservasjoner, lån og ferdige lån til en liste med alle lån.



Figur 11: Valgt design av lån

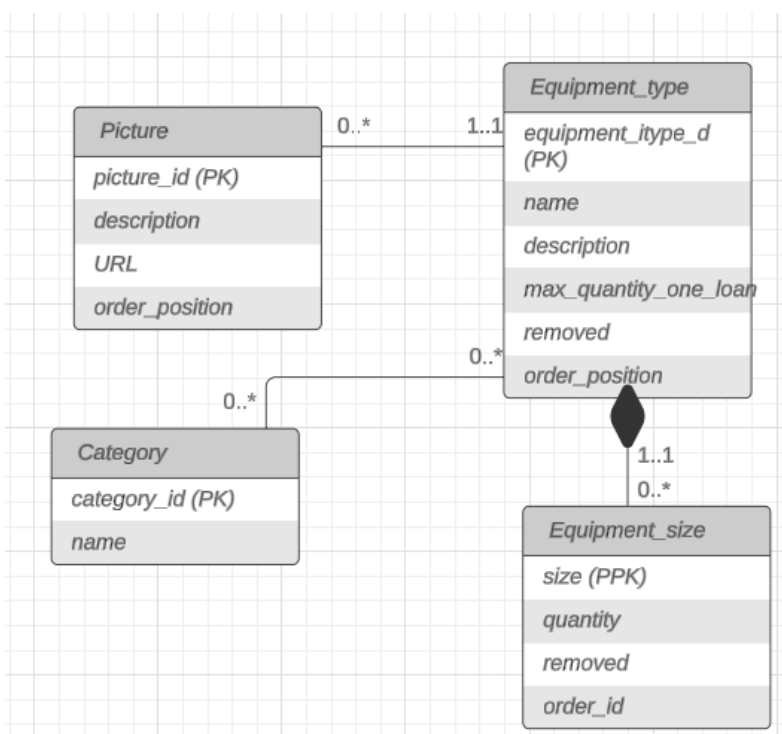
En endring som kom etter første iterasjon av databasedesignet var at lån ikke lenger skulle kunne inkludere flere typer utstyr og størrelser, noe som krevde at forholdet mellom lån og utstyr måtte bli endret fra en mange-til-mange relasjon til en en-til-mange relasjon

5.8.3 Utstyrstyper og størrelser

Det ble avgjort tidlig at designet for lagringen av utstyr ikke skulle involvere å lagre data for hvert enkelt eksemplar av utstyr ettersom at oppdragsgiver ikke var interessert i å spore individuelt utstyr som ble lånt, bare antallet, størrelsen og typen utstyr. Derfor endte vi opp med en tabell for utstyrstyper og en for størrelser.

Vi valgte å bruke en sammensatt nøkkel av utstyr id og navnet på størrelsen i stedet for en id som primærnøkkel for størrelse. Dette valget kom fra normalisering, ettersom at størrelse skulle være unik for hver utstyrstype vil attributter som "quantity" være avhengig av kombinasjonen av utstyrstype id og størrelse attributt. Dette strider med tredje normalform hvis det skulle vært en id for størrelse. Det kunne likevel vært aktuelt med en størrelse id unik for alle størrelser i systemet siden det hadde forenklet fremmednøkler til størrelser, som hadde gjort både andre tabeller og SQL spørringer lettere å håndtere.

Under designprosessen av første iterasjon av databasen ble det vurdert at utstyrstype skulle ha et attributt for måleenhet, og verdien for størrelse i størrelse entitetene var et tall i angitt måleenhet. Vi gikk bort fra dette designet ettersom at enkelte utstyrstyper ikke har størrelser som passer dette mønsteret, for eksempel størrelser som "Medium" eller samlinger av flere størrelser som for eksempel "50-55cm". Det ble mer fleksibelt når vi bare lagret det som tekst, men det ble umulig å sortere det på en måte som ga mening. Derfor introduserte vi "order_id" som lar administrator manuelt velge rekkefølge



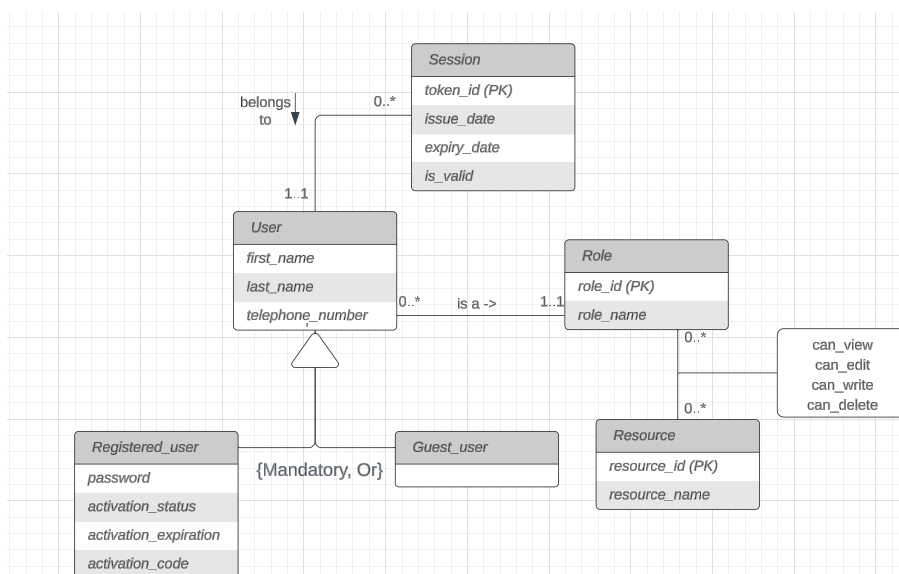
Figur 12: Lagring av utstyrstyper

5.8.4 Brukere, roller og ressurser

Brukertabellene ble først designet som flere tabeller for de ulike rollene. En for vanlige brukere, en for ansatte og en for administrator. Dette designet var unødvendig for Utstyrskontrollens sine krav fordi de forskjellige rollene er såpass like at de kan være i samme tabell uten problemer.

I det nye designet er alle de nevnte rollene i én entitet med et attributt for rolle som pekte til en tabell med de ulike rollene. Med introduksjonen av gjestebbrukere måtte vi likevel ha tre tabeller for brukere. En med data felles til både registrerte brukere og gjestedata og to med data unikt til de to typene brukere.

Ressurs-entiteten ble opprettet for de ulike typene objekter i systemet (brukere, lån osv.) samt til tilganger som muligheten til å registrere en bruker og promotere en bruker til ansatt. For hver rolle og ressurs vil det være handlinger som rollen enten har tilgang til eller ikke. Disse handlingene er de ulike CRUD-operasjonene (lag, se, oppdater og slett).



Figur 13: Hvordan data om brukere, roller og ressurser er lagret

Telefonnummer for registrerte brukere ble i en senere iterasjon gjort til å være "nullable", som betyr at verdien kan være "null" altså ikke eksisterende. Dette gjør det mulig å fjerne alle personopplysninger uten å måtte slette brukeren fullstendig

5.8.5 Historikk og statistikk

Databasen ble designet med behovet for å kunne se oversikt over alle ferdige lån, hendelser i systemet over en lang periode, og statistikk om lån av utstyr.

Det ble vurdert å ha en tabell med statistikk lagret i databasen, men dette ville ha vært unødvendig redundans. Redundansen hadde kommet fra at all statistikken vi trengte allerede kunne bli funnet i tabellen som inneholder lån og reserverasjoner. Med et design hvor vi finner all statistikk fra tabellen for lån må vi unngå at utstyrstyper og deres størrelser blir fullstendig slettet når administrator velger å fjerne en utstyrstype eller størrelse fra listene over utstyrstyper som brukere kan låne og administrator kan redigere.

Det er hvorfor mange av tabellene i databasen har en "deleted"-attributt som sier om entiteten er slettet eller ikke. Hvis det ikke var såkalt *soft deletion* og utstyrstyper og deres størrelser ble slettet fullstendig fra databasen ville vi måtte tatt i bruk to potensielle løsninger.

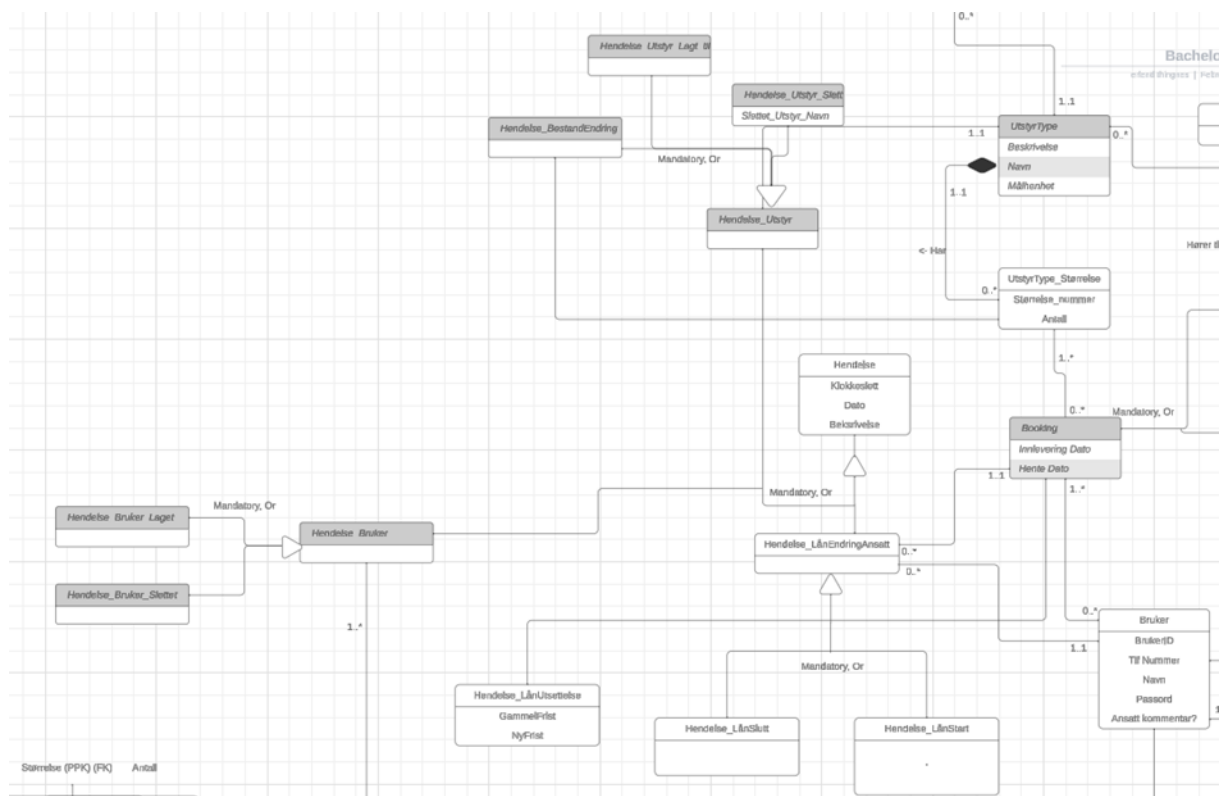
- Den første kunne vært å slette lån når størrelsene eller utstyrstypen de tilhører ikke lenger eksisterer på grunn av at referanseintegritet, en av de viktige egenskapene for databaser, krever at alle fremmednøkklene fra lån til størrelser peker på størrelser som eksisterer.
- Den andre kunne vært å ikke slette lån, men da hadde det ikke vært noe referanseintegritet og det hadde vært lån uten eksisterende utstyrstyper eller størrelser.

En nedside med å ikke slette dataene fullstendig er at databasen bare blir større og større. Hvis det ikke bare var et "removed" flagg som enten var sann eller usann og i stedet var en dato som hadde vært "null" for det som ikke var slettet kunne vi hatt *SQL* events som slettet utstyr og størrelsene og lånene knyttet til disse slettede utstyrstypene

5.8.6 Hendelseslogg

Hendelsesloggen var en utfordring å designe på grunn av alle de forskjellige type hendelsene som kan oppstå. Hendelser som lagrer informasjon om oppretting av utstyr kommer til å trenge fremmednøkkel til utstyrstype, mens hendelser som lagrer informasjon om bekreftelse av levering kommer til å trenge fremmednøkler til lånet som ble levert og til den ansatte som bekreftet det.

Først ble det designet en struktur hvor det ble forsøkt optimal normalisering. Denne hadde forskjellige entiteter for hver type hendelse slik at en hendelse ikke trengte mange attributter den ikke trengte. Vi valgte bort denne løsningen fordi det hadde blitt for komplisert å måtte hente data fra så mange forskjellige databaser for de ulike typene hendelser.



Figur 14: Denne delen av ER diagrammet viser hvordan hendelser for hendelsesloggen er strukturert etter normalisering

For å unngå denne kompleksiteten kom vi på en løsning hvor all informasjonen om hendelsen var i ett attributt som besto av JSON eller tekst. Dette er en langt enklere løsning men den er ikke like fleksibel fordi den ikke tar bruk av fordelene av referanser i en relasjonell database, som at hvis en bruker oppdaterer navnet sitt blir navnet også automatisk oppdatert. Det å måtte skrive kode for å redigere all teksten/JSON-en hver gang en endring blir gjort i den relevante dataen som kan være alt fra utstyrstyper til brukere hadde vært mye arbeid.

Løsningen vi endte opp med var ikke for kompleks og brukte fremmednøkler, men den bryter normalisering ved å ha bare en tabell med mange fremmednøkler som kan være "null" uten å peke til en annen tabell. En hendelse som gir informasjon om henting av lån vil kun måtte bruke fremmednøklerne for ansatt og lån, og de andre fremmednøklerne som den for kategorier vil være null.

5.8.7 SQL events

Etterhvert som tjenesten tas i bruk vil deler av databasen fylles opp med gammel og unyttig data. Dette gjelder blant annet åpningstidene for unntaksdager som har allerede vært samt gamle og/eller ugyldige sesjoner. Av hensyn til ytelse og lagringsplass er det ønskelig å periodisk slette disse dataene.

Det finnes noen måter å løse dette på. En er at backend gjør SQL-spørringene som kreves for å gjennomføre slettingen periodisk ved bruk av *goroutineer*. En annen er å sette opp noen skript som utfører slettingen som kjører periodisk på den virtuelle maskinen som kjører databasen. Men, det viser seg at MySQL allerede har funksjonalitet for å periodisk kjøre SQL-spørringer, *SQL events* [1]. Dette lar oss gjøre opprensningen kun ved å involvere databasen.

I tillegg er det ønskelig at reservasjoner som ikke ble plukket opp i tide også blir kansellert, dette må gjøres riktig slik at antallet tilgjengelig av de ulike utstyrstypene ikke blir feil. Dette kan også gjennomføres via *SQL events*, men å gjøre dette krever to SQL-spørringer, en for å oppdatere antallet tilgjengelig av hver utstyrstype og en for å oppdatere reservasjonene med ny status. Dermed måtte vi sikre at det kjøres som en transaksjon slik at vi ikke risikerer at det blir liggende feil data i databasen dersom for eksempel kun en av spørringene er suksessfulle (kodene for eventene kan sees i vedlegg H).

6 Implementasjon

Dette kapitlet tar for seg implementasjon av programvaren. Kapitlet er delt opp etter frontend- og backend-spesifikke temaer deretter kommer temaer som berører begge komponentene i programvaren vår. Alt ved implementasjonen vil ikke dekkes her, kun det viktigste trekkes fram.

6.1 Frontend-spesifikke temaer

6.1.1 Kodestruktur

Koden for frontend er strukturert som en standard React App. Resten av kodestrukturen for koden i frontend er løst delt inn i disse mappene:

- **public** mappen inkluderer noen komponenter som vises på nettsiden. Dette er i hovedsak ikoner og metadata
- **src** mappen holder selve kildekoden. Den mest spesielle filen her er "App.tsx", som er hovedfilen for applikasjonen
- **api** mappen inneholder funksjoner som blir brukt overalt i koden. I hovedsak inkluderer dette konfigurasjonen for Axios, et Promise-basert bibliotek for å kjøre [Hypertext Transfer Protocol \(HTTP\)](#)-requests
- **content** inkluderer resten av komponentene som blir brukt til å bygge siden vår.

6.1.2 Datastrukturer og Zod

Gjennom utviklingen av frontend har vi hatt behovet for å definere datastrukturer som vi mottar fra backend. Det finnes et par måter å gjøre dette på, som å definere et interface eller en type:

```
// example of an interface
interface Item {
  name: string;
  description: string;
  amount: int;
}

// example of a type declaration
type Item = {
  name: string;
  description: string;
  amount: int;
}
```

Figur 15: Et eksempel på et standard interface og type i TypeScript

Å bruke disse typene fungerer fint for å bruke koden, og man får også beskyttelse fra å bruke feil typer når man kompilerer programmet. Poenget med typer i TypeScript er simpelt: Med tradisjonell JavaScript er objekter formløse. Man kan legge til hva som helst på et objekt, som krever presisjon når man bruker koden. Det er ingen feilhåndtering eller advarsler om objekter inkluderer eller mangler noen felt, som krever mer jobb.

Desverre er det en mangel med TypeScript som kan føre til problemer: informasjonen om typer blir slettet når programmet kjører[38]. Dette er en følge av at TypeScript kompilerer ned til vanlig JavaScript. I vårt tilfelle trenger vi å sende, hente, og passere data fra backend rundt i koden. For å gjøre dette effektivt kreves en løsning som ikke bare kan verifisere koden mens den kompileres, men også mens den kjøres.

For å løse denne utfordringen rundt datastrukturer brukte vi et bibliotek kalt Zod. Zod er svært nyttig i at du kan bygge datastrukturer som kan lese inn data fra filer eller nettverk enkelt.

```
export const item = z.object({
  name: z.string(),
  description: z.string(),
  amount: z.number(),
})
```

Figur 16: Et eksempel på definisjon og bruk av Zod i programmet vårt

6.1.3 Routing

Når man navigerer mellom sider på et nettstedet må man ha en måte å bestemme hva sidene skal gjøre. For vårt bruksområde var React Router¹⁰ en god løsning. React Router lar oss definere flere "Router" inni en "Routers" komponent. Den fungerer med å lese av [Uniform Resource Locator](#)[39] (URL)-en og laster inn den samsvarende klassen. For vårt tilfelle har hver side har sin egen komponent, som er ansvarlig for seg selv og elementer som blir vist på siden. Alle elementer som er lagret utenfor en route vil alltid lastes inn som en del av forventningene med en [Progressiv Web App \(PWA\)](#). Noen eksempler på slike komponenter er navigasjonsbaren, skjemaet for innlogging, og skjemaet for registrering. Alternativt til andre nettsteder så er disse komponentene alltid lastet inn slik at brukeren slipper å navigere vekk fra siden for å logge inn.

¹⁰<https://reactrouter.com/en/main/start/overview>

```
// Root div
<div>
  <Toaster/>
  <Router> // Use the router for all components
    <Login
      isOpen={this.state.openLogin}
      onClose={this.closeLogin}
      onSubmit={this.submitLogin}
      openRegistration={this.openRegistration}/>
    <NavBar/>
    <Routes> // Routes will take you places
      // Navigating here will open the Item Page
      // e.x.
      <Route path={ItemPageConst.path} element={
        <ItemPage
          permissions={this.state.permissions}
          items={this.state.inventoryItems}
          categories={this.state.categories}
          loading={this.state.loading} <ItemPage/>
        }
      />

      <Route path={InventoryPage.path} element={
        <Inventory
          items={this.state.inventoryItems}
          categories={this.state.categories} />
        }
      />

      //...
    </Routes>
  </Router>
</div>
```

Figur 17: Struktur av Router element

6.1.4 Håndtering av state

Et viktig konsept i applikasjonsdesign er det som kalles for state. Ideen med state på et applikasjonsnivå er at du kan bruke en variabel til å bestemme hvordan siden ser ut. Noen eksempler er et tall som bestemmer om en knapp skal vises eller ikke, og en string som brukes til å vise innholdet i en tekstboks. Hvor kode funker som en mal til siden, vil state bestemme hvordan malen fylles ut, og hvilke deler av malen som blir brukt.

React har et helt system som lar deg bruke state for denne hensikten. Med noen enkle deklarasjoner kan man lagre variabler som blir brukt til å presentere sidens innhold på riktig måte. Et eksempel er det følgende:

```

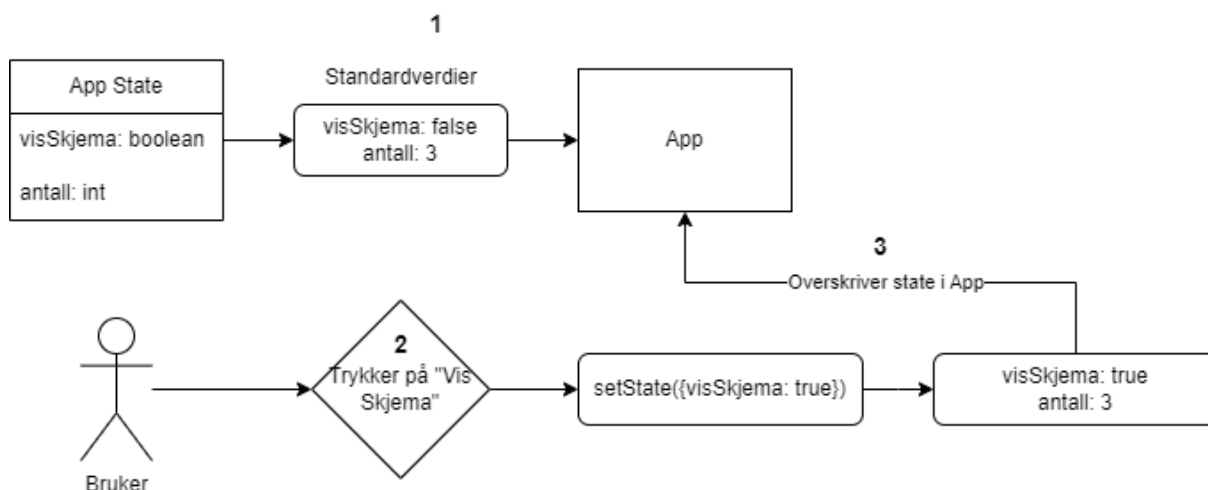
1 interface InventoryState {
2   search: string; // Used for the text box when searching for categories
3   categoriesSearch: Category[]; // Stores a list of which Categories to display
4 }

```

Figur 18: Et kodeeksempel deklarasjon av state

Dette er trukket ut direkte fra InventoryPage komponenten vår, som bestemmer hvilket utstyr som vises i inventaret. Det som gjør state unikt sammenlignet med en statisk variabel er måten man endrer det på. Ved å se på diagrammet 19 nedenfor, går prosessen slik:

1. State blir skapt
2. Noe i applikasjonen gjør et kall som endrer state, f.eks. en knapp blir trykket på
3. En ny state blir skapt hvor verdien er endret på, og den overskriver komponenten sin state



Figur 19: Diagram som viser initialiseringen av state, og stegene som skjer når det endres

Etter en state change vil React laste inn komponenten igjen, med den nye staten, og presentere det til brukeren. Det viktig å være bevisst på når man gjør slike endringer. Man ønsker ikke å laste inn komponenten unødvendig mange ganger, eller å lage en løkke som endrer på state basert på state (som kan gå i det uendelige, eller til programmet kræsjer).

6.1.5 Asynkron programmering

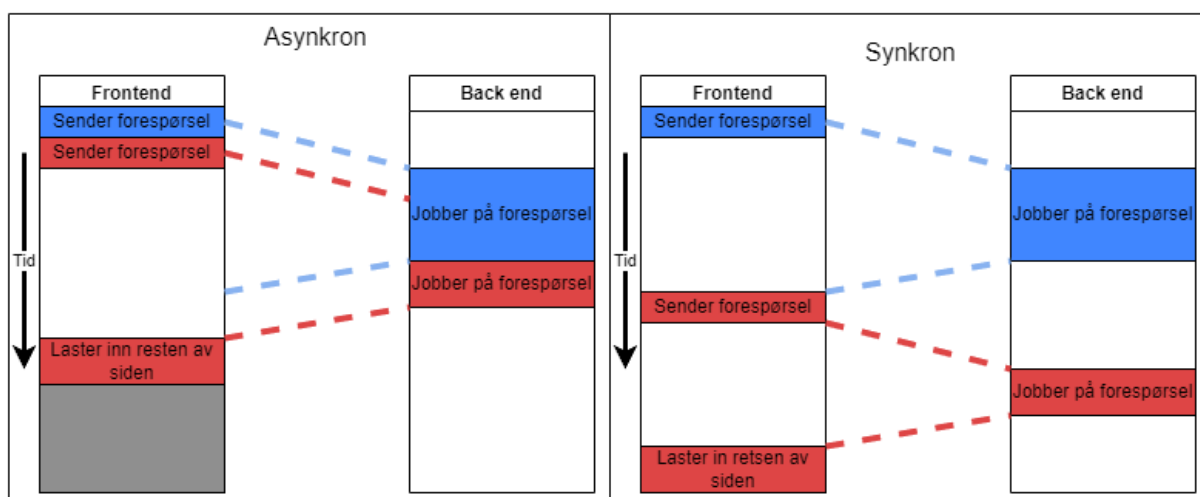
Asynkron programmering handler om å optimalisere ventetid, når man sender en forespørsel til backend så tar det tid å få svar. Selv om responstiden er i millisekunder per forespørsel så vil de dra ut tid siden frontend vil vente på svar før den sender neste forespørsel. Til slutt, først når frontend har fått alle svar fra backend kan den starte å laste inn siden, dette kan derfor lede til mye ventetid spesielt hvis det er trafikk på backend eller hvis bruker har dårlig nett. Asynkron programmering kan hjelpe med dette.

Måten vi har brukt asynkron programmering kan du se i figur 23, både getUser funksjonen og getRoles er begge asynkrone funksjoner som henter data fra backend, det vil si at programmet vil først spørre etter

```
1 // Passed from the component higher in the hierarchy (App)
2 interface InventoryProps {
3   items: ItemData[];
4   categories: Category[];
5 }
6 // This defines what the component holds in its state
7 interface InventoryState {
8   search: string;
9   categoriesSearch: Category[];
10 }
11
12 export class Inventory extends Component<InventoryProps, InventoryState> {
13   constructor(props: InventoryProps) {
14     super(props);
15     // When the page is created, the default values in state are set
16     this.state = {
17       search: "",
18       categoriesSearch: [],
19     }
20   }
21
22   //...
23
24   // Helper function that updates state from search/text input values
25   handleSearchChange = (e: any) => {
26     this.setState(() => ({ search: e.target.value }));
27   }
28
29   // Helper function which modifies the categories stored in the state
30   modifySelectedTags = (tags: Category[]) => {
31     this.setState({ categoriesSearch: tags })
32   }
33
34   render() {
35     return (
36       <div>
37         <div>
38           <input
39             type="text"
40             value={this.state.search}
41             // Update every time the search input field
42             // is changed by the user
43             onChange={this.handleSearchChange}
44             placeholder={"Search"}
45           />
46           //
47           <TagDropdown tags={this.props.categories}
48             modifySelectedTags={this.modifySelectedTags}
49           />
50         </div>
51         //...
52       </div>
53     )
54   }
55 }
56 }
```

Figur 20: Et kodeeksempel på en komponent som bruker både state og props

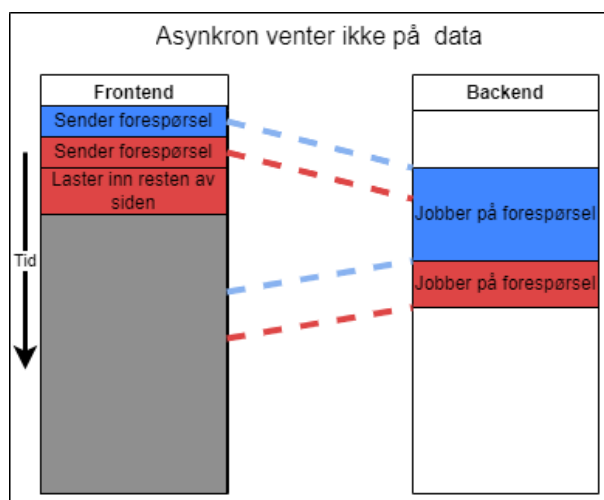
brukerlisten, like etter dette vil den spørre etter roller med getRoles funksjonen, nå som den har spurt begge endepunktene venter den på respons og lagrer responsen i en liste.



Figur 21: Modell som demonstrerer forskjellen mellom asynkron og synkron kode

Dette er raskere fordi backend er bygd for å kunne håndtere flere forespørsler og brukere samtidig, så å gi backend to eller flere oppgaver samtidig er skal ikke være et problem. I tillegg så er det som tar lengst tid i en forespørsel transport av data, så med å kutte ut ventetiden mellom den første og andre responsen har vi allerede der spart mye tid.

Når man bruker asynkron kode så er det en rekke ting man må være oppmerksom på. Først og fremst er det viktig å vente på responsen fra backend, dette var et problem vi hadde i starten, siden å få data fra backend ikke påkaller en state oppdatering, krever det at vi har et system som venter på svar. *Then* er en funksjon som venter på at en asynkron funksjon får svar, før den fortsetter med koden som er inni *then* operasjonen. Løsningen vi hadde i starten var basert på å sette state change funksjon inni en slik *then* operasjon etter den andre forespørselen (den røde på modellen i figur 21).



Figur 22: Asynkron som ikke venter på svar

Dette ville ha funket, hvis det ikke var for at backend er designet for å kunne håndtere mange forskjellige brukere samtidig. En backend er som oftest bygd opp av en database og flere CPU kjerner som kjører parallelt og bruker databasen for å prosessere forespørsler, så for hver gang du sender en forespørsel til en backend så kan det hende at det er en helt annen kjerne som svarer på den forespørselen, og hvis det

er mye trafikk en dag, eller en kjernen som har mottatt den første forespørselen allerede opptatt med en stor forespørsel (den blå på modellen ovenfor), så kan den komme senere en den andre forespørselen. I disse tilfellene vil siden kalle en state change uten å ha fått data fra den første forespørselen, som fører til at ufullstendig data blir vist til bruker. Dette kaller vi en *race condition*. Hvis vi hadde satt en *Then* operasjon på begge responsene ville vi ha kommet full sirkel tilbake til en synkron operasjon.

Promise.all er en funksjon som løste denne *race condition*en, det var faktisk KI som lærte oss om funksjonen, mer om dette i kapittel 9. *Promise.all* er en funksjon som kan gruppere asynkrone funksjoner og kan gruppere alle responsene i en liste. Figur 23 linje 3 til 13 er et eksempel på hvordan vi kombinerte *Promise.all*, *then* og *state change* for å lage en asynkron funksjon som garanterer svar fra alle responser, det eneste problemet med funksjonen er siden den må venter på alle responsene, hvis en feil oppstår med en av dem, vil ingen av responsene bli lagret i state.

```

1  getData = () => {
2      RefreshAuthOnExpiry().then(_ => {
3          Promise.all([
4              this.getUsers(),
5              this.getRoles()
6          ]).then((res) => this.setState(
7              {
8                  guests: res[0].guests,
9                  users: res[0].users,
10                 roles: res[1]
11             })
12         ).catch(err => console.error(err))
13     })
14 }
15
16 getUsers = async(): Promise<{users: User[], guests: Guest[]}> => {
17     return instance.get("/api/users/list")
18         .then((res) => {
19             console.log(res)
20             const parse = userResponse.parse(res.data)
21             return {
22                 users: parse.registered ? parse.registered : [],
23                 guests: parse.guests ? parse.guests : []
24             }
25         })
26         .catch((err) => {
27             console.error(err)
28             return {users: [], guests: []}
29         })
30 }
31
32 //...
```

Figur 23: Asynkron kode med *Promise.all* i brukerliste-side for administratorer

6.1.6 Kopiering av verdier

Det er tilfeller hvor vi har lyst til å lagre en verdi to ganger,. Et eksempel er i koden for modifisering av størrelser hvor det blir brukt to lister av størrelser, en med de originale størrelsene slik de er lagret i databasen, og en annen som blir modifisert. Hvis begge bare hadde vært referanser til listen av størrelser etter den ble hentet fra backend hadde begge listene vært identiske, selv etter endringer blir gjort til en av dem.

For å fikse det måtte vi sørge for at enten den originale listen eller den som skulle kunne bli modifisert var kopier av listen hentet fra backend og ikke bare referanser til samme lokasjoner i minne. Dette ble oppnådd ved å bruke *spread operator* som vises i dette eksempelet:

```
"data.sizes.map(size => ({ ...size })))" // Notice triple dots in front of size
```

Figur 24: Bruk av en spread operator

Hver kopi av *size* objektet i listen fra *data* objektet blir lagt til i en liste. Den typen kopiering kalles *shallow copying* [33] og kopierer bare primitive enkeltverdier som string og number som gjør at `...data.sizes` ikke hadde fungert da `data.sizes` er en array.

Deep copying som kopier verdier som lister og objekter i tillegg til primitive verdier kunne blitt gjort gjennom å bruke `"JSON.parse(JSON.stringify(data.sizes))"` som bruker konvertering til og fra JSON, men *shallow copy* av hver størrelse ble valgt på bakgrunn av at det som regel er raskere å bruke *shallow copying* enn *deep copying*.

6.2 Backend-spesifikke temaer

I dette underkapittelet dekkes temaer som er spesifikke til backend-programvaren i applikasjonen vår. ¹¹

6.2.1 Inndeling av kildekoden

For å organisere koden til backend og gjøre den lettere å jobbe med har koden blitt fordelt utover flere filer som videre er samlet i Go-pakker som representeres med filmapper. Dette vil også gjøre det lettere for de som eventuelt skal ta over og videreutvikle å få oversikt over eksisterende kodebase. Hver "pakke" håndterer sin del av funksjonaliteten som trengs for å kunne gjennomføre all funksjonalitet som trengs av backend. Koden til backend er delt inn i følgende pakker:

- **app**: Denne pakken inneholder kjernefunksjonaliteten ("business logic") til backend. For eksempel, når man ber backend om å lage en ny reservasjon blir logikken for det håndtert her (som sjekk av hentedato og leveringsdato, om det er nok av utstyret, kall til *db*-pakken for å registrere nytt antall tilgjengelig samt selve reservasjonen).
- **clicksend**: Denne pakken inneholder all funksjonalitet knyttet til bruk av ClickSend sitt API. Det betyr funksjoner for sending av SMS om aktiveringskode samt koden som trengs for å bytte passord, malen for meldingene finnes også her.
- **cmd**: Denne pakken definerer alle endepunktene til HTTP-serveren, kaller funksjoner for å opprette en forbindelse til databasen og starter rutiner for opprensning av gammel data knyttet til rate limiting. Dette er hovedpakken og definerer det som skjer ved oppstart av programmet.
- **config**: Denne pakken håndterer innlasting av konfigurasjonen både fra fil og fra miljøvariabler.
- **data**: Denne pakken definerer datastrukturer og faste verdier som brukes av resten av programmet.
- **db**: Denne pakken inneholder funksjoner som utfører alle databaseoperasjonene via SQL-spørringer, essensielt så er denne pakken en måte å kjøre en rekke forhåndsdefinerte spørringer. Denne pakken er svært viktig for funksjonaliteten til backend siden så og si alle handlingene som backend håndterer involverer databasen.
- **error_handling**: Denne pakken inneholder *LogError* som er en funksjon som håndterer hvordan feil blir vist, en mer generisk JSON-feilmelding blir sendt til klient og en mer detaljert feilmelding logges til konsollen. Pakken inneholder også en del forhåndsdefinerte feilmeldinger.

¹¹Alle kodeeksempler har funksjonsnavn, variabelnavn samt kommentarer på engelsk, dette er på grunn av at kildekoden til backend også er på engelsk.

- **handlers:** Denne pakken inneholder såkalte handler-funksjoner, dette er funksjoner som har som oppgave å ta imot innkommende forespørsler og svare dem basert på forespørselen. Dette gjøres i all hovedsak via kall til funksjoner i **app**-pakken for så å returnere svaret derifra som **JSON**.
- **limiter:** Denne pakken inneholder all funksjonalitet knyttet til *rate limiting* (mer om dette i kapittel 6.2.10). Det er også her du vil finne de globale variablene som lagrer informasjon om hvor mange forespørsler hver besøkende (med besøkende menes som regel IP adresse) har gjort slik at *rate limiting* kan gjennomføres.
- **minio_storage:** Denne pakken håndterer kommunikasjon med *MinIO*-serveren for å lagre og hente filene lagret der.
- **util:** Denne pakken inneholder noen hjelpe-funksjoner som brukes av de andre pakkene.

Denne inndelingen skiller kode for å kommunisere med de andre tjenestene backend er avhengig av (*db* for MySQL databasen, *clicksend* for ClickSend APIen og *minio_storage* for MinIO serveren), dette bidrar for å få oversikt og gjøre det lettere for andre deler av applikasjonen å benytte seg av tjenestene.

6.2.2 Konfigurasjon

Fra starten av innså vi at det hadde vært svært nyttig å kunne konfigurere en del variabler etter behov. En del av disse er knyttet til oppsettet som adresse, brukernavn og passord til databasen, innloggingsdetaljer til tredjepartstjenester eller om API-et skal bruke **TLS** i kommunikasjonen med frontend. Andre går mer på applikasjonen selv som f.eks. hvor lenge *Access Tokens* skal være gyldig eller om man skal gjennomføre verifisering av telefonnummer ved registrering.

Konfigurasjon kan gjøres på mange ulike former, via kommandolinjeparаметre, konfigurasjonsfiler og miljøvariabler. Siden det er snakk om ganske mange ulike variabler som kan konfigureres blir fort kommandolinjeparametere uhandterbart. Konfigurasjonsfiler er fint under utvikling, men er vanskeligere å bruke sammen med Docker ved utrulling. For Docker-baserte applikasjoner er det enklere å bruke miljøvariabler. Av hensyn til både utvikling og utrulling ble det derfor bestemt at backend skal kunne konfigureres både via konfigurasjonsfil og via miljøvariabler.

Når det kommer til format for en konfigurasjonsfil er det ønskelig å benytte et format som gir en viss struktur og som kan både enkelt endres av mennesker og leses inn av programvaren. Man kan bruke **JSON** til dette formålet, men **JSON** mangler muligheten til å legge inn kommentarer som kan forklare hva de ulike variablene brukes til. Alternativet vi endte opp med å bruke er **YAML**-formatet, det tillater kommentarer og det finnes en pakke for Go som gjør det mulig å enkelt konvertere mellom datastrukturene i Go og **YAML** tilsvarende det som allerede eksisterer for **JSON** i standardbiblioteket.

Dermed endte vi opp med følgende prosedyre for å laste inn konfigurasjonen for backend:

1. I koden er det definert en standardverdi for konfigurasjonen. Dersom hverken konfigurasjonsfilen eller miljøvariablene er definert som noe annet, er det denne som brukes. Dette hindrer backend i å kræsje ved for eksempel manglende konfigurasjonsfil eller manglende variabler i konfigurasjonsfilen.
2. Dersom det ikke eksisterer en konfigurasjonsfil, blir den laget med standardverdiene definert i koden.
3. Konfigurasjonsfilen lastes inn og overskriver standardverdiene
4. Miljøvariablene sees igjennom og overskriver eventuelt verdiene lest inn fra konfigurasjonsfilen.

Siden miljøvariablene leses inn etter konfigurasjonsfilen vil de ha prioritet over det som er definert i konfigurasjonsfilen, dette ble gjort siden det er miljøvariablene som brukes under utrulling.

6.2.3 Standardprosess

Når backend mottar en forespørsel fra frontend om å utføre en handling er det et standard oppsett som stort sett følges. Alle forespørselene blir håndtert av en såkalt handler-funksjon som er knyttet til et spesifikt endepunkt. Disse finnes som hovedregel to typer av i backend. Den ene kan kalles ”enkel-handler” som betyr at den kun kan utføre en handling via en [HTTP-metode](#), den andre kan kalles ”multi-handler” og kan utføre flere ulike handlinger avhengig av hvilken [HTTP-metode](#) som brukes.

Prosessene som følges for en ”enkel-handler” kan beskrives slik:

1. [HTTP](#)-serveren mottar forespørselen fra klient og kaller riktig handler-funksjon basert på forespurt URL.
2. Forespørselen logges til konsoll av handler-funksjonen.
3. Det sjekkes at man ikke bruker en ugyldig [HTTP-metode](#).
4. Mer detaljer om hva som blir forespurt blir hentet fra URL ([path parameter](#), [query parameter](#)) og/eller fra medsendt [JSON](#). Dette kan være for eksempel hvilken bruker man skal hente lånene til eller navn, telefonnummer og passord til en ny bruker som skal registreres.
5. Videre brukes medsendte informasjonskapsler for å autentisere klienten og sjekke om rollen til (innlogget) klient har tilgangen på ressursen som kreves for å gjennomføre handlingen.
6. Deretter kalles den relevante funksjonen i *app*-pakken for å gjennomføre den forespurte handlingen. Denne funksjonen vil som regel gjennomføre et eller flere kall til *db*-pakken for å hente, oppdatere, sette inn eller slette noe fra databasen for å kunne gjennomføre handlingen som er forespurt.
7. Svar (returverdi) fra funksjonen i *app*-pakken blir konvertert til [JSON](#) og sendt til klient.

Et eksempel på en slik ”enkel-handler” kan sees i vedlegg [G.1](#).

Forskjellen med en ”multi-handler” er at handler-funksjonen som brukes kan gjøre flere ulike handlinger, hvilken som blir utført er avhengig av hvilken [HTTP-metode](#) forespørselen bruker. For eksempel kan handler-funksjonen som endrer brukerdata også brukes til å slette brukere, hva som gjøres avhenger om det er en [PATCH](#)-forespørsel eller [DELETE](#)-forespørsel som sendes.

Kort oppsummert betyr det at:

- Handler-funksjonene håndterer valg av handling basert på forespørsel ([HTTP-metode](#), URL-sti), henter ut inndata fra forespørselen ([path parametere](#), [query parametere](#), [JSON](#)) samt gjør kall for å autentisere og autorisere forespørselen ved å validere informasjonskapslene sendt med i forespørselen. Til slutt sendes resultatet av handlingen tilbake til avsender som respons i form av [JSON](#).
- Funksjonene i *app*-pakken håndterer resten av logikken som kreves for å utføre handlingen som regel via et eller flere kall til database-funksjonene. Da de fleste handlingene som backend skal gjennomføre er av typen [CRUD](#)-operasjoner. App-funksjoene vil også returnere svar-objektet som sendes til avsender som [JSON](#) til handler-funksjonen.
- Funksjoner som ligger i *db*-pakken utfører alle [SQL](#)-spørringene som kreves for å gjennomføre de fleste av handlingene som backend håndterer.

Enkelte handler-funksjoner følger ikke denne standardprosessen, det skyldes hovedsaklig at de ble skrevet før standardprosessen ble definert. Dette gjelder i hovedsak handlerene som håndterer autentiseringsrelaterte forespørsler. Det bør derfor vurderes å skrive om disse handler-funksjonene til å bruke standardprosessen for å være konsistent med de andre handler-funksjonene.

Grunnen til at vi har valgt til å dele inn prosessen på denne måten handler blant annet om å gjøre det enklere å skrive enhetstester for de viktige funksjonene i *app*-pakken samt reduksjon av duplisert kode, særlig gjelder dette database-funksjonene. I løpet av utviklingen har vi gjort endringer til databasen som har krevet omskriving av database-funksjoner og dersom omskrivningen ikke endrer dataene som returneres vil man ikke trenge å omskrive noe annen kode for å kunne fungere med de nye endringene i databasen.

6.2.4 Autentisering og autorisering

Autentisering blir gjort via informasjonskapsler som blir generert via en suksessfull forespørsel på innloggingsendepunktet. Dataene i informasjonskapslene er av typen **JWT** (JSON Web Token [34]). En **JWT** [34] er en måte å enkode **JSON**-data på med en signatur. Med signaturen kan backend bekrefte at det er den selv som har skapt **JWT**en og den har ikke blitt tuklet med siden den ble skapt. Dermed forhindres falske informasjonskapsler som kunne blitt brukt til å late som man er logget inn som en valgfri bruker. En **JWT** består av tre deler, informasjon om hvilken type signatur som er brukt, **JSON**-objektet samt signaturen selv. Alle deler bruker **base64** enkoding. I **JSON**-objektet lagres blant annet skapelsesdato, utløpsdato samt hvilken bruker **JWT**en er knyttet til. I **JWT** standarden defineres det standardfelt, disse brukes for aktuell informasjon, ellers benyttes egendefinerte felter for informasjon som navn og rolle ID.

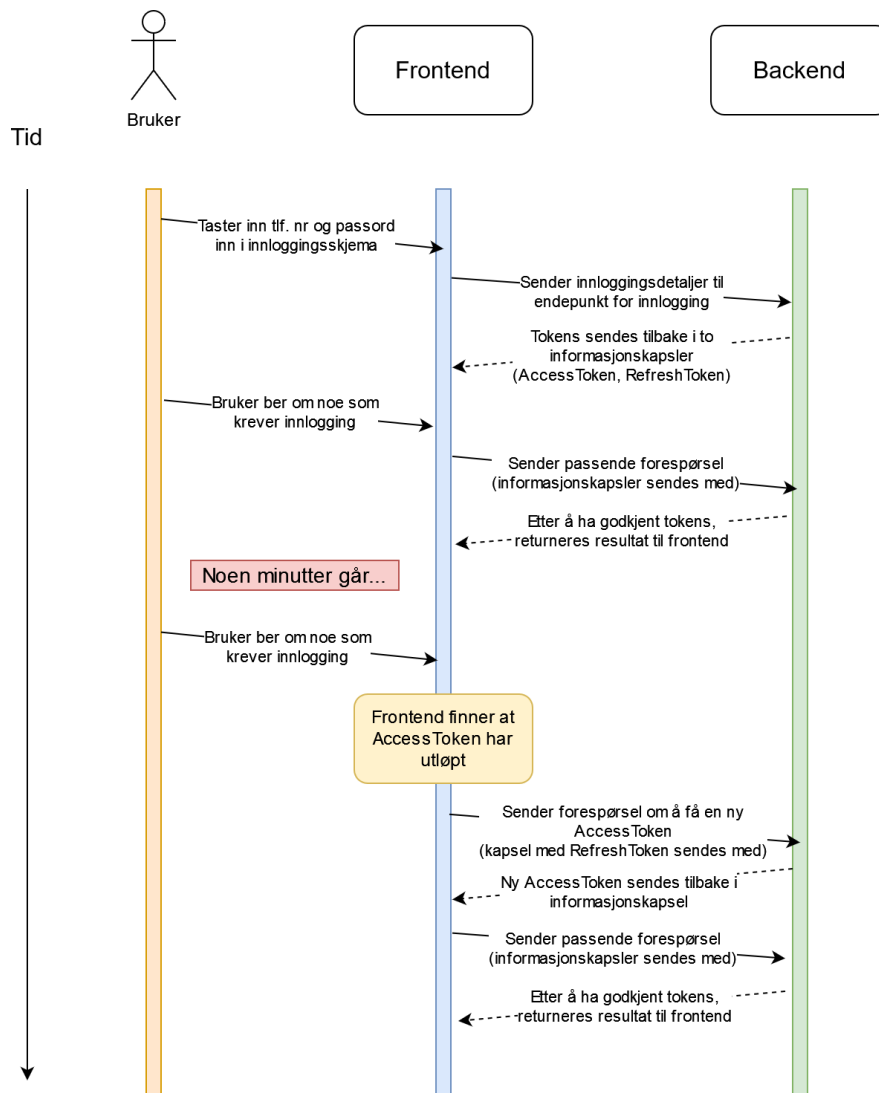
Grunnprinsippet til en **JWT** er at når den er skapt, trenger man ikke lagre sesjondata serverside siden man kan bruke informasjonen fra **JWT**en og bruke signaturen til å bekrefte at dataene ikke er forfalsket. Ulempen med dette er at det gjør det umulig å ugyldiggjøre **JWT**er for eksempel ved utlogging eller passordbytte. Derfor har vi valgt en slags mellomløsning der vi har to informasjonskapsler, en *AccessToken* som ikke kan ugyldiggjøres og en *RefreshToken* som er knyttet til sesjondata i databasen og dermed kan ugyldiggjøres. *AccessTokens* er kun gyldig i korte perioder og må regelmessig genereres på nytt, man bruker en *RefreshToken* med en gyldig tilknyttet sesjon for å gjøre dette.

Dermed når backend ønsker å autentisere en klient vil den se etter om forespørselen inneholder en informasjonskapsel kalt *AccessToken*, dekode informasjonen lagret der samtidig som den verifiseres via signaturen og utløpsdato sjekkes. Dermed har backend både ID til brukeren samt ID til rollen som brukeren har. Sistnevnte brukes for å sjekke tillatelser, mer om det senere.

For enkelte forespørsler/handlinger vil backend også se etter informasjonskapslen med *RefreshToken* for å bekrefte at klient har en gyldig aktiv sesjon. Her dobbeltsjekkes også rolle ID med det som er lagret i databasen. Dersom rolle ID har nylig endret seg for brukeren vil ikke dette vises med en gang i *AccessToken*. I koden kalles dette ”*Strict Auth*”. Handlinger som krever slik ekstra autentisering vil være handlinger som sletting av utstyrstyper og registrering av returnert lån.

For å utføre en handling er det ofte ikke nok med autentisering, men også autorisering. Backend har nemlig også den rollen å sjekke at autentisert bruker har lov til å utføre handlingen som blir forespurt. Dette gjøres ved å sjekke om rollen til brukeren har en tilgang på ressursen som er knyttet til handling. Som nevnt tidligere, kan man ha en eller flere følgende tilganger på ressurser, altså: *view*, *write*, *edit*, *delete* (tilsvarer typisk **CRUD**).

For å redusere unødvendig databaseaksess og siden data om hvilke roller har hvilke tilganger er omtrent statisk blir dette lastet inn i minnet i backend ved oppstart. Dermed trenger man ikke å kontakte databasen hver gang man ønsker å sjekke om en spesifikk rolle har tilgangen som kreves for å utføre handlingen. Dette har sideeffekten at man må kjøre omstart av backend hver gang man ønsker å endre på hvilke tilganger de ulike rollene har, det vil sannsynligvis være sjelden å endre på dette under normal drift så det er ingen stor ulempe.



Figur 25: Sekvensdiagram som viser autentisering (med innlogging og evt. henting av ny AccessToken om nødvendig)

6.2.5 Om passord

For å overholde et av sikkerhetskravene satt i kapittel 2, må det brukes en *hash*-algoritme for å sikkert kunne lagre passord. Her fant vi raskt BCrypt-algoritmen som brukes via *bcrypt*-pakken (denne er del av *crypt*-biblioteket til Google). BCrypt-algoritmen har en *cost* innstilling som vil bestemme hvor vanskelig det er å gjenopprette den opprinnelige verdien så høyere *cost* vil være sikrere. Ulempen er at dette krever mer prosessorkraft og dermed ta lenger tid, man må derfor finne et kompromiss. Siden det er usikkert hvor kraftig serveren er som vil ende opp med å kjøre backend i applikasjonen, er denne *cost* innstillingen konfigurabel i backend via både konfigurasjonsfil og miljøvariabler.

En begrensning i BCrypt-algoritmen er at passord kan ha en maks lengde på 72 bytes. Det er derfor lagt inn en begrensning på maks lengde på passord som godkjennes ved registrering og passordbytte. Nøyaktig hvor mange tegn det svarer til vil være avhengig av hvor mange bytes som kreves for å lagre tegnene som brukes, da enkelte tegn vil kreve flere bytes enn andre. For øyeblikket er det ikke satt noen flere krav til passord i backend.

Proessen er slik at ved registrering vil passordet til brukeren kjøres igjennom hashing-algoritmen før passordet lagres i databasen. Ved innlogging vil passordet som ble sendt i forespørselen kjøres igjennom samme algoritme og deretter sammenlignes med hashen som er lagret i databasen. Siden passordene sendes i klartekst til backend blir det viktig å sikre kommunikasjonen mellom klient og backend slik at passordene ikke blir eksponert for uvedkommende aktører.

6.2.6 Sikring av informasjonskapsler

Som nevnt tidligere, sender backend med informasjonskapsler slik at klient kan bruke de for å autentisere seg. Når man sender med informasjonskapsler er det noen innstillinger som bør settes for å sikre at onsinnde aktører ikke får tilgang på kapslene. Dersom de får tak i informasjonskapslene vil de muligheten til å stjele sesjonen til brukerne og gjennomføre handlinger som om de var den brukeren, dette er spesielt alvorlig dersom det er en administrator sin sesjon som stjeles med tanke på at administratorer har økt tilgang.

En slik innstilling er *SameSite* som har tre mulige verdier [7], *Strict*, *Lax* og *None*. *SameSite* definerer om informasjonskapslen skal sendes med når forespørsler skjer fra andre nettstedet. Informasjonskapslene i vår applikasjon har sin *SameSite*-verdi satt til *Strict* slik at ingen av de sendes med ved bruk av slike forespørsler da disse kan brukes til *CSRF-angrep*.

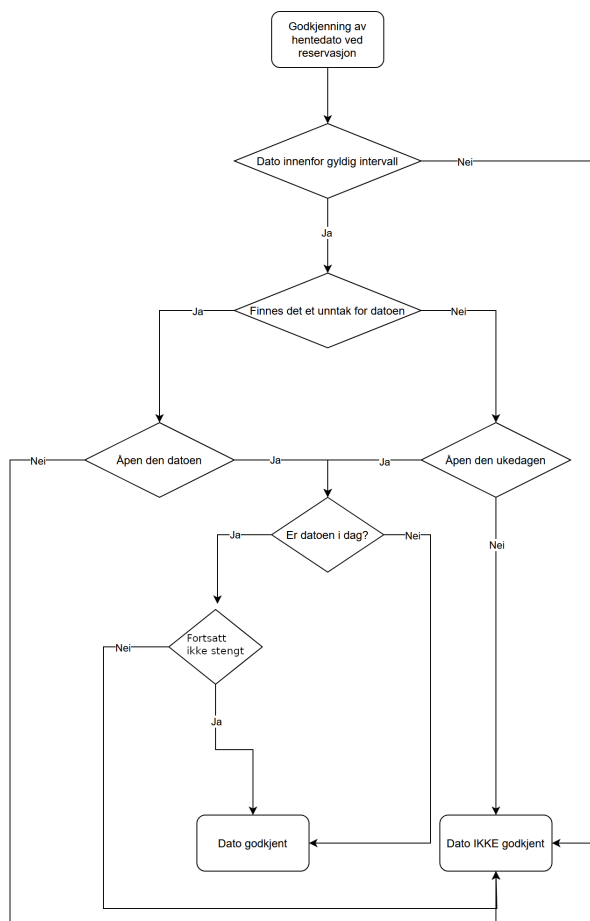
De to andre innstillingene er *HttpOnly* og *Secure* [7]. Førstnevnte bestemmer om JavaScript kjørende på klienten skal ha tilgang på informasjonskapslene eller ikke. Det bør ikke JavaScript ha fordi da kan onsinnde aktører potensielt kapre sesjonen til brukerne via et *XSS-angrep* som bruker JavaScript til å la angriperen få tak i informasjonskapslene. Sistnevnte bestemmer om kapslen kun kan sendes med forespørsler som bruker *HTTPS*. Dette bør aktiveres siden dersom siden plutselig ikke bruker *HTTPS* så er noe galt med siden som kan indikere et forsøk på et angrep mot siden.

Disse tiltakene bidrar til at sikkerheten knyttet til sesjonen ivaretas og at kun brukeren som logger inn kan benytte sesjonen sin.

Det viser seg at en liten tabbe ble gjort med *SameSite*, da ved utlogging erstattes informasjonskapslene med kapsler som bruker *SameSite* satt til *None*. Siden disse nye kapslene ikke inneholder tokens, er det i utgangspunktet ikke et problem. Det viser seg derimot at dersom man vil bruke *SameSite None* må kapslen også bruke *Secure* [36] ellers vil ikke nettleseren akseptere disse nye informasjonskapslene. Dermed vil frontend beholde de gamle informasjonskapslene og bruker vil fremstå som fortsatt innlogget inntil *AccessToken* løper ut, dersom backend er konfigurert til å ikke bruke *TLS*. Løsningen ville vært at kapslene brukte *SameSite* satt til *Lax* istedet når *TLS* ikke er i bruk.

6.2.7 Godkjenning av datoer

Når en bruker ønsker å reservere utstyr skal brukeren, som nevnt tidligere i kapittel 2, velge en hentedato for når brukeren ønsker å hente utstyret og en leveringsdato for når brukeren ønsker å levere utstyret. Utifra hvordan åpningstidene fungerer (se kapittel 5.6) og verdiene som er satt for hvor langt i fremtiden det er mulig å reservere ser prosessen med å bestemme om datoen blir godkjent eller ikke ut som sånn den er beskrevet i figur 26.



Figur 26: Flytskjema som beskriver godkjenningsprosessen av hentedato for reservasjoner.

Med *gyldig intervall* i figur 26 menes det at datoen er mellom dagens dato og hvor mange dager inn i fremtiden en reservasjon kan starte. Sistnevnte er implementert slik at den kan endres etter ønske fra Utstyrskontrollen.

Prosessen for godkjenning av leveringsdato er veldig lik, bortsett fra at man ikke trenger å sjekke om datoen er i dag og dermed også om Utstyrskontrollen er stengt for dagen, det betyr selvfølgelig at leveringsdato ikke kan være i dag. Her må man istedet i tillegg sjekke om leveringsdato er etter hentedato.

I backend implementeres dette via en funksjon som henter alle godkjente datoer (se vedlegg G.2 for koden) og ser om valgte datoer er mellom de. Det gjøres via en for-loop som sjekker alle datoer innenfor spesifisert intervall, for hver dato sjekkes det om Utstyrskontrollen er åpen den ukedagen og dersom det er et unntak for datoen så vil den ha prioritet over vanlig åpningstid. Etter å ha laget flytskjemaet for rapporten, viser det seg at en mulig optimalisering kunne ha blitt gjort. I koden sjekkes det først på de ordinære åpningstidene så deretter unntaksdatoene, det hadde vært bedre å sjekke på unntaksdatoene

først da det ikke er nødvendig å sjekke på de ordinære åpningstidene dersom datoen er en unntaksdato.

6.2.8 Bruk av prepared statements i SQL-spørringene

Dersom en SQL-spørring inneholder inndata man får utenifra for eksempel fra klient kan man risikere at inndataen tolkes som SQL-instruksjoner. Dersom dette er tilfellet vil man være sårbar for SQL-injeksjonsangrep [43] som lar ondsinnede aktører gjennomføre database-operasjoner de ikke skal ha muligheten til å gjøre.

```
1 // Prepare session select statement
2 stmt, err := db.Prepare("SELECT * FROM sessions WHERE token_id=?")
3 if err != nil {
4     // Handle error
5 }
6 defer stmt.Close()
7
8 // Run session select statement
9 res := stmt.QueryRow(tokenId)
10
11 // Scan data from database
```

Figur 27: Bruk av "prepared statement" i Go for å hente en sesjon fra databasen. (?) representerer parameteren som brukes når spørringen kjøres med *QueryRow*-funksjonen.

Løsningen er å bruke prepared statements [40] som vil si at man kompilerer spørringen med instruksene og placeholderne der man senere setter inn parametrene/inndataene. Deretter kan man kjøre spørringen ved å sette inn parametrene/inndataene, og siden spørringen allerede er compilert vil MySQL kunne skille mellom SQL-instrukser og inndata, dermed skal ondsinnede aktører ikke kunne gjennomføre SQL-injeksjonsangrep.

Alle SQL-spørringer i backend bruker prepared statements der man bruker inndata som kan komme fra klient (et eksempel kan sees i figur 27), dersom spørringen ikke involverer inndata trenger man ikke å bruke en prepared statement siden risikoen for et SQL-injeksjonsangrep oppstår kun når brukerstyrt inndata er involvert.

Et unntak til reglen er spørringen som henter data om flere brukere, det var nemlig en begrensning fra pakken vi brukte for å kommunisere med databasen at man ikke kunne bruke arrays som parameter i en prepared statement. Dermed ble løsningen å konvertere arrayen med tall tilbake til en string, se koden i figur 28.

```
1 // Convert array into string
2 // [1,2,3] -> 1,2,3
3 ids := ""
4 for idx, uid := range userIds {
5     ids += strconv.Itoa(int(uid))
6     if idx+1 < len(userIds) {
7         ids += ","
8     }
9 }
10
11 // Setup query to fetch user data
12 sqlQuery := fmt.Sprintf("SELECT * FROM `user` WHERE user.user_id IN (%s)", ids)
13
14 // Run selection statement
15 rows, err := db.Query(sqlQuery)
16 if err != nil {
17     // Handle error
18 }
19
20 // Scan data from database into Go structures
```

Figur 28: Deler av koden som henter data om flere brukere fra databasen. Koden er noe modifisert fra opprinnelig kildekode.

Inndataen her er en string fra klient (endepunktet bruker en [query parameter](#)) på formen "1,2,4" som så konverteres til en array med tall [1,2,4] og deretter tilbake igjen. Dermed skal det være umulig å kunne passere tekst til spørringen som kjøres og dermed også SQL-instruksler som betyr at i teorien skal også denne spørringen være beskyttet mot SQL-injeksjonsangrep. Derfor regnes det ikke som et problem at denne ikke bruker en prepared statement.

6.2.9 SQL-transaksjoner

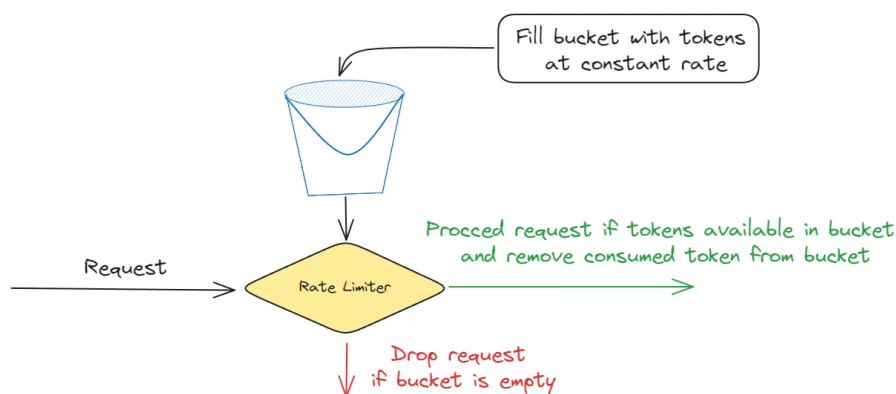
I en del tilfeller er det ønskelig at dersom man ikke klarer å gjennomføre alle SQL-spørringene som kreves for å gjennomføre en handling uten feil så skal alle endringer som ble gjort ruller tilbake. Et eksempel på når dette er ønskelig er når en ny reservasjon skal opprettes. Da må spørringen som setter inn den nye reservasjonen og spørringen som reduserer antallet tilgjengelig av utstyret som reserveres skje uten problemer. Dersom kun en av de kjører uten problemer vil databasen inneholde feil data. For da risikerer man enten at det blir reservert mer utstyr enn det er tilgjengelig (fordi reservasjonen ikke førte til at antall tilgjengelig ble redusert) eller at det i databasen er registrert mindre antall utstyr som tilgjengelig enn det faktisk er (antallet ble redusert, men reservasjonen ble ikke lagt inn).

For slike problemer finnes det et konsept kalt transaksjoner i de fleste varianter av SQL, inkludert MySQL [41]. Med en transaksjon kan man kjøre flere SQL-spørringer, dersom en feiler kan man velge å rulle tilbake alle endringer som har skjedd hittil som del av transaksjonen. Dersom alle spørringene gjennomføres uten problemer blir endringene satt til å være permanente.

I pakken vi bruker for å kommunisere med databasen er støtte for SQL-transaksjoner innebygget og brukes i alle tilfeller der det er nødvendig.

6.2.10 Rate limiting

For å redusere påvirkningen spam mot backend har er det blitt implementert *rate limiting* på backend, det betyr at dersom man sender for mange forespørsler for fort blir man sperret i en kort tidsperiode ifra å sende flere forespørsler. Spesielt viktig er dette for handlinger som fører til sending av SMS-er som for eksempel registrering av brukere. Dette er fordi sending av SMS-er koster penger, dermed vil spamming føre til reelle kostnader for Utstyrskontoret. Derfor vil disse endepunktene ha en strengere form for rate limiting der antall forespørsler man kan gjennomføre innen et gitt tidsrom er mye lavere. I tillegg er det også ønskelig å kunne begrense hvor mange innloggingsforespørsler hver besøkende kan utføre, det gjør det vanskeligere å gjennomføre såkalte brute force-angrep ved at angrepet vil ta mye lengre tid enn det ellers ville ha gjort på grunn av rate limiting.



Figur 29: Figur som beskriver tokenbøtte-algoritmen. Hentet fra [46]

Rate limiting er implementert ved bruk av *rate*-pakken [24] fra Google. Den implementerer en *Limit*-type som vi kan bruke til å bestemme om forespørselen tillates eller ikke, *Limit*-typen benytter *tokenbøtte-algoritmen* for å bestemme om forespørselen skal gå igjennom eller avslås. Rate limit-funksjonen tar en handler-funksjon som parameter, den kjøres dersom forespørselen tillates og dermed kan rate limit-funksjonen gjenbrukes for alle endepunktene.

Men, koden i figur 30 har et problem. Her implementeres det en rate limiter som har en global begrensning på hvor mange forespørsler alle brukere kan gjennomføre, dette vil føre til at dersom ondsinnede brukere spammer tjenesten vil forespørselene til alle brukere av tjenesten bli sperret av rate limiting. Man vil dessuten måtte øke antall forespørsler før rate limiting inntreffer når antallet aktive brukere øker. Løsningen på dette ble å kunne sette en begrensning som gjelder for hver besøkende på siden, dette må også inkludere gjestebukere så vi kan ikke for eksempel lagre en *Limit* basert på bruker ID. Dermed er det ikke mye som er unikt vi kan bruke for å identifisere en besøkende, et alternativ er IP-adressen til klienten. Det vil fortsatt føre til at flere besøkende blir lagt under samme *Limit* siden alle på samme nettverk vil ha en felles offentlig IP-adresse som er den som blir synlig for backend, men det vil likevel redusere problemet.

```
1 // Instance of Limiter-type, first number is number of tokens to "refill the bucket" every second
2 // , defined as the maximum number of requests allowed per second
3 // , and second parameter is "bucket capacity" otherwise known as "burst"
4 var limiter = rate.NewLimiter(10,20)
5
6 func RateLimiter(next http.HandlerFunc) http.HandlerFunc {
7     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
8
9         // Check if limiter disallows the request
10        if limiter.Allow() == false {
11            // Handle rate limited error
12            error_handling.LogError(w, r, error_handling.RATE_LIMITED_ERROR)
13            return
14        }
15
16        // Run handler function to handle the request
17        next.ServeHTTP(w, r)
18    })
19 }
```

Figur 30: Rate limiting med *rate*-pakken i sin enkleste form.

Limiter-dataene blir lagret i minne da det er ingen behov for å lagre disse dataene permanent. For å unngå stadig økende minneforbruk dersom backend får kjøre over tid, er det satt igang en *goroutine* som periodisk fjerner limitere til besøkende IP-adresser som ikke har brukt backend på en stund. Siden *goroutiner* kjører parallelt må *mutex-låser* brukes for å unngå *race condition* når den modifierer den globale variabelen som inneholder Limiterene.

Som nevnt tidligere, vil man at innlogging skulle ha strengere begrensninger enn andre forespørsler. Dette implementeres ved at innlogging reguleres av egne Limiterer som har mye strengere begrensninger. Utfordringen her er om man ønsker å støtte at mange ulike brukere ønsker å logge på fra samme IP-adresse (tenk ved større nettverk som skoler og mobilnett) vil man måtte sette disse begrensningene ganske høye. Opprinnelig var ideen å ha en Limiter for hvert telefonnummer (hver registrert bruker har et unikt telefonnummer), men det kan føre til en lignende problematikk som ved bruk av en global limiter. Ondsinnede brukere kan spamme innlogging for å effektivt stenge ute brukere fra å kunne logge inn. Løsningen ble dermed å lagre Limiterer for hver kombinasjon av IP-adresse samt telefonnummer. Da kan vi støtte mange brukere fra en IP-adresse samtidig som at muligheten for ondsinnede å kunne utestenge brukere ved å trigge rate limitingen blir sterkt redusert. De vil riktignok fortsatt kunne gjøre det om de har samme IP-adresse som brukeren som forsøker å logge seg inn.

6.2.11 Feilhåndtering

```
1 returnVal, err := Function(parameter)
2 if err != nil {
3     // Handle error
4 }
```

Figur 31: Typisk feilhåndtering i Go

I Go er standard måte å håndtere feil på at funksjonen returnerer to eller flere verdier der siste verdi er en referanse til en eventuell feilmelding, deretter kan man bruke en simpel if-sjekk for å se om en feil skjedde, se figur 31. [19] Dette gjør det relativt enkelt å håndtere feil og gi en nyttig respons tilbake om hva som gikk galt. Dessverre fører dette til at man på repeterer if-sjekken ganske ofte i koden. Man vil også måtte sjekke for hvert lag når man har en funksjon som kaller andre funksjoner som skjer ganske ofte i backend. For eksempel dersom det skjer en database-relatert feil når man har et endepunkt som henter data om en bruker:

1. I database-funksjonen (*db*-pakken) må man sjekke etter feil med samme type if-sjekk som ble beskrevet i figur 31 ved hvert steg av prosessen for så å returnere informasjon om feilen et lag lengre opp.
2. I app-funksjonen (*app*-pakken) sjekker man etter feilen som ble returnert av database-funksjonen med samme type if-sjekk, her velges det å returnere og dermed videresendes feilen enda et lag opp.
3. I handler-funksjonen (*handler*-pakken) sjekker man etter feilen igjen og videresender dette til *LogError*-funksjonen til *error_handling*-pakken som tar seg av å logge feilen til konsollen samt en feilmelding i JSON-format til klient.

Når man returnerer feilmeldinger via **HTTP** indikerer man typen feil via **HTTP-statuskoder**. Disse er noe begrenset i seg selv, det er ikke alltid like lett å velge riktig **statuskode** å returnere, men i de fleste tilfeller vil backend returnere passende **statuskode** som *404* dersom ressursen ikke ble funnet eller *406* om noe ikke er riktig med forespørselen. Frontend benytter kun **statuskoden** for å bestemme hvilken feil som vises, dette er fordi feilmeldingene backend sender er på engelsk. En mulig løsning er at vi kunne ha vurdert å definere våre egne feilmeldingskoder slik at man kunne vist tilsvarende norske feilmeldinger i frontend, det ble ikke implementert på grunn av prioritering av annen viktigere funksjonalitet.

For å gjøre det lettere å returnere korrekt **statuskode** og kunne gi en egen feilmelding til klient brukes vår egen datatype kalt *AppError*. Dermed kan feilmelding og beskrivelse til klient, **statuskode** til respons samt en standard Go error-type slik at feilmelding fra andre biblioteker ikke blir "glemt". Denne kan da returneres av funksjonene i programvaren.

I *error_handling*-pakken finnes funksjonen *LogError* (se figur 32 som nevnt tidligere. Pakken inneholder også standardiserte feilmeldinger som for eksempel feil under dekoding av medsendt **JSON** i forespørselen og at brukeren ikke har tillatelse til å utføre handlingen. Dette har som formål å redusere duplikat kode og gjør det enklere å endre ordlyden på feilmeldingene dersom man ønsker det senere.

```
1 func LogError(w http.ResponseWriter, r *http.Request, appError *data.AppError) {
2     // Print error message to console
3     log.Printf("Error processing %s request on URL %s\n", r.Method, r.URL)
4     printError(appError)
5     // Assemble response to client
6     response := data.AppError{
7         StatusCode: appError.StatusCode,
8         Text:         appError.Text,
9         Description: appError.Description,
10    }
11    // Set header and status code to return
12    w.Header().Set("Content-Type", "application/json")
13    w.WriteHeader(appError.StatusCode)
14    // Send error to client as JSON
15    err := json.NewEncoder(w).Encode(response)
16    if err != nil {
17        http.Error(w, "error returning error details", http.StatusInternalServerError)
18        log.Println(err)
19        return
20    }
21 }
```

Figur 32: LogError, brukes til å logge de fleste feilmeldinger som oppstår både til konsoll og til klient

6.2.12 Håndtering av bilder

Som nevnt i kapittel 4, var det en del usikkerhet knyttet til hvor vi burde lagre bildene. Men, en ting er sikkert, frontend bør ikke snakke med systemet som lagrer bildene direkte. Ved å la frontend laste opp og hente bildene via backend, har man muligheten til å kunne regulere tilgangen ved å bruke vårt eksisterende rollebaserte aksesskontrollsystem. I tillegg slipper vi å direkte eksponere MinIO-serveren til internett.

Backend må også verifisere at filene som lastes opp faktisk er bilder, samt at filstørrelsene ikke er for store. For øyeblikket sjekkes det kun på filtype (altså filetternavnet, for eksempel *.jpeg*) som er lett å lure ved å endre på filetternavnet. En mer involvert sjekk bør derfor vurderes i fremtiden slik at man sikrer at kun bilder kan lastes opp.

Ved opplasting brukes en spesiell datatype kalt *multipart-form*. Med en slik type forespørsel kan man sende med både bildet og metadata i en og samme forespørsel, dette fungerer via nøkkel-verdi par der verdien kan enten være tekst eller fildata (se vedlegg G.3 for å se hvordan backend håndterer en *multipart-form*). Alternativet ville vært å sende kun selve bildet og deretter sende metadataen i en egen forespørsel som JSON, det er ikke helt ideelt. Det kan skje at bildet lastes opp, men forespørselen med metadataen feiler. Likevel er dette fortsatt en utfordring selv med bruk av en *multipart-form* siden backend må lagre metadataen i MySQL-databasen og selve bildefila på MinIO-serveren. Ved opplasting er det implementert slik at bildet lastes opp til MinIO først, deretter settes metadataen inn i databasen. Dersom sistnevnte feiles, forsøkes det å slette bildet fra MinIO.

Ved sletting av bildet kan det gjøres både ved en DELETE-forespørsel mot [endepunktet](#) for metadataen og [endepunktet](#) for selve bildet (se vedlegg J.6 for detaljert informasjon om hvordan denne delen av API-et brukes). Begge vil fortsatt forsøke å slette både metadataen og selve bildet, eneste forskjell er rekkefølgen. I vanlig bruk er det anbefalt å slette bildene via DELETE-forespørsler mot [endepunktet](#) for metadataen siden når metadataen mangler vil ikke frontend forsøke å laste inn bildet lengre selv om det viser seg at sletting fra MinIO feilet. I det motsatte tilfellet vil frontend forsøke å laste inn et bilde som

ikke eksisterer som vil se mindre pent ut.

6.3 Registrering av brukere

I backend skjer registrering av brukere via POST-forespørsler mot [endepunktet](#) `/api/users/register` (se vedlegg J.4 for mer detaljer om hvordan [endepunktet](#) brukes). I backend håndteres dette via standardprosessen som beskrevet tidligere, her skal vi se nærmere på funksjonen i `app`-pakken som tar seg av registrering.

Det første som skjer er validering av inndataen om den nye brukeren, det vil si blant annet å sjekke om telefonnummeret er innenfor gyldig intervall samt om etternavnet ikke er tomt. Videre brukes inndataen til å konstruere et bruker-objekt, her hashes passordet samt dersom verifisering er skrudd på vil en aktiveringskode for brukeren også genereres. Dermed kan brukeren settes inn i databasen. Dette gjøres ved å starte en `SQL-transaksjon` og deretter kjøre funksjonen fra `db`-pakken som setter inn brukeren i databasen. En SMS med aktiveringskode sendes, dersom dette feiler vil transaksjonen bli avbrutt og brukeren blir ikke satt inn. Til slutt returneres brukerdata tilbake til handler-funksjonen som da sender dataene som `JSON` tilbake til klient.

Backend må også håndtere verifiseringen av brukerne, dette gjøres ved å sende `POST`-forespørsler til [endepunktet](#) `/api/users/verify` (se vedlegg J.4 for mer detaljer om hvordan [endepunktet](#) brukes) med telefonnummer og aktiveringskode som ble sendt på SMS til telefonnummeret spesifisert ved registrering. Måten dette gjøres på i backend er relativt enkelt. Etter å ha fått handler-funksjonen til [endepunktet](#) til å håndtere dekoding av `JavaScript Object Notation` [31] (`JSON`) i forespørselen så kjøres funksjonen for aktivering av bruker i `app`-pakken. Der hentes brukerdata assosiert med telefonnummeret, da spesifikt den registrerte brukeren på telefonnummeret. Deretter sjekker man om brukeren allerede er aktivert, før man ser om medsendt aktiveringskode stemmer med lagret kode samt at koden ikke har utløpt. Dersom alle sjekkene er godkjente blir brukeren aktivert og kan nå logge inn.

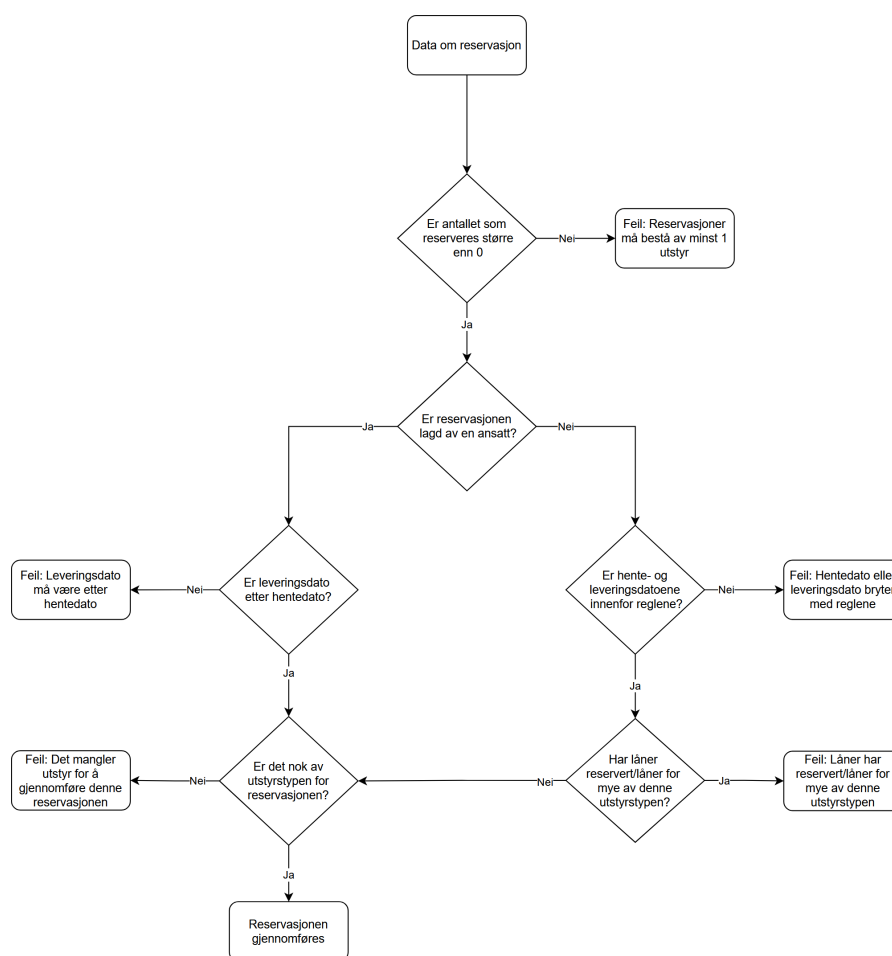
Dersom aktiveringskoden er utgått har backend også implementert funksjonalitet for å opprette en ny aktiveringskode med ny utløpsdato, dette gjøres enkelt via `PATCH`-forespørsler til [endepunktet](#) `/api/users/resend` (se vedlegg J.4 for mer detaljer om hvordan [endepunktet](#) brukes) som vil da oppdatere aktiveringskode og utløpsdato i databasen for brukeren spesifisert med telefonnummer.

6.4 Oppretting av reservasjoner og lån

En viktig del av applikasjonen er muligheten for vanlige brukere, altså besøkende til Utstyrsbanken, å kunne reservere utstyr. Dette ble tidligere nevnt i kapittel 2.

Oppretting av reservasjoner er faktisk en nokså komplisert prosess. I teorien er det ganske enkelt, man må sjekke om det er nok utstyr tilgjengelig for å gjennomføre reservasjonen, redusere antallet tilgjengelig av størrelsen som reserveres og deretter registrere reservasjonen i databasen.

I backend håndteres dette av handler-funksjonen som håndterer `POST`-forespørsler på [endepunktet](#) `/api/reservations/create` (se vedlegg J.3 for hvordan [endepunktet](#) brukes). Etter å ha dekodet medsendt `JSON` og autorisert klienten kalles funksjonen i `app`-pakken for å lage en ny reservasjon.



Figur 33: Flytskjema som beskriver sjekkene for å lage en ny reservasjon, for nærmere beskrivelse av sjekkene på datoene se kapittel 6.2.7

Det første som gjøres i funksjonen er å validere inndataene, det vil si å sjekke om antallet som reserveres er høyere enn 0, deretter vil de videre sjekkene være avhengig av om det er en ansatt eller om det er en vanlig bruker/låner som klient er autentisert som. Dette bestemmes av hvilken tilgang rollen til brukeren har, skrive-tilgang på ressursen *ReservationSelf* betyr i praksis at man regnes som en vanlig bruker mens dersom brukeren har skrive-tilgang på ressursen *Reservation* vil brukeren regnes som ansatt. Grunnen til at det skiller på dette er at ansatte skal kunne ved behov ignorere en del regler som er satt for reserverasjoner.

Dersom man er en vanlig bruker må hentdato og leveringsdato for reservasjonen overholde reglene som er satt for de, det vil si blant annet at Utstyrbanken er åpen de dagene og de ikke er for langt inn i fremtiden, for mer informasjon om datosjekkene se kapittel 6.2.7. Deretter sjekkes det om reservasjonen fører til at maks grensen på hvor mange gjenstander man kan låne av utstyrstypen vil overskrides. Til slutt må man sjekke om det finnes nok tilgjengelig utstyr av størrelsen man vil reservere, implisitt sjekkes det også her om størrelsen eksisterer siden dersom størrelsen ikke eksisterer vil det være umulig å hente hvor mange som er tilgjengelig. For ansatte vil man kun sjekke om datoene ikke er i fortiden og om leveringsdato er etter hentdato og om det er nok utstyr tilgjengelig i størrelsen som skal reserveres.

Når alle disse sjekkene er utført kan reservasjonen endelig legges inn i databasen, i tillegg må antallet tilgjengelig av størrelsen reduseres tilsvarende antallet som reserveres. Siden det kun er ønskelig om at begge operasjonene er suksessfulle og man vil helst unngå scenariet der kun en er suksessfull, brukes det

en SQL-transaksjon for å gjennomføre registreringen av reservasjonen.

Registrering av nye lån fungerer mye på samme måte som med reservasjoner, men siden det i utgangspunktet ikke skal være tillatt for vanlige brukere å lage lån så fungerer lån på samme måte som for reservasjoner gjør for ansatte. En forskjell er at man ikke trenger å sende med hentedato det antas at utstyret hentes idag siden lån skal kun registreres dersom utstyret samtidig hentes, hvis ikke skal det registreres som en reservasjon.

Ved skapelsen av et lån kan man også i medsendt JSON sende med en reservasjons-ID for å lage lånet på grunnlag av reservasjonen, man kan enten omgjøre reservasjonen direkte til lån som vil være det vanligste, men også kansellere reservasjonen mens lånet blir laget dersom lånet avviker fra det som ble reservert. Et eksempel kan være at reservasjonen er for 2 fotballer når man egentlig bare trengte 1. Da må man kansellere reservasjonen først slik at det reserverte utstyret regnes som tilgjengelig igjen og deretter registrere lånet. Muligheten til å gjøre det direkte i en operasjon forhindrer muligheten for at noen andre kan reservere utstyret i mellomtiden mellom kanselleringen og registreringen ved at hele prosessen kjører i en SQL-transaksjon.

6.5 Oppdatering av inventaret

En annen viktig del av applikasjonen vår, som nevnt i kapittel 2, er at administrator ved Utstyrsbanken skal kunne oppdatere inventaret, altså legge til, oppdatere og slette utstyrstyper, kategorier og størrelser.

I backend finnes det et flertall av endepunkter for å håndtere de ulike CRUD-operasjonene som trengs på inventaret, både når det gjelder utstyrstyper, størrelser, kategorier og bilder til utstyrstypene. De fleste av disse er nokså rett frem og sjekker relevante tillatelser før passende SQL-spørring kjøres for å gjennomføre handlingen som er forespurt.

```

1 func ArrayUpdates(oldArray []uint, newArray []uint) ([]uint, []uint) {
2     var newElements []uint
3     var removedElements []uint
4
5     // Find new elements
6     for _, newEl := range newArray {
7         found := false
8         for _, oldEl := range oldArray {
9             if oldEl == newEl {
10                found = true
11            }
12        }
13        if !found {
14            newElements = append(newElements, newEl)
15        }
16    }
17
18    // Find old elements
19    // Return the arrays

```

Figur 34: Utdrag fra hjelpefunksjonen som finner nye og slettede elementer i en array med positive heltall (uint). Disse representerer unike verdier gitt til hver kategori

Et eksempel der det kreves litt mer logikk er når man vil endre hvilke kategorier en utstyrstype hører til siden måten dette lagres på i databasen er rader i en egen tabell som forbinder kategoriene med utstyrstypene, dette er standard for et slikt mange-til-mange forhold (en utstyrstype kan høre til mange

kategorier, en kategori kan inneholde mange utstyrstyper) i en SQL-basert database. Slik det ble valgt å implementere det så skal det ved oppdatering av hvilke kategorier en utstyrstype hører til sendes med en oppdatert liste over kategoriene. For å effektivt gjøre de nødvendige endringene i databasen vil backend finne ut hvilke kategorier som er nye (der koblingen må opprettes i databasen) og hvilke kategorier som er fjernet (der koblingen må fjernes i databasen).

Dette gjøres via en hjelpefunksjon (se figur 34) som tar inn den gamle og den nye listen og deretter returnerer to lister. En med nye elementer og en med slettede elementer. Først går igjennom og ser etter elementer som ikke finnes i den gamle listen og legger så disse til en ny liste som da representerer de nye elementene. Videre vil den gå igjennom og se etter elementer som ikke finnes i den nye listen som da er elementene som ble fjernet.

6.6 Statistikk

Fra oppgaveteksten (se vedlegg K) samt funksjonelle krav (se kapittel 2) var det ønskelig fra Utstyrbanken å kunne enkelt se statistikk over hvor mye av hver utstyrstype som ble lånt i en tidsperiode, samt å kunne se statistikk basert på kategoriene. Slik det er implementert regnes et lån som å være innenfor den valgte perioden dersom hentdatoen til lånet er innenfor perioden, det vil si at et lån som starter i slutten av januar og slutter i begynnelsen av februar vil ikke telle på statistikken for februar.

I backend er statistikk implementert slik at man enkelt henter den ved å utføre GET-forespørsler mot endepunktet `/api/stats`. Statistikk er implementert slik at klient enkelt kan spesifisere ønsket tidsperiode med to query parametere (se vedlegg J.5 for mer detaljer rundt selve bruken av endepunktet) Etter å ha autorisert klienten kalles relevant funksjon fra `app`-pakken.

```
1 SELECT
2 equipment_type.equipment_type_id,equipment_type.name,loan.equipment_size,
3 COUNT(order_id) AS Number_of_loans,SUM(loan.equipment_quantity) AS Quantity_loaned
4 FROM
5 `loan` INNER JOIN equipment_type ON loan.equipment_type_id = equipment_type.equipment_type_id
6 WHERE
7 (loan.status = "current" OR loan.status = "finished") AND (loan.collection_date BETWEEN (?) AND (?))
8 GROUP BY
9 equipment_type.equipment_type_id,loan.equipment_size;
```

Figur 35: SQL-spørring som brukes for å hente antall lån og antall lånt utstyr i perioden. ? representerer start- og sluttdato, disse blir satt inn når spørringen kjøres.

Måten statistikk blir hentet på er via to SQL-spørringer som teller opp antall lån samt antall utlånt utstyr fra aktive og ferdige lån i tidsperioden som er forespurt. Den ene teller opp basert på størrelser av utstyrstypene (se figur 35) og den andre teller opp basert på hvilke kategorier utstyrstypene som er lånt hører til. Dersom en utstyrstype hører til flere kategorier vil lånene av denne typen telle på alle kategoriene som typen har. Førstnevnte spørring gir antallene basert på størrelsene, dermed vil backend legge sammen tallene for hver størrelse av en utstyrstype slik at man også får samlet tall for hver utstyrstype. I tillegg vil navn på størrelser, utstyrstyper og kategorier inkluderes for å redusere antall forespørsler når dette skal vises i frontend.

6.7 Datohåndtering

Under utviklingen av programvaren var det noen utfordringer knyttet til datohåndtering, både i frontend og i backend. Dette skyldes at frontend, backend og MySQL databasen alle har sine egne måter å håndtere datoer og tidspunkt på og i applikasjonen må disse kunne samhandle med hverandre.

6.7.1 I frontend

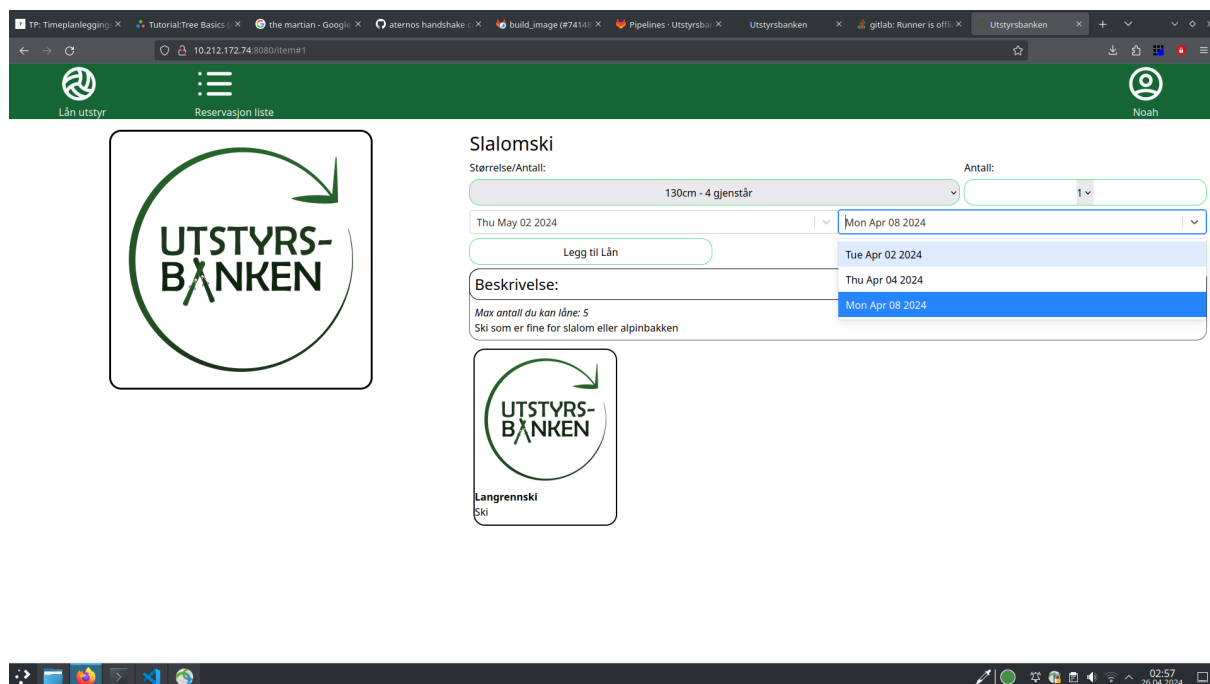
Vi ville unngå å designe og implementere en egen komponent for en kalender med dager som kan bli trykket på så vi valgte å importere "react-day-picker"¹². Vi lagde to tilpassede "DayPickerComponent" komponenter, hvorav den ene var for valg av dato for statistikk og den andre var for valg av unntak på åpningsdager. Begge to tar i mot en prop som inkluderer funksjonen for håndtering av valgt dato, men for statistikk kreves det at to datoer blir valgt. DayPicker komponenten for statistikk bruker verdien "range" i prop for modus slik at et datointervall kan bli valgt. Dette intervallet blir vist på kalenderen ved hjelp av "modifiersStyles" som blir brukt til å gi en unik CSS styling til alle datoene på kalenderen som følger kravet av at de er innenfor intervallet.

"DayPicker" komponenten for åpningsdagunntak bruker en prop kalt "disabled" for å ikke la administrator velge datoer som allerede har vært. Kalenderne hadde originalt navnet på ukedagene på engelsk. For å lokalisere kalenderen til norsk importerte vi et Locale objekt fra "date-fns/locale" med data for norsk bokmål og satte kalenderen til norsk via DayPicker komponenten sitt "locale" prop.

Valg av dato blir også brukt uten "DayPicker" komponenten når bruker velger datoene de vil hente hente og levere lånt utstyr på. Tilgjengelige dager for henting blir hentet fra backend, men for å finne dager som er tilgjengelige for levering finner koden fram hvilke dager som ikke er på stengte ukedager og spesifikke dager valgt som stengt. Den gjør dette ved å bruke en for-løkke som går gjennom alle dagene fra dagen som ble valgt for henting til den dagen som er lengst inn i framtiden, avgjort av verdien for max antall dager et lån kan være lagret i databasen. Først blir datoen som skal bli sjekket om den er gyldig som en dato for levering funnet, så blir det sjekket om den datoen er i listen av åpne ukedager. Hvis den er det blir den lagt til i listen over tilgjengelige dager for lån så lenge den ikke også er i listen over spesifikke datoer markert som stengt. Datoen blir også lagt til hvis den er i listen over spesifikke dager markert som åpen. Dette tar ikke høyde for at en dato kan ha være en på en ukedag kan være markert som åpen og samtidig være en spesifikk dato som åpen, noe som vil resultere i at datoen blir gjentatt. Dette hadde ikke vært et problem hvis det ikke var for at administrator skal kunne overskrive klokkeslett for åpning og stengning på spesifikk dato på en ukedag som vanligvis er åpen

I frontend hadde vi et problem rundt datovelger på siden for en `utstyrstype`. Når datoene inkluderte et månedskifte eller årsskifte, ville datovelgeren alltid ha feil måned og/eller år på sluttdato. Feilen kom fra feil bruk av funksjonen `setDate` når den ble brukt til å gjøre dato definert som "new Date()", som gir dagen i dag, om til dagen som skal være sjekket som gyldig i for-løkken. Siden `setDate` bare setter dagen i måneden og ikke årstallet og måneden kan den ikke bli brukt til det formålet. Det vil resultere i at bare dagen er riktig og måneden og årstallet alltid vil alltid være det den er i dag. Løsningen til problemet var å aldri deklare datoer som skal bli sjekket som gyldig som "new Date()", og heller deklare den som datoen lånet blir hentet på til å begynne med.

¹²<https://react-day-picker.js.org/>



Figur 36: Når datoene inkluderte et månedskifte, ville datovelgeren alltid ha feil måned på sluttdato

6.7.2 I backend

I Go finnes det kun en type for datoer og tidspunkt, *time.Time*, som da inneholder både dato og tidspunkt. Å sende disse i et format som er forståelig for frontend er relativt enkelt, ved enkoding til JSON med Go sitt [standardbibliotek](#) ble de på standardformatet ISO 8601 (eksempel 12:20:00 17/05/2024 med tidssone UTC blir `2024-05-17T12:20:00Z`) som er den internasjonale standarden for utveksling av dato og tidspunkt.

Datoene hentes ofte fra databasen og i MySQL finnes det tre ulike typer, *DATE*, *TIME* og *DATETIME*, hvilken som brukes avhenger av om det er ønskelig å lagre dato, tidspunkt eller begge. Med `sql`-pakken klarer backend å lese *DATE* og *DATETIME* uten problemer inn i Go sin *time.Time*, men *TIME*-typen skapte utfordringer.

```
1     var startTime []uint8
2     var closingTime []uint8
3
4     // Scan the data, code removed from this example, not relevant
5
6     // Parse the byte arrays
7     day.StartTime, err = time.Parse("15:04:06", string(startTime))
8     if err != nil {
9         return nil, error_handling.ProcessingError(err)
10    }
11    day.ClosingTime, err = time.Parse("15:04:06", string(closingTime))
12    if err != nil {
13        return nil, error_handling.ProcessingError(err)
14    }
15
16    schedule = append(schedule, day)
```

Figur 37: Henting av *TIME*-verdier fra databasen. Eksemplet her er innlesning av data om åpningstidene.

Måten det ble løst på var å lese inn tidspunktene i byte-arrayer også bruke *time*-pakken (del av Go sitt standardbibliotek) sin parser for å konvertere byte-arrayene til *time.Time* (se figur 37). Disse får standarddato 01/01/2000. Dette er viktig å huske på f.eks. dersom man ønsker å sammenligne disse med nåværende tidspunkt fra *time.Now()* da denne vil returnere et *time.Time* objekt med dagens dato også.

7 Kvalitetssikring

Dette kapitlet omhandler hvilke tiltak vi har for å sikre at høy kvalitet på produktet. Det innebærer i hovedsak selve koden samt brukergrensesnittet.

7.1 Code review og merge requests

For å sikre at koden har en viss kvalitet og fange opp diverse småfeil må alle endringer i kodebasen igjennom en inspeksjonsprosess. Det vil si at man utvikler en del av programvaren i en egen separat kopi (egen Git-branch), når man sier seg ferdig opprettes en såkalt [merge request](#) der man beskriver endringene sine. Da får man minst en annen gruppemedlem til å se igjennom endringene. Deretter vil [merge requesten](#) enten godkjennes og endringene blir del av hovedbranch (ferdig godkjent kode") eller skrive kommentarer på det som bør rettes opp før endringene kan settes inn. Dette sørger for at koden som brukes i produksjon faktisk holder en viss grad av kvalitet, det vil avhenge av hvor nøye man inspiserer endringene i [merge requesten](#).

7.2 Dokumentasjon

En viktig rutine som ble etablert tidlig er at alle [endepunkter](#) til backend skal dokumenteres med hvilke parametre som brukes ([query parametere](#), [path parametere](#) og/eller [JSON](#) som skal sendes med). Formålet er å gjøre det lettere for utviklerne av frontend å kunne bruke [API-et](#). Måten det ble gjort på er å skrive dokumentasjonen i beskrivelsen av [merge requestene](#) til backend etterhvert som nye [endepunkter](#) ble lagt til. Videre skal dette samles inn i wikien på GitLab slik at det er oversiktlig og enkelt å finne fram.

7.3 Testing

7.3.1 Brukertesting

Fra starten av planla vi å gjennomføre brukertester for å få tilbakemeldinger på brukergrensesnittet. Disse tilbakemeldingene ville vi deretter ha brukt for å forbedre brukergrensesnittet og gjøre nettsiden lettere å bruke. Dette er med på å sørge for at brukergrensesnittet gjør jobben sin.

Brukertestene vil ha blitt gjennomført med to ulike grupper brukere. Dette er fordi gruppene har ganske ulike [gjøremål](#) på nettsiden. Den ene er lånere/kunder som ønsker å reservere utstyr samt se statusen av sine reserverasjoner og lån. Den andre er de ansatte ved Utstyrbanken som ønsker å kunne registrere utlån og tilbakeleveringer samt oppdatere inventaret til Utstyrbanken. Måten vi ønsker å gjennomføre brukertestene på er at vi har forberedt en del oppgaver som skal gjennomføres, oppfordre brukeren til å si høyt hvordan brukeren tenker når brukeren skal gjennomføre oppgavene. Deretter samler vi inn tilbakemeldinger ved å stille dem noen spørsmål.

Dessverre rakk vi ikke gjennomføre brukertesting, dette skyldes i hovedsak at frontend ikke var klar til tiden vi hadde satt av til brukertesting. Utviklingen av frontend gikk tregere enn vi hadde planlagt. I en brukertest må man ha en fungerende brukergrensesnitt for å kunne gjennomføre testene og denne ligger på frontend. I det applikasjonen var klar for brukertest, måtte vi heller fokusere på rapportskrivning.

7.3.2 Stresstesting

Under utviklingen av programvaren er det vanskelig å vite hvor effektiv koden er og hvor mange brukere som programvaren klarer å håndtere før den bryter sammen. Dette er nyttig å kunne vite slik at man kan iverksette tiltak dersom det viser seg at programvaren ikke klarer å håndtere nok brukere som den realistisk sett skal. For å finne ut av dette kan man gjennomføre stresstesting som vil si å simulere et scenario der programvaren blir utsatt for mye trafikk. Deretter kan man måle blant annet ressursbruk (dvs. prosessorbruk, minnebruk osv.) og responstid dvs. hvor lang tid programvaren bruker på en forespørsel når den er under høy belastning.

Vi kunne testet backend ved å for eksempel sette opp en rekke utstyrstyper med et høyt antall tilgjengelig for så å sende mange forespørsler om å reservere utstyr. For å sende nok forespørsler vil vi nok trenge flere klienter som kan sende parallelt. Deretter kan vi måle svartiden og se hvordan den utvikler seg.

7.3.3 Automatiserte tester i backend

For å sikre at backend fungerer som det skal, hadde vi planlagt å sette opp automatiserte tester som tester at koden er riktig. Dette sikrer at tidligere utviklet funksjonalitet fortsatt fungerer, det er viktig ved større omskrivninger av koden som da databasestrukturen ble endret for å støtte [soft deletion](#) av brukere.

Det finnes hovedsaklig to måter å gjøre dette på. Den ene er *enhetstester* der man ville ha testet en av funksjonene ved å kalle dem med kjent testdata for deretter å sjekke om responsen stemmer overens med det som er forventet. Her bør man dekke mange ulike tilfeller via ulike testdata, det inkluderer testdata der man forventer å få en feil tilbake. Som nevnt i kapittel 6, ved å ha kjernefunksjonaliteten i *app*-pakken, separert fra *handler*-pakken, kan vi teste funksjonaliteten uten å måtte sette opp en HTTP-forespørsel. Den andre måten er *HTTP-testing* der man tester ved å sette opp en midlertidig HTTP-server og sende forespørsler mot den, da får man testet hele prosessen inkludert handler-funksjonene.

En utfordring med å gjennomføre *enhetstester* og *HTTP-tester* av backend er at man må simulere/etterligne tjenestene, kalt "stubbing" og "mocking", som backend kommuniserer med. Den viktigste ville vært databasen, det gjør implementasjon av disse testene til en ganske omfattende prosess.

Dessverre rakk vi ikke å implementere disse testene, det skyldes i hovedsak mangel på tid og at implementasjon av ny funksjonalitet ble prioritert over å skrive tester. På grunn av at de fleste funksjonene som er aktuelle å teste bruker databasen så vil det kreve at databasen simuleres under testene, det gjør det ganske tidkrevende å implementere som vi da prioriterte til annen funksjonalitet.

7.4 Andre krav til kildekoden

- Dersom det ikke er en selvfølge hva koden gjør, bør en forståelig forklaring skrives som kommentar i nærheten av relevant kodesnutt. Dersom man er i tvil, bør koden kommenteres. Det er bedre med litt for mye kommentarer enn for lite.
- Relevante standarder for blant annet navngivning av variabler skal følges.
- Variabel- og funksjonsnavn skal navngis slik at de er forståelige i kontekst. Det vi si blant annet å unngå forkortelser der det ikke er åpenbart hva de betyr.

Disse kravene eksisterer slik at kildekoden blir lettere å lese og forstå. Det gjør det enklere når programvaren skal videreutvikles av andre medlemmer i gruppen og også eventuell videreutvikling etter at Utstyrskontrollen har overtatt kodebasen.

8 Utrulling

Den endelige planen for programvaren vår er at den skal avleveres til Utstyrskontrollen i en tilstand hvor de kan bruke den. I denne seksjonen utdypes prosessen med hvordan vi satte opp programvare slik at applikasjonen kan ruller ut i framtiden. Selv om vi ikke nødvendigvis tror applikasjonen er klar for bruk i et produksjonsmiljø, mer om dette i kapittel 10, mener vi at det er viktig å ha en prosess for dette.

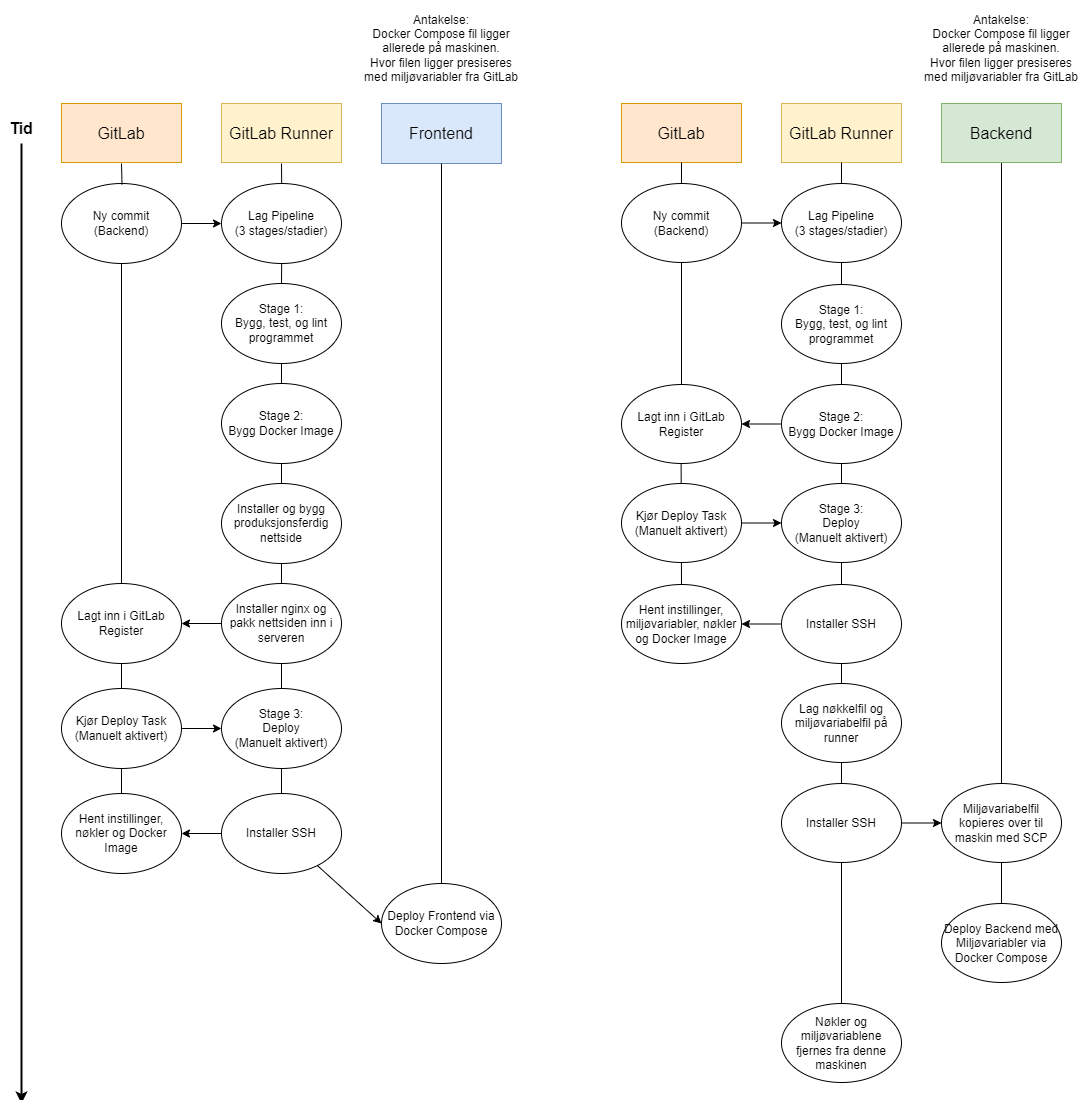
8.1 CI/CD

For å spare tid på å sette opp programvare og konfigurere programvare eller servere, er det praktisk å kunne automatisere slike oppgaver. Det å utvikle fortløpende, integrere programvaren, og rulle det ut fortløpende er noen av de viktigere kjennetegnene på DevOps [32]. I vårt prosjekt brukte vi ideen bak CI/CD, som står for continuous integration/continuous delivery: I vårt prosjekt brukte vi teknologi som eksisterer i GitLab for å praktisere filosofien av å skrive kode, automatisk bygge programmet, og rulle det ut så det kan testes.

Spesifikt brukte vi GitLab Pipelines for å automatisk bygge og teste programmet vårt. Den gjorde det på følgende måte:

1. Når ny kode ble sjekket inn i GitLab, ble en ny pipeline generert. En pipeline er en liste over hvilke oppgaver som må kjøres
2. Oppgavene fra pipelinen blir klagjort til å kjøres i en rekkefølge definert av oss, gjennom en konfigurasjonsfil
3. Når oppgaven er klar, blir den plukket opp av hvilken som helst maskin som ikke er opptatt med en annen oppgave
 - Disse maskinene kalles for [Runnere](#), og blir forklart i kapittel 8.2
4. Oppgaven kjøres, og alt fra loggføring til status blir rapportert i GitLab sin frontend

For å gå mer inn i detaljer på hvilke oppgaver som ble løst av vår pipeline, referer til sekvensdiagrammet (Figur 38), som viser denne prosessen.



Figur 38: Sekvensdiagram for pipelines til frontend (venstre) og backend (høyre)

Nytten av å bruke CI/CD er flerfoldig. Noen eksempler på disse gevinstene er at vi kan verifisere at programmet ikke knekker mens vi videreutvikler prosjektet. Samtidig med å bygge programvaren eksternt, unngår vi problemer i naturen av at noe kun fungerer på en enkelt utviklers maskin, men ikke på en annens. I dette tilfellet om det går gjennom i pipelinen, er det mye mer sikkert at om en utvikler gjør de samme stegene så vil programmet fungere.

8.2 GitLab runneren

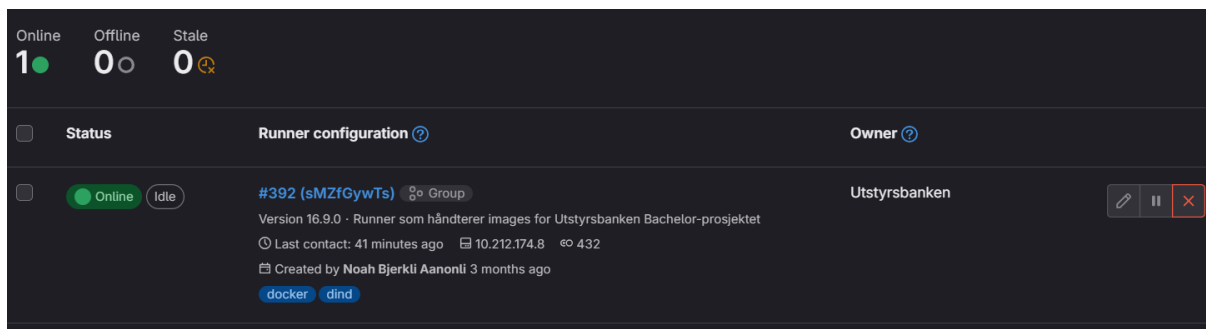
Som tidligere nevnt i kapittelet om CI/CD8.1 er en runner en maskin som er designet for å kjøre oppgaver. Siden vi bruker GitLab i vårt prosjekt blir den referert til som *GitLab runneren* for resten av kapittelet.

For prosjektet vårt ønsket vi å automatisere oppgavene våre med GitLab Pipelines. Raskt oppdaget vi et problem med runnerne når vi prøvde å bygge Docker Images med de: de var ikke konfigurert til å kunne kjøre Docker korrekt. Hver gang vi kjørte pipelinen fikk vi feilmeldingen "Cannot connect to the Docker daemon".

```
1 concurrent = 1
2 check_interval = 0
3 connection_max_age = "15m0s"
4 shutdown_timeout = 0
5
6 [session_server]
7   session_timeout = 1800
8
9 [[runners]]
10  name = "utstyrspartner"
11  url = "https://gitlab.stud.idi.ntnu.no"
12  id = 392
13  token = "*****"
14  token_obtained_at = 2024-02-16T15:10:19Z
15  token_expires_at = 0001-01-01T00:00:00Z
16  executor = "docker"
17  [runners.cache]
18    MaxUploadedArchiveSize = 0
19  [runners.docker]
20    tls_verify = false
21    image = "docker:25"
22    privileged = true
23    disable_entrypoint_overwrite = false
24    oom_kill_disable = false
25    disable_cache = false
26    volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
27    shm_size = 0
28    network_mtu = 0
```

Figur 39: Kopi av konfigurasjonsfilen for vår GitLab runner

Løsningen var å spinne opp vår egen GitLab Runner, med tillatelsene som trengtes for å kunne kjøre Docker i Docker. I dokumentasjonen kom det fram at Runneren måtte være privilegert for å ha lov til å gjøre operasjoner med Docker Daemon, som å bygge images.



Figur 40: GitLab runneren var klar til bruk

9 Diskusjon

Dette kapittelet handler om ulike temaer knyttet til selve prosjektprosessen som tidsbruk og gruppesamarbeidet samt om resultatet og eventuelle avvik fra planer.

9.1 Valg av dato

Sammenlignet med andre utlånssystemer som de ved biblioteker er det mye viktigere å velge dato når man skal låne fra Utstyrbanken. Dersom man for eksempel velger å reservere en bok vil man ved de fleste biblioteker i Norge kunne hente den snarest innen en viss tid mens ved Utstyrbanken må man velge en spesifikk dag. Grunnen til dette er fordi at Utstyrbanken sine åpningstider er svært begrensede og det påvirker mulighetene til henting og levering av utstyr. I tillegg vil man ved flere bibliotek kunne levere selvbetjent selv når biblioteket er stengt, den muligheten har ikke Utstyrbanken slik det er idag. Derfor er dette med datoer blitt så viktig for applikasjonens design.

9.2 Utlånssystem

Ved starten av prosjektet ble det vurdert et "handlekurvsystem for reserverasjoner. Det vil si at man kunne legge til flere *utstyrstyper* i en kurv" og deretter lage en reserverasjon for alt utstyr i kurven". Dette er et system de fleste kjenner igjen fra nettbutikker og det er også det systemet BUA bruker. Likevel fant vi raskt ut at det er utfordringer med implementasjonen av en slik løsning. Hovedutfordringen slik vi så det for oss er en situasjon der en person ønsker å låne et par ski og et par skistaver. Hen legger da disse i "handlekurven" og trykker reserver, men det viser seg at noen andre kom hen i forkjøpet og dermed blir reserverasjonen ikke godtatt siden det nå lenger ikke er nok utstyr på lager til at reserverasjonen blir godkjent.

For å unngå en slik situasjon vurderte vi å implementere *midlertidige reserverasjoner* som betyr at utstyret blir reservert i det man legger det til i kurven", da med en relativ kort tidsbegrensning før utstyret blir frigjort og tatt ut av kurven". Denne løsningen er veldig lik hvordan seterreserverasjon fungerer når man bestiller billetter til kino eller konserter. Med et slikt system unngås situasjonen der utstyrer i k-kurven" allerede har blitt reservert eller lånt ut, men det krever mye mer kommunikasjon mellom frontend og backend. For sanntidskommunikasjonen som er nødvendig kunne vi ha brukt websockets. Med websockets [12] kan man starte opp en toveis-forbindelse mellom klient og server der serveren kan fortelle klient blant annet at den midlertidige reserverasjonen har gått ut og oppdatere klient om antall tilgjengelig utstyr.

Dette systemet ville vært veldig komplisert å implementere og etter en diskusjon med oppdragsgiver ble vi enige om et enklere lånesystem. Et lån eller reserverasjon kan nå kun inneholde en størrelse av en utstyrstype, men kan bestå av flere av samme størrelse. Hele reserverasjonsprosessen skjer nå på utstyrssiden og reduserer sannsynligheten for at problemet beskrevet overfor inntreffer, men sannsynligheten er ikke null og videre utbedringer ville krevet sanntidskommunikasjon.

9.3 Overholding av personopplysninglover

Det er til slutt Utstyrbanken sitt ansvar å følge lover for personvern fra land de vil drifte i, men vi har gjort vurderinger og foreslått funksjonalitet for å gjøre det enklere for oppdragsgiver å overholde personopplysningsloven. Redigering av egen brukerdata som navn, fornavn og telefonnummer ble lagt til som funksjonelle krav i hendhold til personopplysningslov artikkel 16 om retten av å kunne rette personopplysninger, men ikke implementert i frontend.

Sletting av bruker er heller ikke implementert men var etter kravspesifikasjonen ment å bli gjort av ad-

administrator ved forespørsel av bruker slik at selv brukere med aktive lån kan bli slettet ved behov. Det å ikke inkludere en funksjon som lar brukere slette egen bruker bryter ikke nødvendigvis med personopplysningslov artikkel 17, retten om sletting så lenge det er "uten ugrunnet opphold", noe som kommer an på administratør. Det burde også være lett å kontakte administratør, og administratør burde kunne vite akkurat hvilken bruker som spør om å bli slettet

Det er et mulig problem med den delen av artikkel 17 som handler om at personopplysninger som ikke lenger trengs skal bli slettet. Når en person uten bruker skal låne fra utstyrsbanken og oppgir fornavn og etternavn til ansatt så er også dette personopplysninger. Når det gjelder brukere så gir det mening å ikke slette personopplysninger fordi en hver bruker trenger dem, men for gjestebrukere er det vanskelig å argumentere for at navnet og fornavnet trengs hvis ikke gjestebrukeren har aktive lån eller reserverasjoner pågående. Det å kunne velge en eksisterende gjestebruker gjør det lettere for den ansatte hvis personen uten bruker skal gjøre enda et lån, men om det er grunn nokk til å oppbevare opplysningene uendelig så lenge låner ikke spesifikt spør om sletting av noe de kanskje ikke en gang vet er lagret kan bli vrient å argumentere for.

9.4 Avvik fra framdriftsplan

- Missed MVP deadline. fre 22. Mars
- kommunikasjon vansker med oppdragsgiver tok mye tid
- (Mini problem) SkyHigh var nede under de først møtene (vedlikehold, kilde?)
- Flere prosjekter gående samtidig

Vi greidde ikke å fullføre MVP innen den interne fristen vi hadde satt som førte til at vi ikke fikk gjennomført brukertestene slik vi hadde planlagt. I utgangspunktet hadde vi planlagt å bruke påskeferien til dette da vi kunne teste applikasjonen med venner og familie mens vi var hjemme i ferien. I det frontend ble klar måtte vi dessuten skifte fokus til å begynne på rapportskrivning.

En annen liten kilde til forsinkelser var problemer med *SkyHigh*-plattformen på grunn av nedetid. Vi opplevde også nedetid med *container repository*-delen av NTNU sin GitLab-instans.

Kommunikasjons vansker med oppdragsgiver har ført til mye fram og tilbake I hvertfall når det kommer til krav spesifikasjoner, kommunikasjon gikk som oftest gjennom wire-frames som tokk mye tid å sette opp samtidig som at det er mange kravspesifikasjoner som var for vanskelige å få kommunisert fra oppdragsgiver og mange problemstillinger som vi slet med å kommunisere tilbake SkyHigh ble migrert under starten av prosjektet som førte til forsinkelser med utrulling og testing, samt at vi har hatt problemer med utrulling før møter med oppdragsgiver som førte til bortkastet tid og mye stress med å fikse det under møtene.

9.4.1 Rutiner

I ettertid ser vi at vi burde ha hatt bedre rutiner når vi planla ulike aspekter av programvaren, blant annet gjelder dette hvilke **endepunkter** som trengtes samt hvilken funksjonalitet de skal ha. Et eksempel er funksjonaliteten for å legge til og endre på størrelser til en utstyrstype. I backend er det kun mulig å legge til, redigere og slette en størrelse av gangen, mens i frontend legges det opp til at dette burde gjøres i en operasjon. En årsak til at det ble slik var at **endepunktene** i backend ble designet og utviklet før vi visste hvordan dette kom til å fungere i frontend.

9.4.2 Kutt av funksjonalitet

På grunn av tidspress måtte vi kutte noen av funksjonaliteten nevnt i kravspesifikasjon (se kapittel 2). Denne funksjonaliteten, etter diskusjon med oppdragsgiver, blitt regnet som mindre viktig og derfor blitt nedprioritert. Det vil da være opp til de som eventuelt tar over utviklingen etter oss å implementere denne funksjonaliteten.

Den ene funksjonaliteten er et notifikasjonssystem som skulle for eksempel kunne sende ut purre-SMSer dersom en låner ikke leverer tilbake lånet sitt i tide eller sende påminnelser om at utstyret man har reservert kan hentes imorgen gjerne også med dagens åpningstider. Implementasjonen av dette kunne blitt gjort via en *goroutine* som periodisk sjekker om det skal sendes noen SMSer for denne typen informasjon. For å kunne sende purre-SMSer vil det også ha krevd å lagre når man sendte en SMS sist til den brukeren slik at man kan ha et fast intervall. En annen måte ville vært å bruke et hendelsesbasert system som automatisk planla og kansellerte SMSene etter behov.

En annen funksjonalitet som ble kuttet er hendelsesloggen. En slik logg ville måtte implementeres i store deler av backend, da det var aktuelt å logge de fleste handlinger som ansatte og administratorer skal kunne gjennomføre. Det var også utfordrende å definere strukturen til loggen i databasen og hvordan den skal vises til administrator i frontend. I tillegg var dette noe gruppen kom med selv og noe oppdragsgiver var mindre interessert i.

9.5 Videre utvikling

9.5.1 Av frontend

Frontend vil trenge mer finpuss og fiksing av diverse småfeil som eksisterer for øyeblikket. Blant annet bør brukeren få mer tilbakemelding på handlingene sine, både når de er suksessfulle og når noe går galt. Det vil bidra til å styrke brukeropplevelsen.

På grunn av at utviklingen av frontend henger noe etter backend må en del av funksjonaliteten som eksisterer i backend implementeres i frontend. Blant annet gjelder dette henting og opplasting av bilder, utviding av egne lån og redigering av egen brukerdata.

Kildekoden er også rotete, og trenger bedre organisering. Det kan løses gjennom å sortere funksjonene i pakker. Koden bør også ha tester, for å forsikre seg at funksjonaliteten ikke knekker i framtiden ved refaktoreringer.

9.5.2 Av backend

Det meste av funksjonalitet som kreves for å kunne overholde de funksjonelle kravene satt i kapittel 2, med unntak av kuttene nevnt over, ble implementert i backend i tide for innlevering.

Ved slutten av utviklingsperioden kom det likevel frem et behov for å ha muligheten til å gjøre flere operasjoner av samme type med en forespørsel, altså en form for batch-*endepunkt*. Et eksempel er å kunne lage flere størrelser til en utstyrstype med en forespørsel. Dette ville redusert antall forespørsler som frontend gjør mot backend som gjør. Om vi hadde hatt mer tid har dette vært en logisk videreutvikling av *endepunktene* der det er behov for dette.

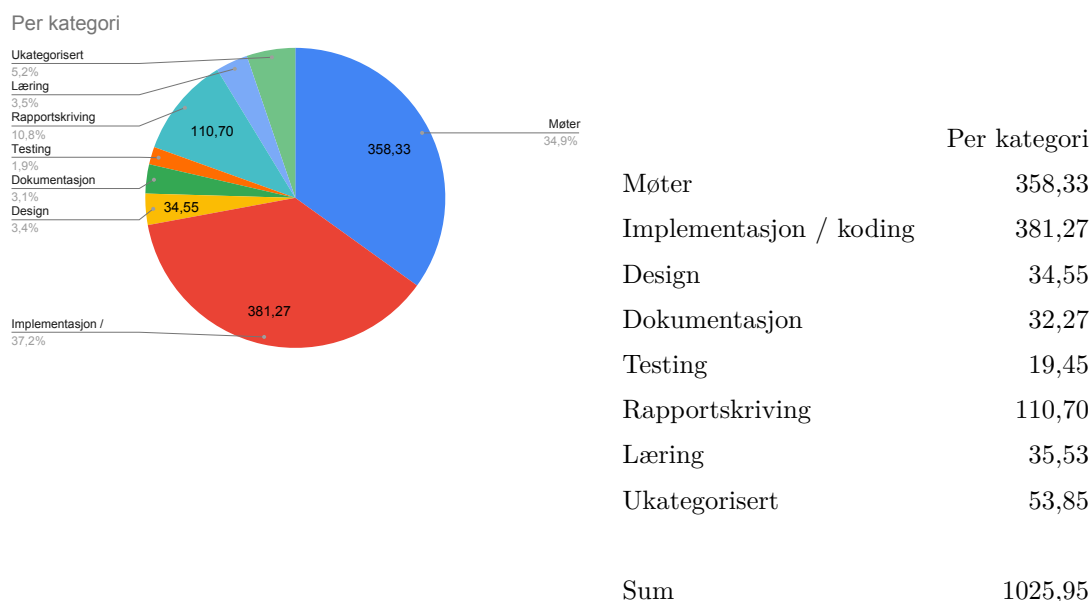
Noe annet som bør sees nærmere på er om man skal innføre krav til passord altså minimumlengde, krav til inklusjon av store og små bokstaver, tall og symboler og så videre. Slik verifisering må implementeres også i backend.

Feilhåndtering er et område der det er rom for forbedring i backend, spesielt gjelder dette feilmeldinger som resultat av database-funksjoner. Det blir mye generiske feilmeldinger av typen *500 Internal Server Error*. Noe av dette stammer riktignok fra at det ikke alltid er så lett å håndtere feil i database-operasjoner. Et godt eksempel er UPDATE-spørringer (oppdatering av eksisterende rader), den eneste informasjonen vi får fra resultatet av en slik spørring (dersom det ikke er en større feil som at databasen er nede) er hvor mange rader i tabellen som er påvirket av spørringen. Det vil si at vi får 0 rader oppdatert om vi prøver å oppdatere en rad med samme verdier som den allerede har og om raden som forsøkes oppdatert ikke finnes. Dette kan gjøre det vanskelig å returnere korrekt informasjon om hva som er galt ved modifisering av ressurser.

Det hadde også vært nyttig for utvikling og testing at hvilken pakke samt funksjon der feilen skjedde ble logget til konsollen slik at man lettere kunne lokalisere eventuelle bugs i koden. Dette vil riktignok kreve en ganske omfattende omskriving av koden, noe som ikke ble prioritert over å utvikle mer funksjonalitet.

En opprydning der man jobber for å deduplisere kode bør også gjøres. Det er en del duplikat kode blant annet bør man ytterligere utvide *error_handling*-pakken slik at man får samlet alle de ulike typene feilmeldinger. Samt er det ønskelig å øke bruken av konstantverdier for verdier som gjenbrukes i koden. Dette vil gjøre det enklere å vedlikeholde koden i fremtiden.

9.6 Tidsbruk



Figur 41: Tidsbruk fordelt på de ulike kategoriene, hentet fra regnearket kl. 14 16/05

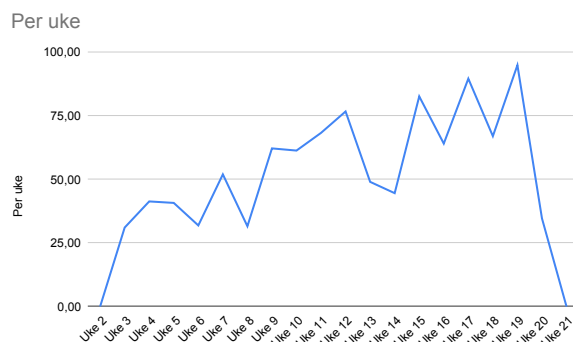
I figur 41 ser vi tidsbruken fordelt på kategori og her ser man at særlig to kategorier skiller seg ut. Tid brukt på møter og tid brukt på selve implementasjonen av applikasjonen. Noe av grunnen til at møter er blant det vi har brukt mest tid på er at vi har hatt mye møter, men også hvordan enkelte gruppemedlemmer valgte å kategorisere øktene sine. Med andre ord vil dette bety at faktisk fordeling mellom de ulike kategoriene er noe annerledes i forhold til det statistikken viser. Hvordan øktene skulle kategoriseres burde nok ha blitt bestemt mer nøyaktig for slik det er nå betyr det at fordelingen mellom de ulike kategoriene ikke er helt reelle.

Kategorifordelingen viser også at det er aspekter vi burde ha brukt mer tid på som testing, design og rapportskriving, rapporten er tross alt den viktigste komponenten når det kommer til vurdering av

bacheloroppgaven. En grunn til at mye av tiden gikk til implementasjonen var at det var overraskende mye å gjøre på denne oppgaven.

Om man ser på tidsbruken samlet sett ser vi at vi har brukt vesentlig mindre tid enn det som er anbefalt for bacheloroppgaven. Fra universitetet anbefales det 30 timer i uka for hver student, det var også dette vi satte som mål i prosjektplan. Det vil si at vi skulle ha hatt omtrent 2100 timer sammenlagt. Det er dog grunn til å tro at målt tidsbruk avviker noe fra faktisk tidsbruk blant annet mangler det data fra en av gruppemedlemmene, målingen av tidsbruk er en veldig manuell prosess og ble derfor av og til glemt. Men, antallet timer som ikke ble registrert vil trolig ikke være i nærheten av de 1000 timer som "mangler". Tre av gruppemedlemmene har riktignok hatt også hatt andre fag som har tatt sin tid i løpet av semesteret, det er mulig at vi har brukt mer tid på de fagene enn det som er anbefalt og dermed delvis er grunnen til mindre tidsbruk på bachelorprosjektet.

En annen grunn kan være at i starten av prosjektet så tok vi ikke på oss nok arbeidsoppgaver under Sprint Planning og brukte derfor for liten tid på prosjektet. Bedre planlegging under Sprint Planning kunne derfor føre til at vi nådde målene våre for tidsbruk og et bedre resultat. Det gikk også en del tid bort når vi skulle holde møtene på reising til og fra campus og venting på at alle gruppemedlemmene møter opp til møtet. Denne tiden er ikke faktorert inn da det ikke direkte førte til framgang på prosjektet.



Figur 42: Tidsbruk fordelt på ukene, hentet fra regnearket kl. 14 16/05. **Merk:** Det mangler data for uke 2 og uke 21

Tidsbruken fordelt ukevis (se figur 42) viser at vi har brukt mer og mer tid på prosjektet etterhvert som tidsfristen nærmet seg. Dette skyldes nok at etterhvert som tidsfristen nærmet seg innså vi at vi måtte bruke mer tid for å rekke det vi skulle. I tillegg ble vi etterhvert ferdig med de andre fagene som frigjorde mer tid til bacheloroppgaven.

Mer detaljert og oppdatert data om tidsbruken kan sees i vedlegg F.

9.7 Gruppesarbeid

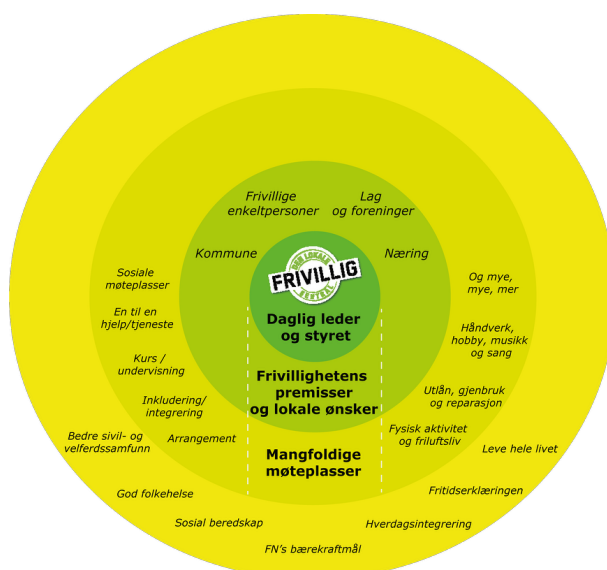
Gruppesarbeidet gikk i hovedsak ganske bra, men i ettertid ser man behovet for småmøter for å bedre kunne oppdatere hverandre. Standupmøtene som ble planlagt som skulle bli brukt til dette formålet, ble sjelden avholdt på grunn av manglende oppmøte. Et annet grep vi kunne tatt for å bedre samarbeidet rundt selve koden er å ta i bruk parprogrammeringsteknikken [44] som vil si at man jobber to sammen om å utvikle en funksjonalitet. Måten det gjøres på er at en koder mens den andre gir tilbakemeldinger og deretter bytter man på. Dette kunne økt kodekvaliteten og ført til at mindre tid ble brukt på å rette opp dårlig kode.

Noe annet som påvirket gruppesarbeidet var at enkelte gruppemedlemmer kom for sent (med sent menes 20 minutter eller mer etter fastsatt tid) til flere av møtene vi har satt opp, særlig gjaldt det møtene som ikke involverte oppdragsgiver eller veileder. Selvsagt kan man bli forsinket en gang i blant på

grunn av forsovelse eller lignende, men det skjedde litt for ofte. Det førte til litt for mye tid som ble kastet bort på å vente på resten av gruppelemmene. Her burde vi nok ha vurdert å innført noen konsekvenser for manglende oppmøte. Når det gjelder hvilke konsekvenser det burde vært så er det usikkert, men det burde være noe som er nok til at man møter opp. Selvsagt måtte vi ha hatt regler som gav gyldig fravær i enkelte tilfeller der fraværet skyldtes sykdom, begravelser, toget er forsinket eller lignende som er utenfor den enkeltes kontroll.

Vi burde også ha hatt mer rutine på møtereferatene. Blant annet en fast mal så vi får inn detaljer som dato og tid for møte, om møtet er fysisk eller digitalt og hvem som møtte opp. Det ville gjort det lettere å finne frem i referatene og det kunne dermed også ha fungert som en måte å dokumentere fravær.

9.8 Bærekraft



Figur 43: De fundamentale byggesteinene av Norges Frivilligsentraler

FN sine bærekraftsmål står sentralt for både frivilligsentralene i Norges frivillighetssentral (NFS) og Utstyrskontrollen, som kan sees på figuren ovenfor. Særlig gjelder dette sosial bærekraft, men også miljømessig bærekraft. Om vi ser på noen spesifikke mål:

- "Mål 4: God utdanning" og "Mål 11 Mindre ulikhet": med å gi alle elever lik tilgang på utstyr uansett økonomisk situasjon
- 11 Bærekraftige byer og lokalmiljøer + 12 Ansvarlig forbruk og Produksjon + 13 Stoppe klimaendringene: med å styrke samfølelse innad i lokalmiljø med donasjoner og frivillighet arbeid samt gjenbruk av utstyr

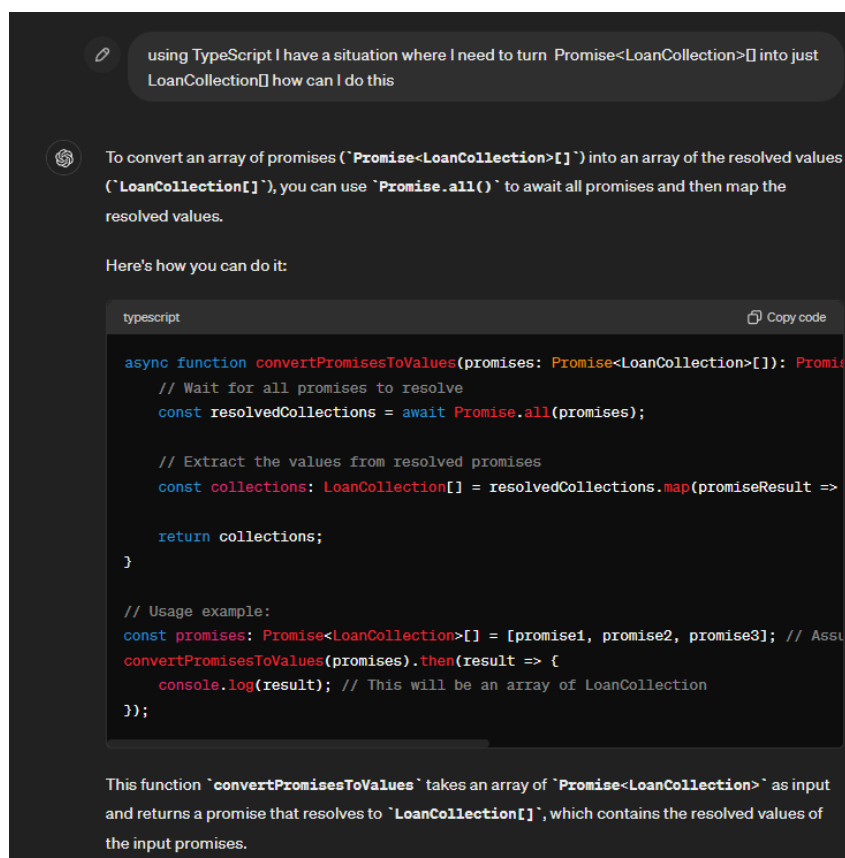
Ustyrskontrollen tilbyr sine tjenester gratis, støttet av frivillighet fra ansatte og donasjoner fra lokalmiljøet. Dette gir alle i lokalmiljøet tilgang på utstyr for å delta på ulike aktiviteter uavhengig av egen økonomi. I tillegg er skolene i nærheten av Ustyrskontrollen blant Ustyrskontrollens største brukere og Ustyrskontrollen hjelper til med å la alle elever delta i aktiviteter arrangert av skolen uansett hvordan økonomien i husstanden er. Slik sett bidrar Ustyrskontrollen til mye sosial bærekraft.

Siden nesten alt utstyret er donert fra lokalmiljøet inspirerer det til gjenbruk av utstyr som potensielt har mange år igjen istedetfor at det står ubrukt eller kastes, dette minsker forsøpling som er en stor del av målene til FN.

Ved å utvikle denne programvaren bidrar vi også, både med å erstatte den eksisterende papirbaserte skjema-løsningen, men mest av alt ved å gjøre Utstyrskontorens tjenester mere tilgjengelige for brukere. Når Utstyrskontoret så til å modernisere papirløsningen deres så ble det vurdert å bli med i BUA-nettverket. BUA har allerede en digital løsning, men BUA stiller en rekke krav som går imot Utstyrskontoret sine verdier som å være drevet av lokal miljøet, av donasjoner og frivillighet. Blant annet krever BUA at alt utstyr skal kjøpes inn nytt og at lokalene skal pusses opp og matche resten av BUA-lokalene rundt om i landet. Ikke bare ville dette antagelig være for kostbart for Utstyrskontoret, men kravet om nytt utstyr er også til dels mindre bærekraftig sammenlignet med muligheten til å ta imot og kjøpe inn brukt utstyr. Dermed måtte Utstyrskontoret ordne seg en egen løsning i stedet.

9.9 KI og oppgaven

KI ble brukt i prosjektet for å hjelpe til å utvikle deler av frontend, spesifikt har vi brukt ChatGPT versjon 3.5, til varierende grad av suksess. Hvor enkelte ganger så ga ChatGPT innsikt i funksjoner og biblioteker som var perfekt for jobben, mens andre ganger så resulterte løsningene til ikke mere enn bortkastet tid på løsninger som aldri ville fungert. Et par eksempler på hvordan vi har brukt KI og hvor nyttig det har vært:



```
using TypeScript I have a situation where I need to turn Promise<LoanCollection>[] into just LoanCollection[] how can I do this

To convert an array of promises ("Promise<LoanCollection>[]") into an array of the resolved values ("LoanCollection[]"), you can use "Promise.all()" to await all promises and then map the resolved values.

Here's how you can do it:

typescript Copy code

async function convertPromisesToValues(promises: Promise<LoanCollection>[]): Promise<LoanCollection[]> {
  // Wait for all promises to resolve
  const resolvedCollections = await Promise.all(promises);

  // Extract the values from resolved promises
  const collections: LoanCollection[] = resolvedCollections.map(promiseResult => {
    // ... (omitted) ...
  });

  return collections;
}

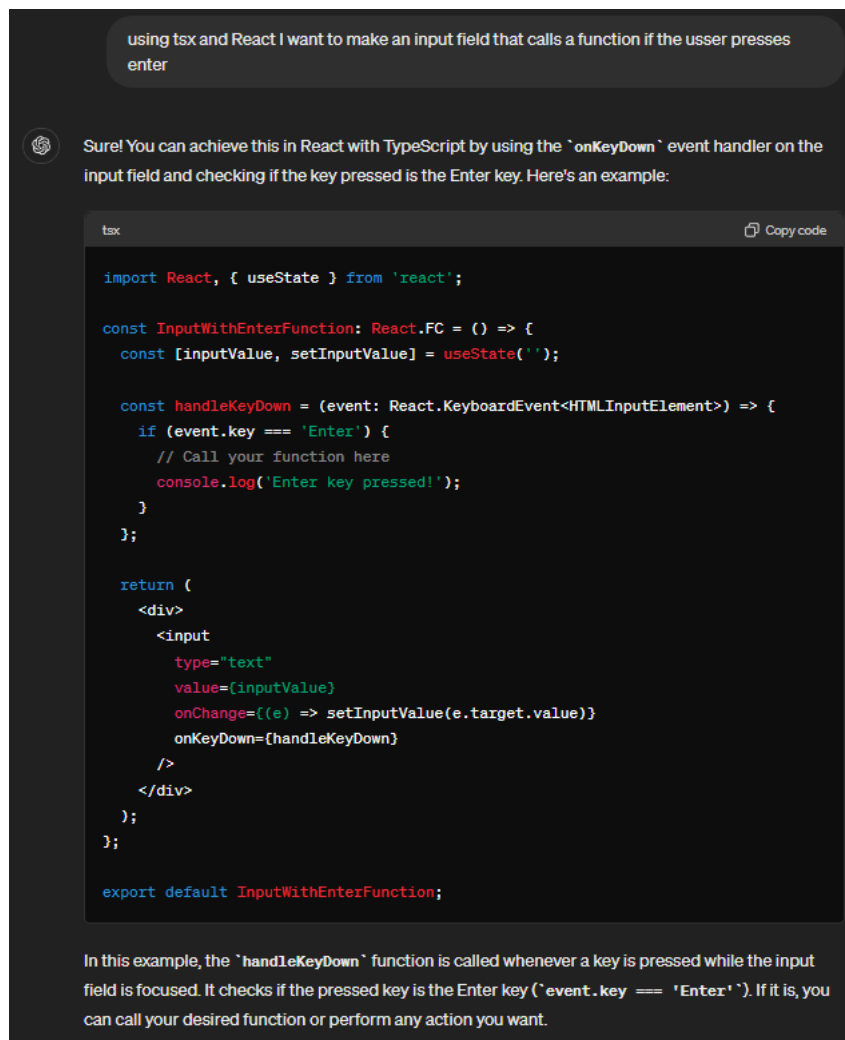
// Usage example:
const promises: Promise<LoanCollection>[] = [promise1, promise2, promise3]; // Assume promises are defined elsewhere
convertPromisesToValues(promises).then(result => {
  console.log(result); // This will be an array of LoanCollection
});

This function "convertPromisesToValues" takes an array of "Promise<LoanCollection>" as input and returns a promise that resolves to "LoanCollection[]", which contains the resolved values of the input promises.
```

Figur 44: ChatGPT 3.5 eksempelkode om Promise.all

Bakgrunnen til eksemplet i figur 44 er at vi sendte en forespørsel asynkront til backend hvor vi spør om å hente alle lån, se implementasjon frontend. Dette førte til at siden ikke ville vente på svar men heller vise bruker en tom side. Dette fikk oss til å spørre ChatGPT om den hadde en løsning til å få nettsiden til å vente på forespørselen. ChatGPT kom tilbake med svaret som er ovenfor hvor den demonstrerer en liten del av Promise funksjoner. Koden ChatGPT demonstrerer endte ikke opp i koden vår, men hintet oss til Promise.all funksjonen som vi så nærmere på. ChatGPT sin bruk av Promise.all er til og med feil,

siden den skal egentlig brukes når mere en en forespørsel blir sendt om gangen. Til tross for at eksempelet ChatGPT ga oss ikke var optimalt, brukte den en funksjon som ingen i gruppen var kjent med fra før, som viste seg uvurderlig for resten av prosjektet og som vi ønsket at vi viste om fra starten av når vi skrev koden for frontend.



```
using tsx and React I want to make an input field that calls a function if the user presses enter

Sure! You can achieve this in React with TypeScript by using the `onKeyDown` event handler on the input field and checking if the key pressed is the Enter key. Here's an example:

tsx Copy code

import React, { useState } from 'react';

const InputWithEnterFunction: React.FC = () => {
  const [inputValue, setInputValue] = useState('');

  const handleKeyDown = (event: React.KeyboardEvent<HTMLInputElement>) => {
    if (event.key === 'Enter') {
      // Call your function here
      console.log('Enter key pressed!');
    }
  };

  return (
    <div>
      <input
        type="text"
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)}
        onKeyDown={handleKeyDown}
      />
    </div>
  );
};

export default InputWithEnterFunction;

In this example, the `handleKeyDown` function is called whenever a key is pressed while the input field is focused. It checks if the pressed key is the Enter key (`event.key === 'Enter'`). If it is, you can call your desired function or perform any action you want.
```

Figur 45: ChatGPT 3.5 overkompliserer en simpel løsning, der man vil lese inndata fra bruker

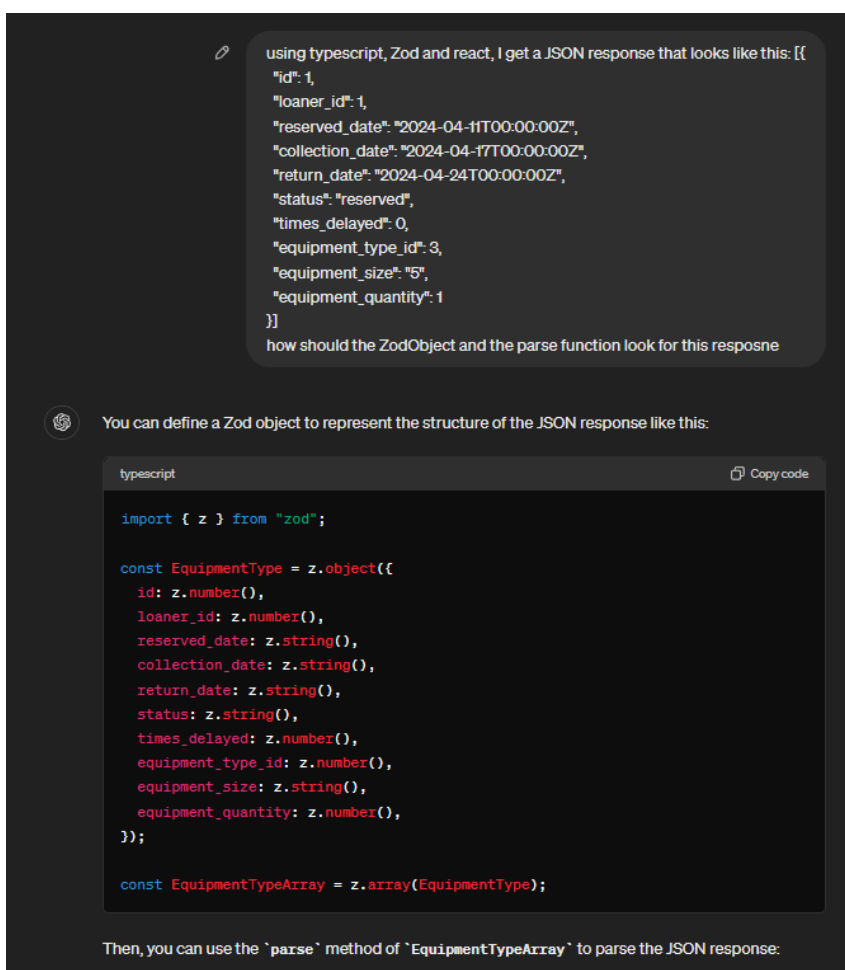
Vi har ikke alltid vært like heldig med KI, ChatGPT sin respons i figur 45 er et eksempel. Her har ChatGPT overkomplisert løsningen på grunn av måten forespørselen er formulert. Først og fremst fungerer ikke koden, i hvertfall ikke innenfor miljøet vi hadde, men den har fokusert for mye på feil deler av forespørselen. ChatGPT har fokusert alt for mye på den første setningen som er "using tsx and React". Med å starte med dette gir vi kontekst til ChatGPT om hvordan utviklings miljøet til frontend ser ut, ChatGPT har antagelig tolket dette som at løsningen må inneholde kode som må være en del av React-språket, så den brukte React sitt input system ment for mere interaktive sider som webspill med spillkontroller-støtte for web-apps. ChatGPT endte opp med å overkomplisere koden. Mens funksjonaliteten som vi ønskes kan løses med funksjonalitet fra HTML og TypeScript. Vi endte opp med å forsøke å få denne koden til å funke og kom med en løsning som er tungvinn og overkomplisert. Senere i utviklingen brukte vi HTML forms, som gir en mye simplere løsning

Dette er en hypotetisk løsning som bruker forms og er mye simplere, funksjonaliteten er standard i vanlig HTML kode, ChatGPT hadde oversett dette for å heller skape en situasjon som måtte bruke React funksjoner. Dette er en problemstilling som kunne ha blitt fikset med et bedre formulert spørsmål


```
<form onSubmit={this.loadLoans(LoanCall.Phone)}>
  <input
    type="tel"
    inputMode={"tel"}
    value={this.state.phoneSearch}
    onChange={this.handlePhoneChange}/>
</form>
```

Figur 46: Eksempel på en lettere løsning som leser inn telefonnummer og søker når man trykker enter

til ChatGPT eller med en annet KI-verktøy som GitHub sin Co-Pilot da denne har tilgang til hele prosjektets kildekode for å lage beslutninger. Oppsummert viser dette at KI som ChatGPT ofte kan komme med overkompliserte løsninger, hvis man ikke er konsekvent og vet hva man ser etter.



Figur 47: KI for repetitive oppgaver

Dette er hvor Zod (se kapittel 6.1.2) kommer inn i bildet. Hvis du allerede vet ganske konkret hva du vil ha så kan KI være hjelpsomt for simple men repetitive oppgaver, dette siste eksempelet har vi en eksempel JSON respons fra backend, men for å få det inn i frontend trenger vi et Zod objekt som matcher datastrukturen, med både navn og type. Dette er en oppgave som krever at Zod objektet og JSON responsen har presist like variabel navn. Isteden for å fylle ut Zod objekt for en hver type respons fra databasen, valgte vi heller å optimalisere tiden vår med å bruke de dokumenterte eksempelresponsene og gi dem til ChatGPT som du kan se ovenfor, ChatGPT ga tilbake eksempel kode som ikke var komplett

fordi vi hadde en annen datastruktur for å lagre dato en strings (tekst), som ChatGPT foreslo, men siden vi allerede viste om det så var det lett å bytte det ut. ChatGPT viste seg veldig nyttig på dette feltet og ga de riktige variabel navnene slik at JSON fra backend ble dekodet på riktig måte. På denne måte sparte vi tid på en repetitiv oppgave, som vi allerede viste svaret på, men som tar tid å fylle ut.

9.9.1 KI og framtiden

Det er vanskelig å spå framtiden, men basert på det vi ser idag er det grunn til å tro at dersom et tilsvarende prosjekt i framtiden ville brukt mer KI under utviklingen av applikasjonen [52]. Det finnes KI-verktøy som idag kan generere kode, men koden som genereres er langt fra feilfri. Kode skrevet av KI bør nok alltid kvalitetssikres av gruppemedlemmene, men med fremtidens KI-verktøy vil man antagelig bruke mindre tid på å rette opp i feilene som gjør KI-verktøyet mer verdt å bruke. Man kan også tenke seg at vi får KI-verktøy som kan lete og finne bugs og sårbarheter i koden samt forslag til hvordan koden kan optimaliseres. Generelt vil KI, med mindre utviklingen virkelig skyter fart de neste årene, fungere som en assistent og vil ikke erstatte de menneskelige utviklerene. Derimot kan man se for seg at man får høyere effektivitet og får derfor et mer ferdig produkt med samme tidsbruk.

En annen mulighet er et KI-verktøy som kunne generert styling og basiskode for en side for frontend basert på en wireframe eller lignende diagrammer. Deretter kunne man fylle inn med logikken som kreves, dette ville vært til god hjelp og kunne fått opp farten på utviklingen av frontend.

I selve produktet har KI liten relevans. Det er rett og slett lite behov i vår applikasjon. Man trenger ikke KI for å kunne låne et par ski og noen skistaver. Det som kunne blitt vurdert er en algoritme som anbefalte utstyr som det er sannsynlig at personen vil låne eller en algoritme som sorterer utstyret basert på hva som folk sannsynligvis vil låne basert på faktorer som tidspunkt og været eller lignende. Begge ville krevet utvikling av en egen modell og ville antagelig tatt unødvendig mye utviklingstid i forhold til hvor viktig slik funksjonalitet ville vært for tjenesten. Selv noen år inn i framtiden er det usannsynlig at dette vil endre seg.

10 Avslutning

I dette siste kapittelet gir vi vår egen vurdering av resultatet (inkluderer både programvareløsningen og rapporten) samt en konklusjon av rapporten.

10.1 Egen vurdering av resultat

Til tross for litt for lav tidsbruk, som nevnt i kapittel 9, så vurderer vi det slik at vi har fått gjort mye med den tiden vi brukte. Vi er stort sett fornøyd med deler av den utviklede programvaren og fikk implementert de viktigste funksjonalitetene som ønskes. Backend har det meste som trengs bortsett fra [batch-endepunkter](#) som vi fant ut at trengtes underveis og fungerer stort sett svært godt. Frontend har mangler og noen bugs, men kjernefunksjonalitet er implementert og med litt finpuss virker brukergrensesnittet til å være intuitivt og enkelt å forstå og bruke. En årsak til at frontend er som det er, er antagelig på grunn av gruppens generelle mangel på erfaring med moderne frontendutvikling med rammeverk som React og koden bærer dessverre preg av dette. Det betyr dessverre at applikasjonen ikke er klar til bruk og at Utstyrsbanken må finne noen andre som kan videreutvikle applikasjonen.

Det skal sies at på grunn av tidsbruken kan man lure på hvordan resultatet ville vært dersom tidsbruken vår var på nivået som ble anbefalt, ville vi da hatt en applikasjon som faktisk var ferdig og kunne bli tatt i bruk av Utstyrsbanken. Det er rimelig å tro det ihvertfall, med tanke på at man ville hatt dobbelt så myetid og med den tiden ville vi nok greidd å lagd en mye mer ferdig frontend.

Selve rapporten ble dessverre i alt for lang tid nedprioritert blant annet på grunn av behovet for mer utvikling av frontend slik at vi noenlunde ble ferdig med den. Dermed ble tiden for å skrive og spesielt finpusse rapporten nokså knapp. Likevel føler vi at rapporten dekker oppgaven godt og går igjennom sentrale temaer knyttet til designet og implementasjonen av programvaren som ble utviklet.

10.2 Konklusjon

For å konkludere kan vi si at vi har lært mye fra å jobbe på dette prosjektet. Derfor kan vi si at vi har oppfylt de fleste av læringsmålene vi har satt for prosjektet. Vi fikk blant annet lært mer om [fullstack-utvikling](#) og hvordan man samarbeider med en ekstern oppdragsgiver om en slik oppgave. Det vi derimot ikke fikk lært om var brukertester, da vi simpelthen ikke rakk å gjennomføre de. Infrastrukturen til programvaren er stort sett solid, men krever større endringer i presentasjonslaget for å være klar til bruk. Det som kan konkluderes med er at hovedårsaken til mangler i hele prosjektet er på grunn av dårlig tidsstyring innad i gruppen. Vi har brukt altfor liten tid som har ført til at vi ikke rakk å utvikle en programvare som tilfredsstillte alle kravene i kravspesifikasjon og også flere av resultatmålene. Siden programvaren trenger videre utvikling vil vi heller ikke kunne se effektmålene i praksis før den faktisk er videre utviklet til det stadiet at den kan settes i bruk.

Referanser

- [1] 27.4.1 Event Scheduler Overview. URL: <https://dev.mysql.com/doc/refman/8.0/en/events-overview.html> (besøkt den 01.05.2024).
- [2] BUA. BUAs offisielle nettside. URL: <https://www.bua.no/> (besøkt den 12.05.2024).
- [3] Cloudflare. What is a brute force attack? URL: <https://www.cloudflare.com/en-gb/learning/bots/brute-force-attack/> (besøkt den 29.04.2024).
- [4] World Wide Web Consortium. WCAG 2 Overview. URL: <https://www.w3.org/WAI/standards-guidelines/wcag/> (besøkt den 08.05.2024).
- [5] Datatilsynet. Rett til sletting. URL: <https://www.datatilsynet.no/rettigheter-og-plikter/den-registrertes-rettigheter/rett-til-sletting/> (besøkt den 13.05.2024).
- [6] Datatilsynet. Skytjenester. 2018. URL: <https://www.datatilsynet.no/personvern-pa-ulike-omrader/internett-og-apper/skytjenester/> (besøkt den 17.05.2024).
- [7] MDN Web Docs. Attributes - Set-Cookie. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie#attributes> (besøkt den 03.05.2024).
- [8] MDN Web Docs. Cross-site scripting (XSS). URL: https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting (besøkt den 03.05.2024).
- [9] MDN Web Docs. CSRF. URL: <https://developer.mozilla.org/en-US/docs/Glossary/CSRF> (besøkt den 03.05.2024).
- [10] MDN Web Docs. HTTP request methods. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods> (besøkt den 01.05.2024).
- [11] MDN Web Docs. Idempotent. URL: <https://developer.mozilla.org/en-US/docs/Glossary/Idempotent> (besøkt den 07.05.2024).
- [12] MDN Web Docs. The WebSocket API (WebSockets). URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (besøkt den 14.05.2024).
- [13] MDN Web Docs. Using HTTP cookies. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies> (besøkt den 15.05.2024).
- [14] R. Fielding, M. Nottingham og J. Reschke. RFC 9110: HTTP Semantics. 2022. URL: <https://www.rfc-editor.org/rfc/rfc9110.html> (besøkt den 02.05.2024).
- [15] Internet Engineering Task Force. About RFCs. URL: <https://www.ietf.org/process/rfc/> (besøkt den 02.05.2024).
- [16] Norges Frivilligsentraler. Om frivilligsentraler. URL: <https://frivilligsentral.no/dokument?om-frivilligsentraler&Id=77456> (besøkt den 17.05.2024).
- [17] GeeksforGeeks. Disk Defragmentation in Operating System. 2024. URL: <https://www.geeksforgeeks.org/disk-defragmentation-in-operating-system/> (besøkt den 20.05.2024).
- [18] GeeksforGeeks. Token Bucket Algorithm. URL: <https://www.geeksforgeeks.org/token-bucket-algorithm/> (besøkt den 29.04.2024).
- [19] Andrew Gerrand. Error handling and Go. 2011. URL: <https://go.dev/blog/error-handling-and-go> (besøkt den 26.04.2024).
- [20] GitHub. About commits. 2024. URL: <https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits> (besøkt den 18.05.2024).
- [21] GitHub. About self-hosted runners. URL: <https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/about-self-hosted-runners> (besøkt den 20.05.2024).
- [22] GitLab. GitLab Runner. URL: <https://docs.gitlab.com/runner/> (besøkt den 20.05.2024).

- [23] GitLab. *Merge requests*. 2024. URL: https://docs.gitlab.com/ee/user/project/merge_requests/ (besøkt den 18.05.2024).
- [24] Google. *rate (Go-pakke)*. URL: <https://pkg.go.dev/golang.org/x/time/rate> (besøkt den 29.04.2024).
- [25] Jim Gray. *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem*. Tekn. rapp. MSR-TR-2006-45. Apr. 2006, s. 10. URL: <https://www.microsoft.com/en-us/research/publication/to-blob-or-not-to-blob-large-object-storage-in-a-database-or-a-filesystem/>.
- [26] IBM. *What is a REST API?* URL: <https://www.ibm.com/topics/rest-api> (besøkt den 07.05.2024).
- [27] IBM. *What is three-tier architecture?* URL: <https://www.ibm.com/topics/three-tier-architecture> (besøkt den 08.05.2024).
- [28] Tilsynet for universell utforming av ikt. *WCAG-standarden*. URL: <https://www.uutilsynet.no/wcag-standarden/wcag-standarden/86> (besøkt den 08.05.2024).
- [29] MinIO Inc. *MinIO's hjemmeside*. URL: <https://min.io/> (besøkt den 01.05.2024).
- [30] MongoDB Inc. *What Is Full Stack Development?* 2024. URL: <https://www.mongodb.com/resources/basics/full-stack-development> (besøkt den 16.05.2024).
- [31] *Introducing JSON*. URL: <https://www.json.org/json-en.html> (besøkt den 26.04.2024).
- [32] Ramtin Jabbari mfl. «What is DevOps? A Systematic Mapping Study on Definitions and Practices». I: *Proceedings of the Scientific Workshop Proceedings of XP2016*. XP '16 Workshops. Edinburgh, Scotland, UK: Association for Computing Machinery, 2016. ISBN: 9781450341349. DOI: 10.1145/2962695.2962707. URL: <https://doi.org/10.1145/2962695.2962707>.
- [33] Javatpoint. *Shallow Copy and Deep Copy in JavaScript*. URL: <https://www.javatpoint.com/shallow-copy-and-deep-copy-in-javascript> (besøkt den 19.05.2024).
- [34] M. Jones mfl. *RFC 7519: JSON Web Token (JWT)*. 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519> (besøkt den 28.04.2024).
- [35] S. Josefsson og SJD. *RFC 4647: The Base16, Base32, and Base64 Data Encodings*. 2006. URL: <https://datatracker.ietf.org/doc/html/rfc4648> (besøkt den 28.04.2024).
- [36] Andrew Lock. *Understanding SameSite cookies*. 2023. URL: <https://andrewlock.net/understanding-samesite-cookies/> (besøkt den 16.05.2024).
- [37] Mark McGranghan og Eli Bendersky. *Go by Example: Goroutines*. URL: <https://gobyexample.com/goroutines> (besøkt den 29.04.2024).
- [38] Microsoft. *TypeScript for the New Programmer*. URL: <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html%5C#runtime-behavior> (besøkt den 20.05.2024).
- [39] Tom Heine Nätt. *URL*. URL: <https://snl.no/URL> (besøkt den 20.05.2024).
- [40] Oracle. *15.5 Prepared Statements*. URL: <https://dev.mysql.com/doc/refman/8.0/en/sql-prepared-statements.html> (besøkt den 29.04.2024).
- [41] Oracle. *transaction - MySQL Glossary*. URL: https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_transaction (besøkt den 29.04.2024).
- [42] Oracle. *What is a Virtual Machine (VM)?* URL: <https://www.oracle.com/no/cloud/compute/virtual-machines/what-is-virtual-machine/> (besøkt den 20.05.2024).
- [43] OWASP og kingthorin. *SQL Injection*. URL: https://owasp.org/www-community/attacks/SQL_Injection (besøkt den 29.04.2024).
- [44] Sohrab Salimi. *Pair Programming*. URL: <https://www.agile-academy.com/en/agile-dictionary/pair-programming/> (besøkt den 17.05.2024).
- [45] Scrum.org. *What is Scrum?* URL: <https://www.scrum.org/resources/what-scrum-module> (besøkt den 12.05.2024).

- [46] Suraj Shende. *Token Bucket Algorithm (Rate Limiting)*. 2023. URL: <https://medium.com/@surajshende247/token-bucket-algorithm-rate-limiting-db4c69502283> (besøkt den 10.05.2024).
- [47] Mark Smallcombe. *PostgreSQL vs MySQL: The Critical Differences*. 2023. URL: <https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/> (besøkt den 17.05.2024).
- [48] SQLite. *Internal Versus External BLOBs in SQLite*. 2011. URL: <https://www.sqlite.org/intern-v-extern-blob.html> (besøkt den 20.05.2024).
- [49] statcounter. *Desktop vs Mobile vs Tablet Market Share Norway*. 2024. URL: <https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/norway> (besøkt den 14.03.2024).
- [50] statcounter. *Screen Resolution Stats Norway*. 2024. URL: <https://gs.statcounter.com/screen-resolution-stats/all/norway> (besøkt den 14.03.2024).
- [51] *Using prepared statements*. URL: <https://go.dev/doc/database/prepared-statements> (besøkt den 29.04.2024).
- [52] Mattis Vaaland. - *Om 5 år vil det være noe helt annet å være utvikler, mener norsk AI-ekspert*. 2023. URL: <https://www.kode24.no/artikkel/om-5-ar-vil-det-vaere-noe-helt-annet-a-vaere-utvikler-mener-norsk-ai-ekspert/78782835> (besøkt den 17.05.2024).

A Prosjektplan

Opprinnelig prosjektplan fra januar 2024.

Begynner på neste side

Prosjektplan Utstyrsbanken

INNHold**Innhold**

1 Mål og rammer	4
1.1 Bakgrunn	4
1.2 Prosjekt mål	5
1.2.1 Resultat mål	5
1.2.2 Effekt mål	5
1.2.3 Lærings mål	5
1.3 Rammer	5
2 Omfang	6
2.1 Problemstilling	6
2.1.1 Oppgavebeskrivelse	6
2.1.2 Lån	7
2.1.3 Brukere av tjenesten	7
2.2 Avgrensninger	7
3 Prosjektorganisering	8
3.1 Ansvarsforhold og roller	8
3.2 Rutiner og gruppereregler	8
3.3 Plan for møter	9
3.4 Scrum	9
3.5 Scrum roller	10
3.6 Alternative prosessrammeverk	10
4 Kvalitetssikring	12
4.1 Dokumentasjon	12
4.2 Utviklingsrutiner	12
4.3 Plan for testing	12
4.3.1 Testing av API	12
4.3.2 Brukertesting	12
4.4 Risikoanalyse	13

INNHold



5 Plan for gjennomføring	14
A Aktuelle teknologier	18
A.1 Frontend	18
A.2 Backend API	18
A.3 Database	18
A.4 Verktøy	18

1 Mål og rammer

1.1 Bakgrunn

Utstyrbanken i Vestre Toten er et tilbud for lokalbefolkningen der man kan gratis låne ut utstyr til ulike fritidsaktiviteter. Utstyrbanken ligger under Vestre Toten frivilligsentral som er en ideell organisasjon. Når folk vil låne noe fra Utstyrbanken i dag må de enten dukke opp fysisk på stedet før de en gang vet om utstyret de ønsker å låne er utlånt eller ikke, eller ta kontakt med Utstyrbanken som da må manuelt se igjennom om de har utstyret som låner ønsker å låne.

Utstyrbanken

Utstyrbanken, Raufosshallen.
Prøvenvegen 65

Aldersgruppe: For alle

Tirsdag kl. 14.15 - 15.45 (ikke i skoleferier)

Torsdag kl. 17.00 - 19.00

Dersom dagene ikke passer, kan Frivilligsentralen kontaktes for å gjøre en avtale. Det bør gis beskjed i god tid.

Frivilligsentralen: **416 46 475** (dagtid)

Vaktmobil: **947 83 045** (ettermiddag)

Mangler du utstyr til friluftsliv og aktivitet?

Lån det du trenger fra Utstyrbanken.

Har du friluftsliv- og turutstyr du ønsker å gi bort?

Gi det til Utstyrbanken og bidra til at flere får mulighet til friluftsliv og aktivitet!

Figure 1: Nettsiden slik den er i dag

Oppdraget er å lage en moderne løsning som lar brukere se utstyret som er tilgjengelig og reservere det de trenger, og som lar personalet få oversikt over utstyret de har og registrere lån som blir hentet og levert tilbake.

1.2 Prosjekt mål



1.2 Prosjekt mål

1.2.1 Resultatmål

Målet er å lage en nettside i form av en progressiv web applikasjon for utlån av utstyr som skal være klar til bruk for de som jobber i Utstyrsbanken og for lånerne. Prosjektet bør også være godt dokumentert og koden burde være lett å lese for eventuelle personer som skal videreutvikle løsningen.

1.2.2 Effektmål

- Løsningen skal gjøre det lettere å benytte tilbudet. Etter 3 måneder er det ønskelig at antall lån per måned er økt med 25% sammenlignet med før løsningen ble tatt i bruk.
- Løsningen skal gjøre det lettere for de ansatte å holde oversikten over tilgjengelig utstyr, hvilket utstyr som skal inn osv. Tid brukt på slik administrasjon reduseres med 30% sammenlignet med før løsningen ble tatt i bruk.

1.2.3 Læringsmål

- Lære hvordan det er å jobbe for en arbeidsgiver
- Lære hvordan effektivt jobbe i en gruppe gjennom bruk av metodikker og kommunikasjon
- Lære hvordan forbedre produktet gjennom brukertester
- Lære om hvordan man utvikler en moderne progressiv webapplikasjon

1.3 Rammer

Oppdragsgiver ønsker en progressiv web applikasjon (PWA). Dette krever etableringen av noen teknologiske rammer:

- Nettsiden skal fungere på Chromium-baserte nettlesere (Chrome, Edge, Opera osv.), nettsiden bør også fungere på Firefox og Safari.
- Nettsiden skal fungere på iOS fra og med versjon 11.3 selv om PWAer ikke er like godt støttet på iOS.
- Planen er å bruke NTNU sin SkyHigh-tjeneste (OpenStack) for å ha en server for testing og utvikling i løpet av prosjektet. I fremtiden når løsningen settes i drift må den kunne kjøre hos en cloud-tilbyder som AWS, Google Cloud eller Microsoft Azure. Dette er et krav som følger av at Utstyrsbanken ikke har ressurser til fysisk server-hardware for å drifte løsningen.

Det er også relevant lovverk som må følges når man utvikler en slik løsning. Personopplysningsloven [3] (samt EU sin GDPR) er spesielt viktig siden løsningen skal håndtere informasjon som navn og telefonnummer. Alle IKT-løsninger (som inkluderer nettsider) omfattes også av regelverk knyttet til universiell utforming basert på WCAG-standarden [1] [5]. På grunn av at Utstyrsbanken er tilknyttet det offentlige via kommunen betyr det at løsningen må følge de strengere kravene som er satt for offentlig sektor sammenlignet med privat sektor.

2 Omfang

Omfanget til tjenesten vil involvere alle feltene innenfor en full-stack applikasjon. Planen er å prøve å lage en Progressiv Web-app (PWA). Effektivt er dette en webside som kan også installeres lokalt på hvilken som helst moderne enhet (f.eks. smarttelefon, nettbrett eller laptop). Denne siden/applikasjonen vil fungere som en frontend, som kan kommunisere med en server (backend) for å hente ut og modifisere aktuell data fra en database, f.eks. lister over tilgjengelig utstyr, inklusive bilder.

Et annet fokus vil være å optimalisere tjenesten for å kunne driftes enkelt og billig, samtidig som at tjenesten har et intuitivt brukergrensesnitt slik at personer som reserverer utstyr har en positiv opplevelse.

2.1 Problemstilling

2.1.1 Oppgavebeskrivelse

Vår oppgave er å levere en progressiv web applikasjon til Utstyrbanken, som forenkler og moderniserer utlånsprosessen de allerede har. Planen for funksjonaliteten til programvaren ser slik ut:

- Webapplikasjonen skal ha et brukersystem med 3 forskjellige roller: Bruker, Ansatt, Administrator
- Det skal finnes en oversikt over inventaret, det vil si utstyret som eksisterer i Utstyrbanken, som brukere og ansatte har tilgang til å se.
 - Inventarlisten skal kunne dynamisk oppdatere seg basert på reserveringer og utlån som gjennomføres
 - Inventarlisten skal være intuitiv å fylle opp og søke igjennom, basert på sesong og idrett
 - Hver gjenstand kan ha et bilde og skal holde telle på hvor mange av hver størrelse som er tilgjengelig, reservert og lånt ut (de to sistnevnte kan kun sees av de ansatte)
 - Hvert type utstyr og kategori skal samle statistikk. Noen eksempler er antall utlån av hver størrelse, hvor ofte gjenstanden blir ødelagt og mistet, og hvor lenge utstyret ble lånt.
- Brukerne skal kunne reservere utstyret fra Utstyrbanken. Listen viser det som er tilgjengelig for øyeblikket, og de ansatte registrerer i systemet når utstyret er utlevert til bruker (utlån / lån).
- Når bruker leverer tilbake utstyret registrerer de ansatte også dette i systemet.
- Bruker skal kunne forlenge lån av utstyr en gang og bli informert nærmende innleveringsfrist eller purring gjennom nettsiden, SMS eller ved oppringning fra ansatte
- Ansatte skal kunne legge til nytt utstyr og nye kategorier gjennom applikasjonen
- Det skal kun være en administrator som kan endre roller på brukere. For øyeblikket er det kun aktuelt å promotere en bruker opp til ansatt.
- Administrator har tilgang på en revisjonslogg som viser handlingene til alle brukerne i systemet.

For å avklare usikkerheter knyttet til dette blir det viktig å ha god kontakt med oppdragsgiver slik at det ikke blir noen misforståelser, det er planlagt faste møter for dette (se kapittel 3).

2.2 Avgrensninger



2.1.2 Lån

Når det kommer til å håndtere registreringen av reserverasjoner og utlån elektronisk.

- Grenser på lån og reserverasjoner per person. En bruker kan finne på å reservere større deler av inventaret. Dette kan forhindre andre som ønsker å låne det samme utstyret.
- Varsler om status på reserverasjoner og lån som «klar for henting» slik at folk ikke glemmer reserverasjonene/lånene sine
- Lengden på lån og hvor lenge en reserverasjon kan være før den må bli hentet bør være konfigurabelt av administrator av siden.

Det finnes andre lånesystemer for utstyr som for eksempel nettsiden til BUA¹. Til forskjell fra Utstyrsbanken Vestre Toten består BUA av mange flere utstyrssentraler. Nettsiden deres har en del av den samme funksjonaliteten som er ønskelig vår løsning, men det er noen endringer som Utstyrsbanken ønsker:

- Det trengs ikke et kart for å velge hvilken sentral man ønsker å reservere fra.
- Layouten på siden som viser det tilgjengelige utstyret har mye tomrom, har for store bilder og svært liten tekst, spesielt på PC. Det ønskes en mer kompakt layout på vår løsning.

2.1.3 Brukere av tjenesten

Med order brukere definerer vi det som en person/låner, en ansatt hos Utstyrsbanken, eller en administrator som bruker tjenesten. Det er flere punkter å ta hensyn til for brukerne av tjenesten:

- Lagring av personlig data
- Bekreftelse av identitet under registrering
- Forskjellige roller og tillatelser i systemet
 - Vanlige personer kan kun se og reservere utstyr
 - En ansatt kan registrere at utstyr blir lånt ut, og at utstyret kommer tilbake til Utstyrsbanken
 - En administrator kan gjøre det samme som en ansatt, men skal også ha funksjonalitet for å administrere tjenesten. (F.eks. legge til nye ansatte, eller konfigurere grenser på utlån)

2.2 Avgrensninger

- Vi har ikke ansvaret for registrering av Utstyrsbankens nåværende inventar.
- Ved avslutningen av prosjektet er det Utstyrsbanken som skal videreføre prosjektet. De tar over driften og vedlikehold av servere og dataene som har blitt brukt.
- Tjenesten skal ikke visualisere statistikken som samles inn om lånene. Det er derimot aktuelt å tilby en måte å eksportere dataene slik at de ansatte kan benytte dataen.

¹BUA[2] er en liknende tjeneste til Utstyrsbanken. Se <https://www.bua.no/om-oss>

3 Prosjektorganisering

3.1 Ansvarsforhold og roller

Gruppen vår har bestemt seg for å fordele ansvaret mellom seg slik:

Rolle	Person	Beskrivelse
Frontend ansvarlig	Ferdinand	Hovedansvar for utviklingen av frontend delen av programvaren.
Innleveringsansvarlig	Ferdinand	Ansvarlig for å gjennomføre innleveringer ved aktuelle frister.
Scrum mester	Ferdinand	Ansvarlig for å kalle inn til møter og presentere sprinter med arbeidsgiver.
Møtereferent	Ferdinand	Fører referater av møter.
Backend ansvarlig	Åsulv	Har hovedansvaret for backend programvaren. Dette inkluderer API'en for kommunikasjonen mellom frontend og backend.
Database-designer	Erlend	Har ansvaret for å designe databasesystemet. Dette involverer planlegging, diagrammer, normaliserte Påbegyntsering, og hjelp med implementering av den.
Reservereferent	Erlend	Fører referater av møter hvis hoved-referent ikke er tilstede.
Ansvar for drift	Noah	Har ansvaret for å drifte serverne og OpenStack miljøet. Hensikten er for testing av systemet.
Kontaktperson	Noah	Har ansvaret for å opprettholde kontakten mellom oppdragsgiver og veileder.
Rombestiller	Noah	Bestiller rom til enhver tid for å holde samlinger/møter.

Table 1: Grupperollene

3.2 Rutiner og gruppe-regler

Mål for tidsbruk til prosjektet er 30 timer i uka per person. Tidsbruken skal loggføres, og hvert gruppemedlem har selv ansvar for å loggføre sin egen timebruk. Det forventes at hver person jobber med oppgavene de er tildelt. Dersom man trenger hjelp bør man ta kontakt med de andre gruppemedlemmene.

Dersom fravær planlegges, meld ifra til resten av gruppa minst 1 dag før. Dersom eventuelle forsinkelser oppstår på vei til møte (f.eks. sen buss/tog), meld ifra via Discord eller telefon så fort som mulig.

Dersom en person ikke møter opp, men er forventet å møte opp, går det greit dersom det er snakk om engangstilfellet eller særlige gode grunner. Dersom det skjer gjentatte ganger over tid blir det et problem som må tas opp med veileder.

Det forventes at gruppemedlemmene skal være tilgjengelig til å svare på meldinger på Discord eller mobil mellom kl. 09:00-18:00 hver ukedag. De faste møtene har møteplikt.

Når det gjelder uenigheter i gruppen er det i utgangspunktet flertallet som bestemmer. Dersom vi fortsatt ikke kommer til en beslutning er det den personen som har hovedansvar ved området som har siste ord (eksempelvis front-end ansvarlig har siste ord i front-end uenigheter). Dersom det oppstår større problemer tas det opp med veileder.

3.3 Plan for møter



3.3 Plan for møter

- Statusmøter skal alltid starte med det siste møtes møtelogg, dette gjør at alle på gruppen er oppdatert. Mandagsmøter blir i oddetallsukene der forrige sprint avsluttes med en gjennomgang (Sprint Review), og følges opp med planlegging av neste sprint (Sprint Planning).
- Oppfølgingsmøtene på tirsdager og onsdager er ment for å få starte arbeidsdagen. Hvert gruppedlem informerer om hva de har gjort siden sist de møttes, hva de planlegger å jobbe med for dagen, og melder om eventuelle problemer som har dukket opp. Disse møtene fungerer effektivt som standupmøter.
- Torsdagsmøter med veileder er ment for å evaluere hvordan det går med prosjektet og for eventuelle spørsmål spesielt tilknyttet rapporten.
- Fredagsmøter i partallsukene blir det møte med oppdragsgiver der vi viser fremgangen i prosjektet siden sist og tar opp eventuelle spørsmål.

Dag	Tid	Beskrivelse
Tirsdag	10:00 - 10:15	Digitalt oppfølgingsmøte
Onsdag	10:00 - 10:15	Digitalt oppfølgingsmøte
Torsdag	13:30 - 14:00	Møte med veileder
Torsdag	14:00 - 16:00	Gruppemøte

Table 2: Faste møter

Dag	Tid	Beskrivelse
Mandag	10:00 - 12:00	Sprintmøte (planlegge ny Sprint, og Sprint Review)
Fredag	12:00 - 15:00	Gruppemøte

Table 3: Møter på oddetalls-uker

Dag	Tid	Beskrivelse
Mandag	10:00 - 12:00	Gruppemøte
Fredag	12:00 - 14:00	Møte med oppdragsgiver
Fredag	14:00 - 15:00	Gruppemøte

Table 4: Møter på partallsuker

3.4 Scrum

Det valgte prosessrammeverket for dette prosjektet er en modifisert variant av Scrum [6]. Sprintlengder er satt til to uker for å bedre passe inn med møtene med arbeidsgiver (annenhver fredag). Dette gir oss muligheten å oppsummere sprinten for arbeidsgiver før den avsluttes.

Andre styrker for Scrum er følgende:

- Det er lettere å be om hjelp hvis en i gruppen sitter fast på en oppgave
- Scrum master kaller bare inn til møte, hvor gruppen bestemmer hva de skal gjøre i felles. Dette er i motsetning til tradisjonelle ledere som bestemmer hva hver person skal gjøre.
- Alle vet hva alle gjør og hva som må gjøres til neste sprint
- Setter lik vekt på alle i gruppen, dette leder til at alle føler likt ansvar og motivasjon for prosjektet.

3.5 Scrum roller



- Scrum gir også muligheten til å justere på planene underveis dersom f.eks. en oppgave tok lengre tid enn planlagt, eller at noe uforutsett oppstår i løpet av prosjektet. I tillegg er det mulig å fritt hoppe frem og tilbake mellom de ulike utviklingsstadiene. Med andre ord smidig metodikk er en mye friere arbeidsprosess enn for eksempel "waterfall" der man lager en detaljert plan og følger den steg for steg.
- Alle møter skal ha en logg/referat, dette gjør det lettere å følge opp punkter fra tidligere møter, samt at hvis en er syk så kan de lese møteloggen i etterkant.

3.5 Scrum roller

Diagrammet (figur 2) viser frem Scrum-rollene, hvordan de henger sammen og hvem som har hver rolle.

- **Produkteier / Arbeidsgiver - Utstyrskbanken, Gunnhild**
 - Gunnhild har tatt kontakt med NTNU og kommet med oppgaven, oppgaven skal følge hennes visjon av hva Utstyrskbanken trenger
 - Oppgaven er gitt på vegne av Vestre Toten Utstyrskbank som Gunnhild er daglig leder for, dette gjør Utstyrskbanken til bedriftseier av produktet
 - Arbeidsgiver sin oppgave er å gi tilbakemelding på prosjektet, komme med ønsker f.eks. ny funksjonalitet og håndtere ressursforespørsler (som Cloud servere)
 - Gunnhild er vår representant fra Utstyrskbanken
- **Rådgiver / Veileder - Peter**
 - Skal følge opp Scrum-gruppen og sørge for at den fungerer
 - Skal kunne gi tilbakemelding på prosjektet, rapporten, tidsbruk og hva som bør prioriteres
- **Scrum Master - Ferdinand (vara Erlend)**
 - Skal diskutere kravspesifikasjoner, avgrensinger og gjennomførbarhet med arbeidsgiver
 - Skal skrive møtereferat
- **Utviklere - Erlend, Ferdinand, Noah, Åsulv**
 - Har ansvar å gjennomføre de tildelte arbeidsoppgavene hver sprint
 - Overholde levering innenfor de ulike fristene
 - Kommunisere problemer og behov for hjelp

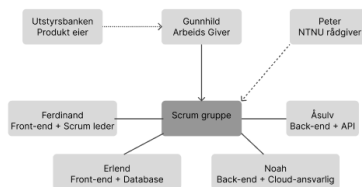


Figure 2: Rollefordelingen visualisert

3.6 Alternative prosessrammeverk

Scrum er ikke den eneste modellen vi har vurdert, noen eksempler på andre modeller er:

- Waterfall, som ikke er en modell som passer til prosjektet. Vår oppgave er å utvikle et program som skal erstatte et manuelt katalog system, som er basert på bruker tillit om retur og identifisering. Prosessen av å digitalisere en slik menneskelig prosess kommer til å føre til mange uforutsette friksjonspunkter som kommer til å være vanskelige å planlegge rundt. Dette vil si at vi trenger en smidig (agile) modell.

3.6 Alternative prosessrammeverk

- Lean modellen passer bedre for dens vekt på prototyper, men passer ikke for utviklingstiden vi har. Lean modellen handler om å få programmet så fort ut til testbrukere, med et stort fokus på MVPs. Etter produktet er ute til testbrukere er det forventet at utviklerne skal iterere over programvaren og jobbe på feil som dukker opp ved bruk. Vi har heller ikke tilgang på nok brukertestere til å gjennomføre ekstensiv testing.



4 Kvalitetssikring

4.1 Dokumentasjon

All kildekode skal kommenteres der det skjer mindre åpenbar logikk i koden. Dette hjelper andre gruppe-medlemmer (og evt. fremtidige utviklere som tar over prosjektet) med å forstå koden. Backend API-en sin funksjonalitet skal dokumenteres for å gjøre det lettere å benytte API-en for frontend-utviklerne. Det planlegges også å bruke relevante standarder for variabel- og funksjonsnavn til programmeringsspråkene vi bestemmer oss for å bruke.

4.2 Utviklingsrutiner

Vi benytter Git via NTNU IDI sin GitLab instans som versjonskontroll for all kildekoden involvert i prosjektet. Dette gir oss muligheten til å ha en versjon av kildekoden som er kvalitetssikret (main-branch) samtidig som ny funksjonalitet utvikles parallellt. Dette bør som regel alltid skje på en annen branch. For at dette skal fungere optimalt har vi dannet noen rutiner knyttet til dette:

1. Hoved-branchen (main) skal ikke pushes til direkte. All kode skal gå gjennom merge requests slik at endringene kan inspiseres og godkjennes av andre gruppe-medlemmer først. Dette er spesielt nyttig for å fange opp slurvfeil, problemer, bugs, og lignende.
2. Man skal derfor lage en ny branch for endringene sine under utviklingen. Branches er knyttet til spesifikk funksjonalitet som skal implementeres eller fiksing av spesifikke feil/bugs.
3. Alle commit-meldinger skal beskrive endringene på en forståelig måte, helst kort og konsist. Dersom det blir lagd commits knyttet til spesifikke problem (f.eks. et issue i GitLab), bør problemet nevnes i commit-meldingen.

4.3 Plan for testing

4.3.1 Testing av API

Det skal fra starten av prosjektet forsøkes å tilrettelegge for automatisert testing av backend sin API. Automatisert testing vil sikre at koden gjør det som er forventet uten at man må manuelt teste ulike scenarier, dette er nyttig for å eksempel verifisere at gammel funksjonalitet fortsatt fungerer etter nye endringer i koden. Viktig kjernefunksjonalitet i APIen skal prioriteres dersom vi trenger tid til noe annet.

4.3.2 Brukertestning

For å sikre at applikasjonen er brukervennlig både for lånere som skal reservere utstyr, og de på Utstyrsbanken som skal registrere utlån, har vi planlagt å gjennomføre brukertester. Disse skal gjennomføres når det er implementert nok funksjonalitet i programvaren til at den lar seg testes på denne måten. Planen er å gi 3-5 personer en oppgave de skal gjennomføre, hvor vi observerer og noterer hvordan det går, og etterpå diskuterer med de erfaringene deres. De skal ikke guides underveis, siden vi er interessert i hvordan de greier oppgavene på egenhånd. Et praktisk eksempel for en bruker kan være å reservere to par ski og to par skistaver. Resultatene av disse testene blir drøftet på og brukt for å gjøre justeringer på brukergrensesnittet slik at nettsiden blir enklere å bruke for målgruppene våre.

4.4 Risikoanalyse

4.4 Risikoanalyse

Med risikoanalysen har vi analysert hvilke risikoer som kan oppstå under prosjektet, hvor sannsynlig de er at de inntreffer, og hvor alvorlig det er hvis de inntreffer. Skalaen brukt er lav, middels og høy, siden det er vanskelig å vurdere mer nøyaktig. For noen av risikoene har vi vurdert tiltak for å redusere sannsynligheten og/eller alvorligheten.

Nummer	Beskrivelse	Resultat	Risiko	Alvorlighet	Tiltak?
1	Tap av viktig data til prosjektet (kildekode, dokumenter osv)	Mye jobb å begynne på nytt, stor tap av progresjon	Middels	Høy	Ja
2	En eller flere av gruppe medlemmene blir syke	Progresjon i prosjektet går tregere	Middels	Høy	Nei
3	En eller flere av gruppe medlemmene mister internettilgangen sin	Digitale møter og bruk av online verktøy blir umulig, ikke mulighet for å pushe og pulle fra Git repoer osv	Middels	Lav	Nei
4	Sårbarheter/feil i programvaren oppstår	Kan forårsake problemer med drift, lekkasje av data, uønsket sletting/ending av data osv	Middels	Høy	Ja
5	Tidspress, vi rekker ikke å nå målene vi har satt i tide	Oppgaver blir ikke fullført	Middels	Middels	Nei
6	Programvaren bryter med GDPR / personvernslover	Kan få konsekvenser for Utstyrsbanken / kommunen	Lav	Høy	Nei
7	Programvaren bryter med lowerk rundt universiell utforming	Færre brukere kan benytte tjenesten, kan få konsekvenser for Utstyrsbanken / kommunen	Lav	Høy	Nei
8	Programvaren er ikke kompatibel med en eller flere populære nettlesere	Færre brukere kan benytte tjenesten	Lav	Middels	Ja
9	Bruk av ny teknologi kan forsinke utviklingen	Progresjon i prosjektet går tregere	Lav	Middels	Nei
10	Uforutsette hendelser som vær og naturkatastrofer	Fysisk oppmøte kan bli vanskelig, generelt jobbing med prosjektet kan bli utfordrende.	Lav	Høy	Nei
11	Dårlig skrevet/dokumentert/strukturert kode	Kan føre til utfordringer når man skal fikse feil eller videreutvikle prosjektet, dette gjelder spesielt om noen andre skal ta over utviklingen etter oss.	Middels	Middels	Ja

Figure 3: Analyse av potensielle risikoer som kan oppstå under prosjektet

	Lav	Middels	Høy	Risiko
Lav				
Middels	3, 8, 9	5, 11		
Høy	6, 7, 10	1, 2, 4		
Alvorlighet				

Figure 4: Hvordan de ulike risikoene fordeler seg mellom sannsynlighet og alvorlighet. Nummerene tilsvarer risikoen listet opp i tabell 3

Risikonummer	Tiltak
1	Etabler gode backuprutiner for viktig data knyttet til prosjektet (helst automatiserte daglige backups)
2	Omfordeling av arbeidsoppgaver mellom gruppe medlemmene som ikke er syke
4	Kjente potensielle sårbarheter som SQL injections bør forhindres, en bør også tenke på f.eks. å ikke stole blindt på hva brukere sender inn (sett begrensinger som lengde)
7	Før man implementerer ny funksjonalitet som krever visse funksjoner av nettleseren, gjør research om kompatibilitet i de ulike nettleserne
10	Etabler standarder for dokumentasjon, samt følg "best practise" når det kommer til f.eks. funksjonsnavn, variabelnavn.

Figure 5: Tiltak som er vurdert for å redusere sannsynlighet og/eller alvorlighet av noen av de viktigste risikoene som er identifisert i Risikoanalysen 3

5 Plan for gjennomføring

Gantt-diagrammet (6) viser hvordan oppgavene foreløpig blir i hver sprint. Som følge at vi bruker Scrum-prosessen kan dette endre seg underveis.

Foreløpige milepæler i løpet av prosjektet ser ut til å bli:

- **31/01/2024:** Prosjektplan er ferdig. Utviklingen av løsningen kan begynne for fullt.
- **24/03/2024:** MVP (Minimum Viable Product) fullføres og vi er klare til brukertest. I MVPen skal vi ha følgende funksjonalitet tilgjengelig i frontend:
 - Innloggingsmulighet for brukere
 - (**Bruker**) Se oversikt over inventar
 - (**Bruker**) Mulighet for reservasjon av utstyr
 - (**Ansatt**) Registrering av utlån
 - (**Ansatt**) Registrering av utstyr som er levert tilbake
 - (**Administrator**) Brukerhåndtering (endre data, telefonnummer, promotering til ansattrolle)
 - (**Administrator**) Legge til nytt/endre data om utstyr
- **31/03/2024:** Brukertest gjennomført. Dette ble gjort i hovedsak for å evaluere hvor intuitiv brukergrensesnittet er for de som skal bruke den.
- **21/04/2024:** Utvikling rundes av. Det blir fokus på skriving av endelig prosjektrapport. Følgende bør bli gjort innen da:
 - Benyttet tilbakemeldinger fra brukertest for å forbedre brukergrensesnittet
 - Eventuell implementasjon av funksjonalitet som ikke er del av MVP (notifikasjoner, statistikk)
- **21/05/2024:** Endelig prosjektrapport leveres inn.

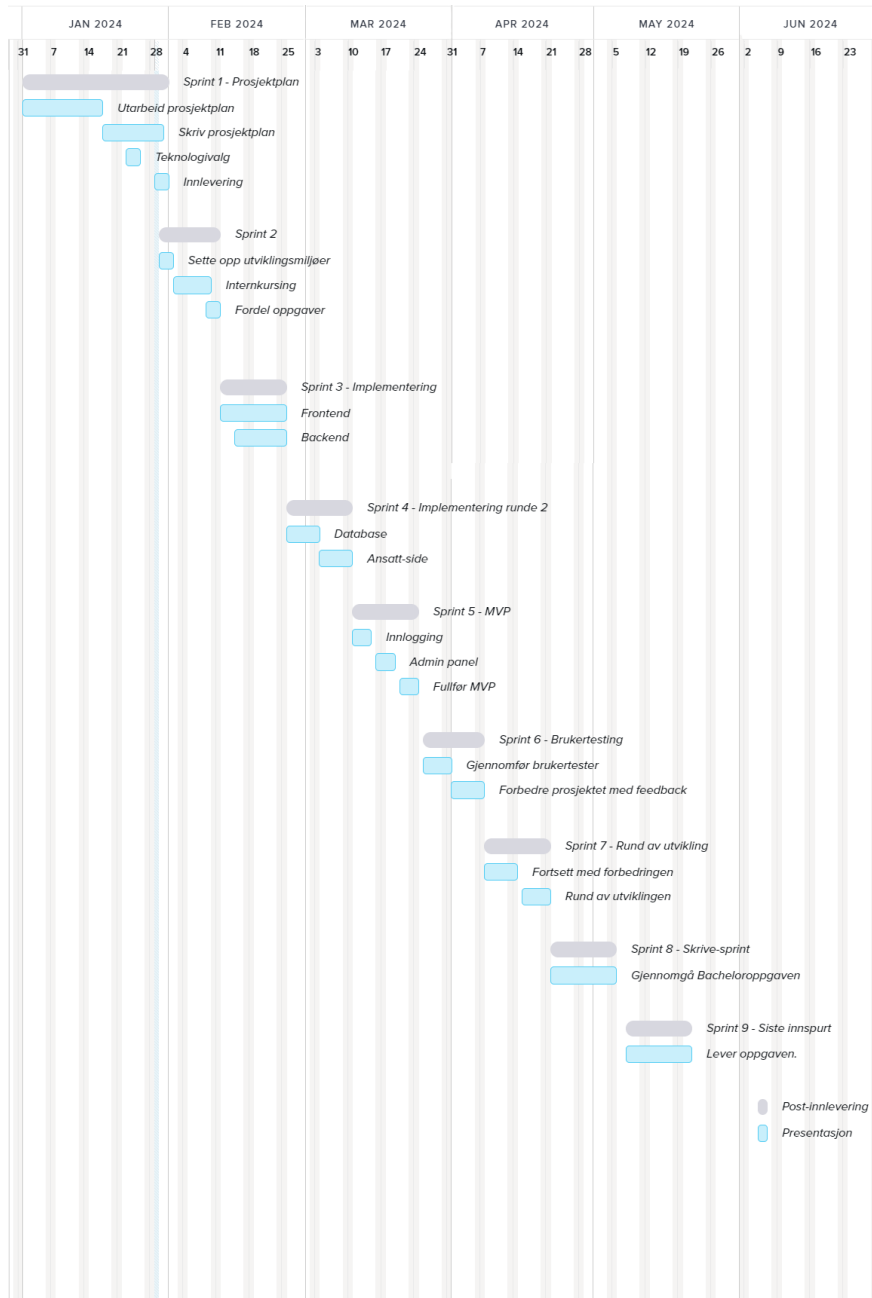


Figure 6: Gantt Diagram

Oppgave	Dato	Periode
Utarbeid Prosjektplan	01/1/24 - 14/1/24	Sprint 1 - Prosjektplan
Skriv prosjektplan	14/1/24 - 28/1/24	Sprint 1 - Prosjektplan
Teknologivalg	23/1/24 - 25/1/24	Sprint 1 - Prosjektplan
Innlevering	29/1/24 - 31/1/24	Sprint 1 - Prosjektplan
Sette opp utviklingsmiljøer	30/1/24 - 01/2/24	Sprint 2
Internkursing	02/2/24 - 09/2/24	Sprint 2
Fordel oppgaver	09/2/24 - 11/2/24	Sprint 2
Frontend	12/2/24 - 25/2/24	Sprint 3 - Implementering
Backend	15/2/24 - 25/2/24	Sprint 3 - Implementering
Database	26/2/24 - 03/3/24	Sprint 4 - Implementering runde 2
Ansatt-side	04/3/24 - 10/3/24	Implementering runde 2
Innlogging	11/3/24 - 14/3/24	Sprint 5 - MVP
Admin panel	16/3/24 - 19/3/24	Sprint 5 - MVP
Fullfør MVP	20/3/24 - 24/3/24	Sprint 5 - MVP
Gjennomfør brukertester	25/3/24 - 31/3/24	Sprint 6 - Brukertesting
Forbedre prosjektet med feedback	01/4/24 - 07/7/24	Sprint 6 - Brukertesting
Fortsett med forbedringen	08/4/24 - 14/4/24	Sprint 7 - Rund av utvikling
Rund av utviklingen	15/4/24 - 21/4/24	Sprint 7 - Rund av utvikling
Gjennomgå Bacheloroppgaven	22/4/24 - 05/5/24	Sprint 8 - Skrive-sprint
Lever Oppgaven	08/5/24 - 21/5/24	Sprint 9 - Siste innspurt
Presentasjon	04/6/24 - 06/6/24	Etter innlevering

Table 5: Oversikt over sprintene

REFERANSER**Referanser**

- [1] World Wide Web Consortium (W3C). *Web Content Accessibility Guidelines (WCAG) 2.1*. 2023. URL: <https://www.w3.org/TR/WCAG21/> (visited on 01/26/2024).
- [2] BUA. *Om oss*. Referanse til konkurrent. URL: <https://www.bua.no/om-oss> (visited on 01/31/2024).
- [3] Datatilsynet. *Om personopplysningsloven med forordning og når den gjelder*. 2023. URL: <https://www.datatilsynet.no/regelverk-og-verktoy/lover-og-regler/om-personopplysningsloven-og-nar-den-gjelder/> (visited on 01/26/2024).
- [4] Steffen Granberg, Steinar Opphus, and Ole André Slettum. "Viten i senter". Deler av prosjektplanen er inspirert av denne bacheloroppgaven. Norsk teknisk-naturvitenskapelige universitet, 2017.
- [5] Tilsynet for universell utforming av ikt. *WCAG-standarden (for offentlige nettsider)*. URL: <https://www.uutilsynet.no/wcag-standarden/wcag-standarden/86?f%5B0%5D=t1%3A207&f%5B1%5D=t2%3A129> (visited on 01/25/2024).
- [6] Scrum.org. *What is Scrum?* URL: <https://www.scrum.org/resources/what-scrum-module> (visited on 01/31/2024).

A Aktuelle teknologier

A.1 Frontend

For frontend ender vi sannsynligvis opp med å bruke et JavaScript-rammeverk som:

- React
- Vue
- Svelte

Dette vil gjøre det lettere å utvikle frontend fremfor å bruke ren HTML/CSS/JavaScript, derfor har det også i praksis blitt bransjestandard innen frontend for web.

A.2 Backend API

Noen av programmeringsspråkene vi kan bruke for backend API-en vår kan være

- Go
- Java
- JavaScript (via Node.js)

Go er noe teamet har erfaring med til backend API-er fra før av og egner seg godt til dette. Samtidig er nok for eksempel Java mer populært blant utviklere. En på teamet har også erfaring med Java.

A.3 Database

Vi ser for oss en relasjonell database for dette prosjektet, da er SQL-baserte databaser det mest relevante, her har vi primært tre valg:

- SQLite
- MySQL/MariaDB
- PostgreSQL

Eventuelt kan skybaserte SQL-løsninger også vurderes.

A.4 Verktøy

- Overleaf (skrive prosjektplan, rapport)
- GitLab (issue tracking, lagring av Git-repoer)
- Git (versjonskontroll)
- Vipps API (logg inn, federering)
- Visual Studio Code (IDE)

B Standardavtale

Standardavtale mellom de ulike partene involvert i prosjektet, altså studentene i gruppen, oppdragsgiver, veileder og NTNU.

Begynner på neste side



Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett

Er den rett som den som skaper et åndsverk har til å fremstille eksemplarer av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater

Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater

Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn

Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring

Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:		
Veileder ved NTNU:	Peter Nussbaum	
e-post og tlf.	peter.nussbaum@ntnu.no	95872815
Ekstern virksomhet:	Vestre Toten Frivilligsentral	
Ekstern virksomhet sin kontaktperson, e-post og tlf.:	Gunnild Andersen, post@vestretoten.frivilligsentral.no, ☎ 41646475	
Student:	Noah Bjerkli Aanonli	
Fødselsdato:	03-07-2001	
Ev. flere studenter ¹	(ordnede alfabetisk) Erlend Thingnes 24.03.99 Asulv Vaa 02.11.01	

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato:	15/1-24
Sluttdato:	21/5-24

Opgavens arbeidstitel er:

Utstyrskassen

¹ Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Tas underveis.

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven². Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

² Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

Videreføre og videreutvikle

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss	Sett dato
<input type="checkbox"/>	ett år
<input type="checkbox"/>	to år
<input type="checkbox"/>	tre år

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: <i>V. Emneansvarlig</i>	<i>Peter Nussbaum</i>
Dato: <i>5/2-24</i>	
Veileder ved NTNU:	<i>Peter Nussbaum</i>
Dato:	Digitally signed by Peter Nussbaum Date: 2024.01.30 10:24:23 +01'00'
Ekstern virksomhet: <i>Gunnhild Andersen</i>	
Dato: <i>26/1-24</i>	
Student: <i>Noah Bjørn Aarvick</i>	
Dato: <i>26-01-2024</i>	
Ev. flere studenter: <i>Erlend Thingnes</i>	<i>Åsolv Vaa</i>
	<i>(Signature)</i>

C Use cases

Use cases ble ferdige 28. Mars og ble brukt hovedsaklig for kommunikasjon og var ikke ment som en plan som måtte bli fulgt. Det er dermed ingen garanti for at dette er det vi ender opp med. Et use case, eller brukstilfelle er en måte en bruker kan få nytte ut av systemet. D

Use Case 1: Registrering

Aktør: Besøkende

Mål: Få tilgang til funksjoner som krever en innlogget bruker

Prebetingelser: Bruker har ikke en konto, og er ikke logget inn

Detaljert hendelsesforløp:

1. Besøkende trykker på registrering knappen
2. Besøkende skriver inn fornavn, etternavn, telefonnummer og passord to ganger
3. Systemet validerer informasjonen
4. Besøkende får tilsendt aktiveringskode på tekstmelding til telefonnummeret som ble skrevet inn og et felt for å skrive det inn dukker opp på grensesnittet
5. Besøkende skriver inn aktiveringskoden.
6. Besøkende får beskjed om at de nå har en bruker og blir vist en knapp for å gå tilbake til inventar siden

Alternativ flyt:

- 1a. Besøkende trykker på en funksjonalitet som egentlig krever en innlogget bruker som å legge til utstyr i handlekurven, og deretter trykker på registrering knappen
- 2a. Besøkende skrev inn ugyldig data og blir bedt om å prøve på nytt. Kan skje hvis for eksempel telefonnummeret allerede er registrert på en bruker
- 2b. Besøkende skriver også inn postnummer
- 5a. Besøkende ber om ny kode. Besøkende får tilsendt ny aktiveringskode hvis de ikke har spurt om en for ofte
- 5b. Tiden på aktiveringskoden går ut og hvis besøkende prøver med denne koden får de beskjed som sier det

Postbetingelser:

- Besøkende er nå en bruker
- Besøkende er nå logget inn

Unntak:

- 5b: Besøkende avbryter registreringen siden aktiveringskoden utløper

Use Case 2: Innlogging

Aktører: Bruker, Ansatt og Admin

Mål: Få tilgang til funksjoner som krever en innlogget bruker

Prebetingelser: Brukeren er ikke logget inn

Detaljert hendelsesforløp:

1. Brukeren trykker på logg inn som er på headeren
2. En boks med felt for telefonnummer og passord dukker opp midt på skjermen
3. Brukeren skriver inn telefonnummer og passord
4. Hvis informasjonen stemmer blir brukeren logget inn og fortsetter der de var når klikket på innlogging knappen

Alternativ flyt:

- 3a. Hvis brukeren har glemt passordet, så kan de velge å lage et nytt passord. En tekstmelding med en kode blir sendt til brukerens telefonnummer. Deretter skriver brukeren inn koden de ble sendt på tekstmelding. Til slutt hvis koden var riktig kan bruker skrive inn nytt passord.
- 4a. Hvis ikke informasjonen stemmer blir brukeren bedt om å prøve igjen. Dette skjer hvis telefonnummeret og passordet ikke stemmer med en bruker i systemet.

Unntak:

- 3a: Brukeren avbryter innlogging

Postbetingelser:

- Brukeren er logget inn og sesjonen har startet
-

Use case 3: Logg ut

Aktører: Bruker, Ansatt og Admin

Mål: Å logge ut

Prebetingelser: Brukeren er logget inn

Detaljert hendelsesforløp:

1. Brukeren trykker på logg ut knappen på bruker menyen

2. Sesjonen avsluttes
3. Informasjonskapsler til brukeren blir slettet/tømt
4. Brukeren blir så logget ut

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelser:

- Bruker er logget ut
- Alle sesjoner for den brukeren blir slettet

Use Case 4: Reserver

Aktører: Bruker, Ansatt og Admin

Mål: Få tak i utstyr

Prebetingelser: Brukeren er logget inn

Detaljert hendelsesforløp:

1. Brukeren klikker på en utstyrstype
2. Brukeren velger antallet de vil ha av utstyret.
3. Brukeren velger hvilken av de åpne dagene de vil hente utstyret opp til to uker fram i tid
4. Brukeren får se dagen de må levere hvis lånet ikke blir utsatt av brukeren eller utstyrsbanken senere bestemmer seg for å stenge den dagen
5. Brukeren trykker på knappen for å fullføre reservasjonen
6. Brukeren får bekreftelse om at reservasjonen ble skapt

Alternativ flyt:

- 2a. Hvis det er flere størrelser velger bruker størrelsen i tillegg til antallet av størrelsen
- 3a. Brukeren får beskjed hvis antallet overskrider grensen for utstyrstypen (skjer bare for vanlige brukere) og de kan velge på nytt
- 6a. Bruker får beskjed om at de ikke kan hente eller levere på valgt dag hvis utstyrsbanken stengte i løpet av reserveringen og de kan velge på nytt
- 6b. Brukeren får beskjed hvis det ikke lenger er nokk gjenstander av størrelsen og utstyrstypen som ble valgt og de kan velge på nytt

Unntak:

3a: Det er ingen tilgjengelige dager å hente eller levere på og bruker kan ikke reservere

5b: Brukeren avbryter reserveringen

Postbetingelser:

- En reservasjon har blitt lagt til i systemet.
- Statistikken om bruk av utstyr oppdateres.
- Hendelsesloggen oppdateres
- Reservasjonen blir lagt til i historikken

Use Case 5: Se inventar

Aktører: Bruker, Ansatt og Admin, Besøkende

Mål: Se hvilket typer utstyr som kan være tilgjengelige får lån

Prebetingelser: Ingen

Detaljert hendelsesforløp:

1. Brukeren blir vist rader med utstyrstyper sine navn og bilder fra startside
2. Brukeren scroller ned for å se resten av utstyret fordi det bare er en side

Alternativ flyt

2a. Brukeren søker etter utstyr i søkefeltet

2b. Brukeren filtrer etter kategori

Postbetingelser: Bruker vet hvilke typer utstyr utstyrsbanken tilbyr

Unntak: Ingen

Use Case 6: Se egen reservasjon/lån historikk

Aktører: Bruker, Ansatt og Admin

Mål: Å få tilgang til informasjon om nåværende og tidligere lån og reserveringer

Prebetingelser: Brukeren er logget inn

Detaljert hendelsesforløp:

1. Brukeren går til lån historikken

2. Brukeren får se reserverasjoner og lån som enten er nåværende, leverte eller aldri hentet
3. Brukeren får se en liste med lån sortert etter når lånet begynte.
 - For alle lån som ble startet gjennom en reserverasjon skal det stå dagen reserverasjonen ble skapt.
 - Alle lån skal ha en leveringsdato, hentedato, utstyrstypene i lånet og antallet og størrelse valgt for de utstyrstypene.
 - Leverte lån skal vise datoen lånet faktisk ble levert

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelser: Bruker vet om alle tidligere lån

Use Case 7: Se utstyrstype informasjon

Aktører: Bruker, Ansatt og Admin

Mål: Se informasjon om en type utstyr

Prebetingelser: Use Case 5: Se inventar

Detaljert hendelsesforløp:

1. Brukeren klikker på en utstyrstype fra inventar skjermen
2. Brukeren får se mulige bilder, størrelser, og beskrivelse av utstyret

Alternativ flyt:

2a: Hvis det ikke er noe igjen av utstyret så vil det stå under beskrivelsen at det ikke er noen gjenstander av utstyrstypen igjen

Unntak: Ingen

Postbetingelser:

- Brukeren vet hva utstyrstypen er, hvilke størrelser utstyrsbanken har av den og hvor mange som er tilgjengelige for lån
-

Use Case 8: Utsett Lån

Aktører: Bruker, Ansatt og Admin

Mål: Forlenge lån hvis låner vil ha utstyret lengre

Prebetingelser: Bruker er logget inn

Detaljert hendelsesforløp:

1. Brukeren går til lån historikken
2. Brukeren finner lånet de vil utsette og trykker på knappen for å utsette
3. Lånet blir utsatt med en uke, eller til neste åpningsdag etter en uke

Alternativ flyt:

3a: Det er ingen åpningsdager innen en uke, og lånet blir utsatt til neste åpningsdag

Unntak: Ingen

Postbetingelser:

- Et lån har en ny innleveringsdato og innleveringen kan ikke lenger bli utsatt.
- Hendelsesloggen blir oppdatert

Use Case 9: Endre passord

Aktører: Besøker, Bruker, Ansatt og Admin

Mål: Å velge nytt passord

Prebetingelser:

- Aktøren har en registrert bruker

Detaljert hendelsesforløp:

1. Aktøren trykker på bruker ikonet på headeren og trykker så på "Endre passord"
2. Aktøren velger å få tilsendt SMS med kode for bekreftelse
3. Aktøren får kode med SMS
4. Aktøren skriver inn koden
5. Aktøren skriver inn nytt passord to ganger
6. Aktøren bekrefter endring og får vite at passordet har blitt oppdatert

Alternativ flyt:

- 1a. Aktøren velger "glemt passord"
- 5a. Brukeren skriver ikke inn samme passord begge ganger og får beskjed om feilen

Postbetingelser: Passordet har blitt oppdatert

Unntak:

6a: 6.a Aktøren avbryter oppdateringen og passordet blir uendret

Use case 10: Se påminnelser

Aktør: Bruker, Ansatt, Admin

Mål: Bruker blir påminnet om at de kan hente eller skal levere når fristen nærmer seg

Prebetingelser:

- Bruker er logget inn
- Bruker har påminnelser

Hendelsesforløp:

1. Bruker trykker på knappen for å se påminnelsene
2. Bruker kan se påminnelser om henting og levering av utstyr

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelser: Ingen

Use Case 11 Opprett Lån for registrert bruker :

Aktører: Ansatt og Admin

Mål:

- Å opprette et lån for andre brukere, slik at sluttbrukere kan låne utstyr uten å måtte logge seg inn
- Ansatt eller admin kan lage lån for brukere som overskrider grensene på maks antall utstyr per person

Prebetingelser: Logget inn som ansatt

Detaljert hendelsesforløp:

1. Ansatt velger utstyrstypen brukeren skal låne fra inventaret
2. Ansatt velger å opprette et lån med antallet og størrelsen brukeren vil ha på datoen gjesten vil levere på
3. Ansatt velger at lånet blir gjort av en bruker, og skriver så inn telefonnummeret til brukeren

4. Ansatt bekrefter lånet og får beskjed om at lånet ble lagt til

Alternativ flyt: Ingen

Unntak:

3a: Ansatt får beskjed om at det ikke er en bruker med det nummeret i systemet

4a: Ansatt får beskjed om at det ikke er mange nok av utstyrstypen på lager

Postbetingelser:

- Systemet har oppdatert antallet igjen av utstyret
- Systemet har lagret enda et lån
- Hendelsesloggen viser hvilken ansatt som gjennomførte lånet, utstyret som ble lånt, og hvilken bruker som lånte utstyret

Use case 12: Opprette Reservasjon for registrert bruker

Aktører: Ansatt og Admin

Mål:

- Å opprette en reservasjon for brukere slik at de fremdeles kan starte en reservasjon uten å måtte logge seg inn,
- At brukere kan opprette reservasjoner som egentlig overskrider grensen på maks antall utstyr per person

Prebetingelser: Logget inn som ansatt

Detaljert hendelsesforløp:

- Ansatt velger utstyrstypen brukeren skal reservere fra inventaret
- Ansatt velger å opprette en reservasjon med antallet og størrelsen brukeren vil ha på datoen gjesten vil hente og levere på
- Ansatt velger at reservasjonen blir gjort av en bruker, og skriver så inn telefonnummeret til brukeren
- Ansatt bekrefter lånet og får beskjed om at lånet ble lagt til

Alternativ flyt:

3a: Ansatt får beskjed om at det ikke er en bruker med det nummeret i systemet

4a: Ansatt får beskjed om at det ikke er mange nok av utstyrstypen på lager

Postbetingelser:

- Systemet har oppdatert antallet igjen av utstyret
 - Systemet har lagret enda et lån
 - Hendelsesloggen viser hvilken ansatt som gjennomførte reservasjonen, utstyret som ble reservert, og hvilken bruker som reserverer utstyret
-

Use Case 13: Opprette Lån for gjestebruker

Aktører: Ansatt og Admin

Mål: Å låne bort utstyr til en som ikke har en bruker

Prebetingelser: Logget inn som ansatt

Detaljert hendelsesforløp:

1. Ansatt velger utstyret gjesten skal låne fra inventaret
2. Ansatt velger å opprette et lån med antallet gjesten skal ha på datoen gjesten vil levere på
3. Ansatt velger at lånet blir gjort av en gjest, og skriver så inn telefonnummeret gjesten sier de vil ha lånet på, sammen med fornavn, etternavn, og også postnummer hvis gjesten vil
4. Ansatt bekrefter lånet og får beskjed om at lånet ble gjennomført

Alternativ flyt:

- 4a. Det er ikke er mange nok av utstyrstypen med den størrelsen på lager

Unntak:

- 4a: Ansatt avbryter lånet

Postbetingelser:

- Antallet av utstyr blir oppdatert
 - En reservasjon har blitt lagt til for en gjestebruker
 - Hendelsesloggen viser hvilken ansatt som gjennomførte lånet, utstyret som ble lånt, og navnet til gjestebrukeren
 -
-

Use case 14: Opprette Reservasjon for gjestebruker

Aktører: Ansatt og Admin

Mål: Å gi de som ikke har en bruker mulighet til å reservere utstyr

Prebetingelser: Logget inn som ansatt

Detaljert hendelsesforløp:

1. Ansatt velger utstyrstypen gjesten skal ha fra inventar siden
2. Ansatt velger størrelsen, antall, og datoer for innlevering og henting
3. Ansatt velger at reservasjonen er for en gjest
4. Ansatt velger at reservasjonen blir gjort av en gjest, og skriver så inn telefonnummeret gjesten sier de vil ha lånet på, sammen med fornavn, etternavn, og også postnummer hvis gjesten vil
5. Ansatt får beskjed om at reservasjonen ble gjennomført

Alternativ flyt:

- 5a. Ansatt får beskjed om at det ikke er mange nok av utstyrstypen på lager

Unntak:

- 4a: Ansatt avbryter reservasjonen

Postbetingelser:

- Systemet oppdaterer antallet av utstyret
- Systemet lagrer enda en reservasjon

Use Case 15: Se oversikt over reservasjoner og lån

Aktører: Ansatt og Admin

Mål: Se alle reservasjoner, nåværende lån, og ferdige lån

Prebetingelser:

- Bruker er logget inn som ansatt

Detaljert hendelsesforløp:

1. Lån oversikt knapp

2. Ser først alle reservasjoner som skal bli hentet og lånene som skal bli levert for førstkommande arbeidsdag

Alternativ flyt:

- 2a Ansatt filtrerer slik at de ser bare reservasjoner, bare nåværende lån eller bare ferdige lån
- 2b Ansatt filtrerer slik at de bare ser en bruker sin historikk
- 2c Ansatt filtrerer slik at kan se lån som skjedde før en gitt dato, etter en gitt dato eller i mellom to datoer

Unntak: Ingen

Postbetingelser: Ingen

Use Case 16: Bekreft henting

Aktører: Ansatt og Admin

Mål: Registrere et lån som hentet

Prebetingelser:

- En reservasjon har blitt opprette
- Logget inn som ansatt

Detaljert hendelsesforløp:

1. Den ansatte finner reservasjonen til låneren som henter utstyret fra oversikten over lån og reservasjoner og trykker på den
2. Den ansatte velger å bekrefte henting

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelser:

- En reservasjon har blitt gjort om til et lån
 - Hendelsen inkludert ansatt som bekreftet henting, låneren lånet tilhører, og utstyrstypen med antallet har blitt lagt til i hendelsesloggen
-

Use Case 17: Bekreft levering

Aktører: Ansatt og Admin

Mål: Se informasjon om en type utstyr

Prebetingelser:

- Et lån eksisterer
- Ansatt er logget inn

Detaljert hendelsesforløp:

1. Ansatt velger lånet til låneren som leverer utstyret fra oversikten over lån og reserverasjoner
2. Ansatt trykker på knappen for å bekrefte leveringen

Alternativ flyt:

- 1.

Unntak: Ingen

Postbetingelser:

- Et lån blir registrert som ferdig og antallet tilgjengelige av utstyrstypen øker
- Hendelsen inkludert ansatt som bekreftet leveringen, låneren lånet tilhørte, og utstyrstypen med antallet har blitt lagt til i hendelsesloggen

Use Case 18: Skriv kommentar om bruker

Aktører: Ansatt, Admin

Mål: Gi potensielle opplysninger til andre ansatte og admin om bruker

Prebetingelser: Logget inn som ansatt

Detaljert hendelsesforløp:

1. Ansatt trykker på brukeren fra brukeroversikten
2. Ansatt trykker på "skriv kommentar"
3. Ansatt skriver inn kommentaren i samme felt som andre kommentarer fra andre ansatte
4. Ansatt bekrefter oppdatering av kommentar

Alternativ flyt: Ingen

Unntak:

4a: Ansatt avbryter og kommentaren oppdateres ikke

Postbetingelser:

- En kommentar har blitt oppdatert
 - Hendelsesloggen viser hvilken ansatt som oppdaterte kommentaren, brukeren den tilhører, det gamle innholdet og det nye innholdet
-

Use Case 19: Se oversikt over hendelser i systemet

Aktører: Admin

Mål: Se oversikt over hendelser i systemet

Prebetingelser: Er logget inn som admin

Detaljert hendelsesforløp:

1. Fra admin menyen så trykker adminen på knappen for hendelse oversikten
2. Admin kan se en side med alle hendelsene med nyeste hendelser øverst
3. Admin kan se typen hendelse, tidspunkt, dato, beskrivelse og brukere involvert i en hver hendelse
4. Hendelser i loggen er generelt relatert til endringer i utstyr, reserverasjoner og lån

Alternativ flyt:

1. Admin kan velge å legge til filtre som
 - hendelser som skjedde på en gitt dato, før en dato, etter en dato eller i mellom to datoer.
 - hendelser gjennomført av en spesifikk bruker (både ansatte og vanlige brukere)
 - type hendelse

Unntak: Ingen

Postbetingelser: Ingen

Use Case 20: Legg til ny utstyrstype

Aktører: Admin

Mål: Gjøre en ny type utstyr tilgjengelig for lån

Prebetingelse: Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger å legge til ny utstyrstype fra inventar siden
2. Admin velger navn på utstyret, bildet som blir vist på inventar siden, beskrivelse, størrelser og antallet av størrelsene på lager
3. Admin får bekreftelse om at ny utstyrstype blir lagt til

Alternativ flyt:

- 2a Admin velger å legge til et eller flere bilder for utstyrssiden
- 3a Det eksisterer allerede en utstyrstype med samme navn

Unntak: Ingen

Postbetingelser: En ny utstyrstype har blitt lagt til og kan bli lånt av brukere

Use Case 21: Slett utstyrstype

Aktør: Admin

Mål: Å fjerne en utstyrstype fra oversikten over utstyrstyper

Prebetingelser:

- Det eksisterer en utstyrstype
- Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger utstyrstype
2. Admin trykker på knappen for å slette
3. Admin bekrefter sletting

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelse: Utstyrstypen kan fremdeles bli sett i lån historikken, statistikken og i hendelsesloggen, men kan ikke bli valgt for reserverasjoner og lån

Use Case 22: Legg til ny størrelse

Aktør: Admin

Mål: Legge til nye størrelser for eksisterende utstyrstyper

Prebetingelser:

- Det finnes en utstyrstype
- Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger hvilket utstyr som skal bli modifisert
2. Admin skriver inn ny størrelse og kan velge antallet som er på lager
3. Admin bekrefter størrelsen og kan se at den har blitt lagt til

Alternativ flyt:

- 2a Admin får en feilmelding hvis størrelsen allerede eksisterer
- 2b Hvis admin ikke velger et antall blir det automatisk 0

Unntak:

- 3a: Admin avbryter modifiseringen av utstyrstype

Postbetingelser: En ny størrelse har blitt lagt til

Use Case 23: Slett størrelse

Aktør: Admin

Mål: Fjerne en størrelse fra systemet

Prebetingelser:

- Minst en utstyrstype med minst en størrelse eksisterer i systemet
- Admin er i prosessen av å modifisere utstyrstypen

Detaljert hendelsesforløp:

1. Admin velger utstyrstypen som skal bli modifisert
2. Admin velger hvilke størrelser som skal bli slettet

3. Admin bekrefter modifisering av utstyrstype

Alternativ flyt: Ingen

Unntak:

3a Admin avbryter modifisering og ingen st rrelse blir fjernet

Postbetingelser: St rrelsen har blitt fjernet, men kan fremdeles bli sett i l n historikken og statistikken

Use Case 24: Se statistikk for utstyr bruk

Akt r: Admin

M l: Finne ut hvor populære de ulike typene utstyr er

Prebetingelser: Logget inn som admin

Detaljert hendelsesforl p:

1. Admin navigerer til statistikk i admin menyen
2. Admin velger utstyr eller kategori de vil se statistikk for
3. Admin kan se informasjon for hvor mange av utstyret eller kategorien som ble l nt for en valgt m ned, sesong eller en angitt dato intervall

Alternativ flyt:

3a Admin filtrerer bruk basert p  postnummer

Unntak: Ingen

Postbetingelser: Ingen

Use Case 25: Legg til utstyrstype bilde

Akt r: Admin

M l: Legge til bilder som informerer kunder om hvordan et eksemplar av en utstyrstype ser ut

Prebetingelser:

- Det eksisterer en utstyrstype
- Admin er i prosessen av   modifisere utstyrstypen
- Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger å legge til bilder for utstyrstypen
2. Admin får se grenser på bilde input som størrelse og filtype
3. Admin velger fil fra deres enhet og hvis den passer begrensningene så legges den til
4. Admin bekrefter modifikasjoner til utstyrstype når de ikke vil legge til flere bilder

Alternativ flyt:

3a Hvis bildefilen ikke passer innenfor begrensningene får admin en feilmelding og må prøve igjen

Postbetingelser:

- En eller flere bilder har blitt lagt til i systemet for en utstyrstype

Unntak:

4a: Admin avbryter redigeringen og ingen nye bilder blir lagret

Use Case 26: Fjern utstyrstype bilde

Aktør: Admin

Mål: Slette et utstyrstype bilde fra systemet

Prebetingelser:

- En utstyrstype har blitt lagt til
- Admin er logget inn
- Admin er i prosessen av å modifisere utstyrstypen

Detaljert hendelsesforløp:

1. Admin velger "Slett bilde"
2. Admin velger bildet som skal bli slettet
3. Admin får vite at bildet er slettet

Alternativ flyt: Ingen

Postbetingelser: Bildet har blitt slettet fra systemet

Unntak:

2a: Admin avbryter og ingen bilder blir slettet

Use Case 27: Velg bilderekkefølge

Aktør: Admin

Mål: Endre rekkefølgen bildene vises i på siden for en utstyrstype Prebetingelser:

- Det finnes minst en utstyrstype med minst 2 bilder
- Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger hvilket utstyr bilde rekkefølgen skal endres for
2. Admin velger rekkefølge
3. Admin får vite at rekkefølgen ble lagret

Alternativ flyt:

2a Admin velger å gå tilbake og rekkefølgen blir ikke lagret

Postbetingelser:

- Bildene sin rekkefølge i databasen blir oppdatert
 - Databasen oppdateres med rekkefølgen på bildene
-

Use Case 28: Oppdater utstyrstype antall

Aktør: Admin

Mål: Øke antallet hvis nytt utstyr ble lagt til i lageret eller trekke fra antall hvis utstyr går tapt eller fjernes

Prebetingelser:

- Minst en utstyrstype har blitt lagt til
- Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger hvilket utstyr antallet skal bli oppdatert for

2. Admin skriver inn mengden antallet økes eller minskes med

3. Admin bekrefter modifikasjoner gjort

Alternativ flyt: Ingen

Postbetingelser: Antallet av utstyret som er på lageret har blitt oppdatert

Unntak:

3a: Admin avbryter modifikasjonen

Use Case 29: Oppdater lånegrense for utstyr

Aktør: Admin

Mål: Hindre en låner fra å reservere all tilgjengelig utstyr

Prebetingelser: Ingen

Detaljert hendelsesforløp:

1. Admin velger utstyret som skal få en oppdatert lånegrense
2. Admin velger max grensen
3. Admin bekrefter valget

Unntak: Ingen

Postbetingelser: Fremtidige reserverasjoner har nå en ny antall max grense

Use Case 30: Sett åpningstider

Aktør: Admin

Mål: Velge dager utstyrsbanken vanligvis er åpen

Prebetingelser: Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger å redigere åpningstider
2. Admin kan se nåværende åpningstider for hver dag
3. Admin velger å markere ukedager som skal være stengt
4. Admin velger tidspunkt på start og tidspunkt på slutt
5. Admin bekrefter endringene

Alternativ flyt: Ingen

Postbetingelser:

- Dersom admin la til en ny ukedag så kan den bli valgt for henting og levering av utstyr når bruker eller ansatt oppretter nye lån og reservasjoner
- Informasjonssiden viser de oppdaterte dagene og tidene
- Dersom admin fjernet en ukedag skal den ikke kunne bli valgt for henting eller levering av nye lån og reservasjoner
- Hendelsesloggen viser alle endringer gjort

Unntak:

5a: Admin avbryter og ingen endringer blir lagret

Use Case 31: Velge unntak til åpningstider

Aktør: Admin

Mål:

- Velge datoer utstyrsbanken skal holde stengt som vanligvis er åpne
- Velge datoer utstyrsbanken skal holde åpen som vanligvis er stengt

Prebetingelser: Logget inn som admin

Detaljert hendelsesforløp:

1. Admin velger redigering av åpningsdager fra admin menyen
2. Admin ser en liste med datoer som har blitt valgt som stengt eller åpen.
3. Admin velger datoer som skal bli holdt åpent og åpningstidene for de dagene eller datoer som skal bli holdt stengt
4. Admin bekrefter endringer

Alternativ flyt: Ingen

Postbetingelser:

- Dato valgt som stengt skal ikke kunne bli valgt
- Valget av å gjøre en dato som vanligvis er åpen om til en stengt dag skal kunne bli angret når som helst
- Valget av å gjøre en dato som vanligvis er stengt om til en åpen dag skal kunne bli reversert om det ikke er en lån

Unntak:

5a: Admin avbryter endring

Kommentar: Valget av unntak tar prioritet over ukentlige åpningstider. Også hvis de ukentlige åpnings-tidene blir endret vil det ikke påvirke valgte unntak.

Use Case 32: Lag ansatt

Aktør: Admin

Mål: Å lage en ny ansatt i utstyrsbanken som skal bekrefte at lån blir hentet og levert på den fysiske lokasjonen

Prebetingelser:

- Logget inn som admin
- Det finnes en vanlig bruker i systemet

Detaljert hendelsesforløp:

1. Admin trykker på brukeren fra bruker oversikten
2. Admin trykker på knappen for å forfremme brukeren til ansatt
3. Admin bekrefter at de er sikre på at brukeren skal bli forfremmet

Alternativ flyt: Ingen

Postbetingelser:

- En bruker har blitt gjort om til en ansatt
- Hendelsesloggen viser hvilken bruker som ble om til en ansatt

Unntak:

3a: Admin avbryter prosessen og brukeren er fremdeles en vanlig bruker

Use Case 33: Slett bruker

Aktør: Admin

Mål: Fjerne brukere fra systemet

Prebetingelser:

- Er logget inn som Admin
- Det finnes en bruker

Detaljert hendelsesforløp:

1. Admin går til brukeroversikten
2. Admin finner brukeren som skal bli slettet
3. Admin trykker på knappen for å slette brukeren.
4. Brukeren blir slettet

Alternativ flyt: Ingen

Unntak:

3a: Brukeren er admin sin bruker og kan ikke bli slettet

Postbetingelser:

1. En bruker har blitt slettet fra systemet
2. Lån gjort av den brukeren er fremdeles i historikken, men viser ikke navn

Use case 34: Se brukeroversikt

Aktører: Admin og ansatt

Mål: Se brukere og informasjon koblet til dem som lån og kommentarer

Prebetingelse: Logget inn som ansatt

1. Ansatt trykker på knappen for å gå til brukeroversikten
2. Ansatt ser listen over brukere

Unntak: Ingen

Postbetingelse: Ingen

Use case 35 Rediger utstyrstype navn

Aktører: Admin

Mål: Å endre navn på en utstyrstype

Prebetingelser: Admin er logget inn

Detaljert hendelsesforløp:

1. Admin velger utstyrstypen som skal bli modifisert
2. Admin velger å redigere navnet
3. Admin skriver inn nytt navn
4. Admin bekrefter nytt navn

Alternativ flyt:

3a Utstyrstype navnet eksisterer allerede og admin må velge på nytt

Postbetingelser:

- Utstyrstypen sitt nye navn blir vist i historikken i alle lån og reserverasjoner av typen, statistikken og inventaret
- Det gamle navnet lagres i hendelsesloggen

Unntak:

4a: Admin avbryter redigering av utstyrstype og navnet endres ikke

Use Case 36: Rediger Beskrivelse Aktører: Admin

Mål: Å endre på den tekstlige beskrivelsen av en utstyrstype

Prebetingelser:

- Admin er logget inn
- Admin er på siden for en utstyrstype

Detaljert hendelsesforløp:

1. Admin velger å redigere beskrivelsen
2. Admin skriver den nye beskrivelsen eller endrer på den eksisterende
3. Admin får bekreftelse om at beskrivelsen har blitt oppdatert

Alternativ flyt: Ingen

Unntak:

4a: Admin avbryter midt i redigering og tidligere beskrivelse forblir

Postbetingelser:

- Beskrivelsen har blitt oppdatert
 - Hendelsesloggen har blitt oppdatert med navnet på utstyr, den nye tekst beskrivelsen
-

Use case 37: Redigere forside

Aktører: Admin

Mål: Å endre på teksten på forsiden som informerer brukere om viktig informasjon

Prebetingelser:

- Admin er logget inn
- Admin har navigert fram til funksjonen for å redigere forside

Detaljert hendelsesforløp:

1. Admin skriver inn ny tekst
2. Admin bekrefter endring
3. Systemet erstatter den forrige teksten med den nye

Alternativ flyt: Ingen

Postbetingelser: Forsiden har blitt oppdatert

Unntak:

4a: Admin avbryter midt i redigering og tidligere forside forblir

Use case 38: Endre rekkefølge på størrelse til en utstyrstype

Aktør: Admin

Mål: Å manuelt endre på rekkefølgen

Prebetingelser:

- Admin er logget inn
- Admin har navigert fram til funksjonen for å endre rekkefølgen på utstyrstyper

Detaljert hendelsesforløp:

1. Admin flytter en størrelse opp eller ned

2. Admin bekrefter endring i rekkefølge
3. Systemet håndterer endringen og bekrefter til admin at rekkefølgen er endret

Alternativ flyt: Ingen

Postbetingelser: Posisjonen til størrelsene som ble flyttet på oppdateres

Unntak:

- 2a: Admin avbryter og rekkefølgen endres ikke
-

Use case 39: Endre rekkefølge på utstyr på inventar siden

Aktør: Admin

Mål: Å manuelt endre på rekkefølgen på utstyr som vanligvis er i rekkefølgen av når de ble lagt til i systemet

Prebetingelser:

- Admin er logget inn
- Admin har navigert fram til funksjonen for å endre rekkefølge

Detaljert hendelsesforløp:

1. Admin velger å forskyve en utstyrstype eller å putte det lenger fram i listen over utstyr.
2. Når alle utstyrstypene er i posisjonen adminen er fornøyd med bekrefter de endringen

Alternativ flyt: Ingen

Postbetingelser: Posisjonen til utstyret som ble flyttet på har blitt oppdatert

Unntak:

- 2a: Admin avbryter og rekkefølgen endres ikke
-

Use case 40: Kansellere egen reservasjon

Aktør: Bruker, ansatt, admin

Mål: Å kansellere reservasjonen

Prebetingelser:

- Bruker har en reservasjon

- Bruker er logget inn

Detaljert hendelsesforløp:

1. Bruker navigerer til egne lån
2. Brukeren finner fram lånet som skal bli kansellert
3. Brukeren kansellerer lånet

Alternativ flyt: Ingen

Postbetingelser:

- Reservasjonen er fremdeles lagret i historikken, og det blir vist at den ikke ble hentet
- Hendelsesloggen viser reservasjonen som ble kansellert med brukernavn, utstyrstype og antall
- Antallet tilgjengelige av utstyrstypen øker

Unntak: Ingen

Use case 41: Modifiser utstyrstype

Aktør: Admin

Mål: Å oppdatere data knyttet til en spesifikk utstyrstype

Prebetingelser:

- Logget inn som admin
- Det finnes allerede en utstyrstype

Detaljert hendelsesforløp:

1. Admin velger en utstyrstype fra inventaret
2. Admin starter redigeringen ved å trykke på en knapp
3. Admin gjør endringer som sletting, legge til, endre rekkefølge på data som bilder, størrelser, beskrivelse, kategorier og navn
4. Admin bekrefter endringer og kan se den oppdaterte informasjonen for utstyret

Alternativ flyt: Ingen

Postbetingelser:

- Systemet har lagret nytt, slettet gammelt eller modifisert data.

- Hendelsesloggen viser gammel data hvis det har blitt slettet eller modifisert, i tillegg til den nye dataen

Unntak:

4a: Admin avbryter prosessen og utstyrstypen går tilbake til slik den var i starten av use caset

Use case 42: Endre fornavn og etternavn

Aktør: Bruker, Ansatt og Admin

Mål: Å oppdatere navnet knyttet til brukeren

Prebetingelser: Logget inn

Detaljert hendelsesforløp:

1. Brukeren trykker på knappen for å gå til brukerkonto siden
2. Brukeren velger å redigere navnet
3. Brukeren skriver inn nytt navn
4. Brukeren bekrefter endringen

Alternativ flyt: Ingen

Postbetingelser:

- Systemet har oppdatert navnet
- Alle reservasjoner og lån gjort av brukeren viser det nye navnet
- Hendelsesloggen viser det gamle navnet og det nye navnet

Unntak:

4a: Brukeren avbrøt endringen og det gamle navnet blir

Use case 43: Endre postnummer

Aktør: Bruker, Ansatt, Admin

Mål: Å korrigere eller oppdatere postnummer

Prebetingelser: Logget inn

Detaljert hendelsesforløp:

1. Brukeren trykker på knappen for å gå til brukerkonto siden
2. Brukeren velger å redigere postnummeret
3. Brukeren skriver inn nytt postnummer
4. Brukeren bekrefter endringen

Alternativ flyt: Ingen

Postbetingelser:

- Systemet har oppdatert postnummeret
- Lån som tilhørte det gamle postnummeret i statistikken tilhører nå det nye postnummeret

Unntak:

4a; Brukeren avbrøt endringen og det gamle postnummeret blir

Use case 44: Endre telefonnummer

Aktør: Bruker, Ansatt, Admin

Mål: Å oppdatere telefonnummer

Prebetingelser: Logget inn

Detaljert hendelsesforløp:

1. Brukeren går til brukerkonto siden
2. Brukeren velger å redigere telefonnummeret
3. Brukeren skriver inn nytt telefonnummer
4. Brukeren velger å få tilsendt bekreftelse SMS
5. Brukeren skriver inn koden fra SMS
6. Hvis koden var riktig får bruker vite at de nå er registrerte på et nytt nummer

Alternativ flyt: 7.a) En bruker med det nummeret eksisterer allerede

Postbetingelser: Systemet har oppdatert telefonnummeret

Unntak: Brukeren avbrøt endringen og det gamle telefonnummeret blir

Use case 45: Kansellere bruker reservasjon

Aktør: Ansatt, Admin

Mål: Å kansellere en reservasjon

Prebetingelser: Logget inn som ansatt

Detaljert hendelsesforløp:

1. Ansatt finner reservasjonen i oversikten over reservasjoner
2. Ansatt sletter reservasjonen

Alternativ flyt: Ingen

Postbetingelser:

- Reservasjonen er fremdeles lagret i historikken, og det blir vist at den ikke ble hentet
- Hendelsesloggen viser hvilken ansatt som kansellerte, innholdet i reservasjonen, og hvilken bruker den tilhørte
- Antallet tilgjengelige av utstyrstypen øker

Unntak: Ingen

Use case 46: Endre egen informasjon

Aktør: Bruker, Ansatt, Admin

Mål: Å redigere egen informasjon

Prebetingelser: Logget inn

Detaljert hendelsesforløp:

1. Bruker navigerer fram til brukerprofilen
2. Bruker velger hvilken type informasjon de vil redigere av passord, telefonnummer eller navn, etternavn og postnummer

Alternativ flyt: Ingen

Postbetingelser: Brukerinformasjon har blitt oppdatert

Unntak: Ingen

Use case 47: Velge spesifikk dato for stenging

Aktør: Admin

Mål: Å endre åpningstider, åpne eller stenge på en spesifikk dag

Prebetingelser:

- Logget inn som admin
- Admin er på siden for å endre unntak for datoer

Detaljert hendelsesforløp:

1. Admin velger en dato
2. Datoen blir lagt til i en liste med unntaksdager
3. Admin velger åpningstider for dagen, eller om den skal være stengt
4. Admin bekrefter og lagrer nye åpningstider

Alternativ flyt: Ingen

Postbetingelser: Hvis datoen hadde henting og levering av lån og reservasjoner blir disse flyttet til en valid dato automatisk

Unntak: Ingen

Use Case 49: Legge til kategori

Aktør: Admin

Mål: Lage en ny kategori for utstyrstyper

Prebetingelser: Ingen

Detaljert hendelsesforløp:

1. Admin navigerer fram til menyen for kategori redigering
2. Admin trykker på knappen for å legge til en ny kategori
3. Admin skriver inn navnet på kategorien
4. Admin bekrefter og kategorien blir lagt til

Alternativ flyt: Ingen

Postbetingelser: En ny kategori uten kobling til noen utstyrstyper har blitt lagt til

Unntak: Ingen

Use Case 50: Fjern utstyrstype kategori

Aktør: Admin

Mål: Fjerne kategori

Prebetingelser: Det finnes minst en kategori

Detaljert hendelsesforløp:

1. Admin navigerer fram til menyen for kategori redigering
2. Admin trykker på sletteknappen for kategorien som skal bli slettet

Alternativ flyt: Ingen

Postbetingelser: En kategori har blitt slettet

Unntak: Ingen

Use Case 51: Redigere kategorier for utstyrstype

Aktør: Admin

Mål: Velge hvilke kategorier et utstyret tilhører

Prebetingelser:

- Det finnes et utstyr
- Det finnes en kategori

Detaljert hendelsesforløp:

1. Admin velger utstyret som skal bli redigert
2. Admin velger fra listen av kategorier hvilke de vil legge til, og kan også fjerne tidligere valgte kategorier
3. Admin lagrer endringer av utstyrstype

Alternativ flyt: Ingen

Postbetingelser: Utstyr har blitt lagt til eller fjernet fra kategori

Unntak: Ingen

Use Case 52: Rediger grense på reservasjon lengde

Aktør: Admin

Mål: Endre på grense på antallet dager en reservasjon kan vare når bruker velger når de vil hente reservasjonen

Prebetingelser: Administrator har funnet konfigurasjon innstillinger

Detaljert hendelsesforløp:

- Administrator velger fra listen av kategorier hvilke de vil legge til, og kan også fjerne tidligere valgte kategorier

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelser: Grensen på reservasjonslengde har blitt oppdatert

Use Case 53: Rediger grense på lån lengde

Aktør: Admin

Mål: Endre på grense på antallet dager et lån

Prebetingelser: Administrator har funnet konfigurasjon innstillinger

Detaljert hendelsesforløp:

- Administrator velger fra listen av kategorier hvilke de vil legge til, og kan også fjerne tidligere valgte kategorier

Alternativ flyt: Ingen

Unntak: Ingen

Postbetingelser: Max lengde på lån har blitt oppdatert

D Guide for Brukbarhetstest

D.1 Testing

Testing er fundamentalt til smidig utvikling. Funksjonene vi utvikler må bli testet slik at vi enten kan bestemme oss for at den er ferdig, eller iterere på den ved å endre designet av den ved å analysere dataen fra testene.

Vi burde teste tidlig og ofte for å spare tid og ressurser.

D.2 Hvorfor brukbarhetstest

Brukbarhetstester kan også bli kalt brukertester. På engelsk blir de kalt usability tests, og International Organization for Standardization definerer dem som « Extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. ».

Brukbarhetstester kan hjelpe oss å finne dårlig design i interaksjonsdesignet som vi kan forbedre til videre iterasjoner. Hvis testeren er forvirret om en handling ble begått er nok problemet tilbakemeldingen brukeren får, og hvis testeren sliter med å finne en funksjon så kan det være et problem med synligheten til funksjonen.

Målet med utstyrsbanken er å sørge for at brukerne har en så god opplevelse som mulig

Et alternativ til brukbarhetstester er å gjennomgå prototypen med en ekspert, for eksempel en foreleser innenfor interaksjonsdesign på NTNU Gjøvik. Fordelen med brukbarhetstesting er at vi får se selv hva problemene er gjennom erfaring i stedet for å måtte stole på at en ekspert sier hva som kan skje.

D.3 Hva vi ønsker å måle

- Hvor mange av oppgavene som blir fullført suksessfullt og hvor mange feil som oppstår
- Hvor lang tid testeren bruker på oppgavene
- Hvor tilfredsstilt testeren er med opplevelsen

D.4 Hvem vi ønsker skal teste

Det hadde vært fint å få tilbakemelding fra et bredt spektrum med forskjellige aldere, fysiske og psykiske evner, og nivåer av erfaring med lignende sider for å få mest innsikt.

De kan være folk vi kjenner, men vi må være klare over at de kan være mindre objektive enn fremmede.

Mellom 5 og 10 forskjellige personer som tester skal være bra nok for kvalitativ datainnsamling som dette. Mer enn det blir lite effektivt fordi det kan bli mange gjentakelser. Så lenge en person har et problem så vet vi om problemet.

D.5 Test data

Det burde være nok testdata i databasen til at testerne må lete etter det oppgaven spør etter.

D.6 Introduksjon

Forklar hvem vi er, og hva oppgaven går ut på

Fortell dem at de burde tenke høyt og at de ikke kan få assistanse

Spør om de har erfaring med lignende systemer

Få samtykke fra til å dele resultater og svar på spørsmål med arbeidsgiver og de som leser bacheloroppgaven

Be testeren å være helt ærlig

D.7 Hoveddel

Ta tiden for å måle effektiviteten

Noter frustrasjonsmomenter om de oppstår. Frustrasjonsmomenter kan være testeren gjør en handling som ikke bidrar til å nå målet til oppgaven, eller bruker lang tid.

En oppgave om gangen

Første gang en tester gjør en oppgave burde vi være oppmerksomme på hvor lett det er å lære, og på videre tester med samme person burde vi være oppmerksomme på hvor lett nettsiden er å huske.

D.8 Avslutning

Spør testeren hvilken score de vil gi fra 1-5 om helhetsopplevelsen

- Spør testeren om de følte noe frustrasjon eller forvirring
- Spør testeren hva de syntes om hastigheten på lastingen
- Spør testeren om de har forslag til endringer
- Spør testeren om hva de synes om estetikken
- Spør testeren om det er noen funksjoner de savner

D.9 Eksempler på Oppgaver

Vanlig bruker:

1. Lag en bruker. (Med sitt eget telefonnummer eller et gitt et)?
2. Reserver langrennski med størrelse 160 centimeter og skistaver til 120 centimeter fra neste tirsdag til neste torsdag
3. Kanseller reservasjonene du nettopp lagde
4. Logg ut
5. Logg inn på en gitt bruker

6. Finn påminnelsene om at utstyr er klart for henting

7. Utsett lån

Ansatt:

1. Opprett et lån for registrert bruker telefon nummer 48604720 fra neste tirsdag til neste torsdag

2. Registrer henting av lånet av skistaver gjort av bruker Ola Nordmann

Administrator

1. Sett onsdag som en åpningsdag med åpningstid 10:00 og stengt 18:00.

2. Finn ut hvilken ansatt det var som bekreftet henting av lånet av en fotball til Kari Nordmann

D.10 Iterasjoner

Vi burde ha klart adskilte versjoner som vi tester. Fordelen med dette er at vi kan gjøre separate tester og diskutere sammen hva vi har funnet og så komme fram til endringer vi burde gjøre sammen. En iterasjon burde være maks 2 uker, samme som lengden av en sprint.

D.11 Analyse i ettertid

Vi burde alle diskutere i ettertid endringer som burde bli gjort til neste iterasjon av tester, og dokumentere endringene som blir gjort basert på både tilbakemeldinger og testresultater. Problemene burde bli rangert i alvorlighet basert på hvor mange testere hadde problemet, og hvor mye effektiviteten til testerne ble påvirket av problemet.

Kilder for videre læring

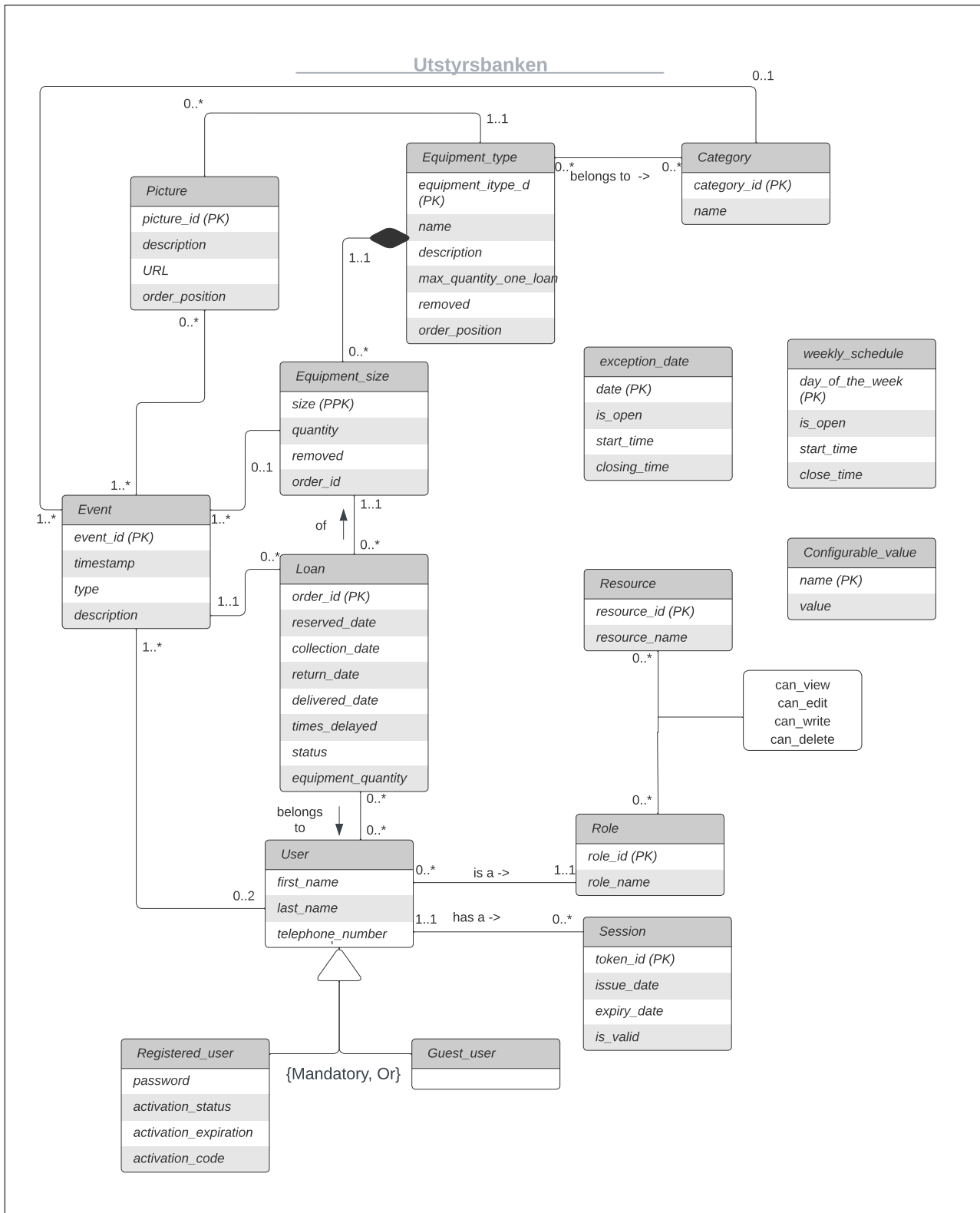
- <https://folk.ntnu.no/baldurk/skolearbeid/MMI/12%20-%20Brukbarhetstesting.html>
- <https://quizlet.com/300093330/tdt4180-punkter-for-utforing-av-brukbarhetstest-flash-cards/>
- <https://no.wikipedia.org/wiki/Brukertesting>
- <https://folk.ntnu.no/baldurk/skolearbeid/MMI/Forelesninger%20MMI/70-Brukbarhetstesting.pdf>
- <https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>
- <https://www.hotjar.com/usability-testing/evaluation-analysis/>

<https://www.glueglue.com/en/blog/expert-review-vs-usability-testing>

E ER-diagram

En kopi av det fulle ER-diagrammet er inkludert på neste side. Et [Entity Relations diagram \(ER-Diagram\)](#) har som hensikt i å beskrive sammenhengen mellom datastrukturene. Deler/seksjoner av dette diagrammet har blitt brukt ekstensivt i kapittel 5.8.

Begynner på neste side

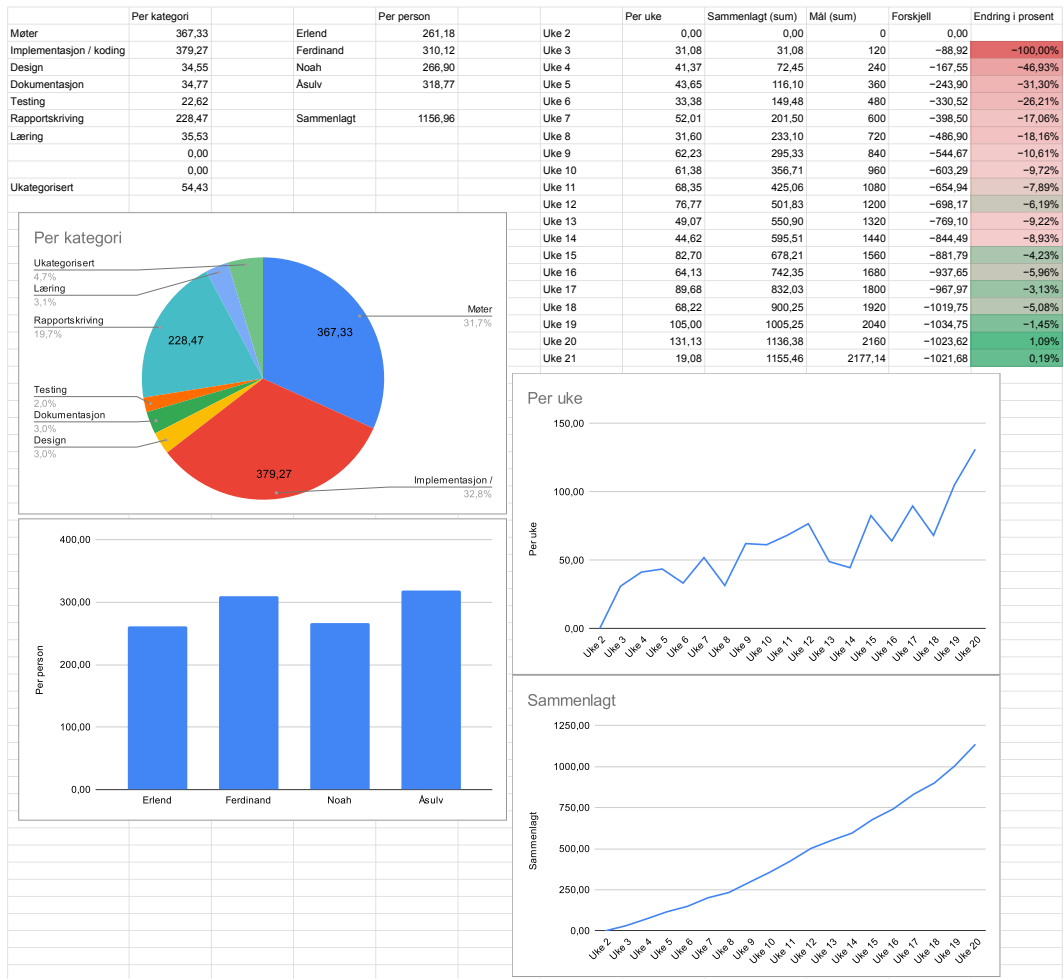


F Detaljert tidsbruk

Disse dataene viser tidsbruk per gruppemedlem, samt per uke i perioden 15/01/2024-20/05/2024

Data hentet 20/05/2024 kl. 17

Begynner på neste side



F.1 Per gruppemedlem

F.1.1 Erlend

Data hentet 20/05/2024 kl. 17

Begynner på neste side

F.1.2 Noah

Data hentet 20/05/2024 kl. 17

Begynner på neste side

Dato	Fra	Til	Timer	Kategori	Beskrivelse	Timebruk
			0,00	Ukategorisert		87,00
			0,00	Ukategorisert		95,17
15/01/2024	kl. 12.00	kl. 14.00	2,00	Møter		11,90
16/01/2024	kl. 12.00	kl. 14.23	2,38	Møter		6,95
18/01/2024	kl. 13.30	kl. 14.00	0,50	Møter		20,45
18/01/2024	kl. 14.00	kl. 15.52	1,87	Møter		37,72
22/01/2024	kl. 13.15	kl. 16.30	3,25	Møter		6,63
24/01/2024	kl. 15.15	kl. 17.17	2,03	Møter		0,00
25/01/2024	kl. 13.35	kl. 14.20	0,75	Møter		0,00
25/01/2024	kl. 15.30	kl. 17.30	2,00	Møter		1,08
26/01/2024	kl. 13.00	kl. 15.10	2,17	Møter		
27/01/2024	kl. 14.00	kl. 16.00	2,00	Dokumentasjon	Gantt diagram	266,90
29/01/2024	kl. 20.00	kl. 23.00	3,00	Dokumentasjon	Gantt diagram	
30/01/2024	kl. 10.00	kl. 14.00	4,00	Møter		
01/02/2024	kl. 13.30	kl. 15.00	1,50	Møter		
02/02/2024	kl. 13.30	kl. 16.22	2,87	Møter		
03/02/2024	kl. 14.30	kl. 15.25	0,92	Implementasjon / koding	Sett opp frontend og backend VM	
05/02/2024	kl. 10.15	kl. 11.43	1,47	Møter		
07/02/2024	kl. 19.00	kl. 19.30	0,50	Testing		
08/02/2024	kl. 12.00	kl. 16.00	4,00	Møter		
09/02/2024	kl. 12.00	kl. 15.00	3,00	Møter		
11/02/2024	kl. 14.00	kl. 15.00	1,00	Læring	Se på CI/CD	
12/02/2024	kl. 10.30	kl. 12.00	1,50	Møter	Sprint review/planning	
13/02/2024	kl. 10.30	kl. 12.30	2,00	Implementasjon / koding	Skriv Dockerfile for backend	
14/02/2024	kl. 02.30	kl. 02.50	0,33	Implementasjon / koding	Fiks Dockerfil	
14/02/2024	kl. 19.10	kl. 19.45	0,58	Implementasjon / koding	Legg til pipeline	
15/02/2024	kl. 12.00	kl. 16.00	4,00	Møter		
15/02/2024	kl. 16.15	kl. 17.41	1,43	Implementasjon / koding	Fix pipeline	
16/02/2024	kl. 12.00	kl. 14.15	2,25	Implementasjon / koding		
16/02/2024	kl. 14.25	kl. 16.20	1,92	Implementasjon / koding		
19/02/2024	kl. 11.15	kl. 14.26	3,18	Møter	Ferdigjør pipeline	
19/02/2024	kl. 18.24	kl. 18.44	0,33	Læring	Se over API utkast	
20/02/2024	kl. 15.30	kl. 16.00	0,50	Implementasjon / koding	Merge pipeline	
21/02/2024	kl. 10.05	kl. 11.00	0,92	Møter	Diskuter database struktur	
21/02/2024	kl. 11.00	kl. 12.00	1,00	Implementasjon / koding	Fiks bygging av frontend	
22/02/2024	kl. 11.24	kl. 12.50	1,43	Implementasjon / koding	Begynn frontend pipeline	
22/02/2024	kl. 13.30	kl. 14.05	0,58	Møter		
22/02/2024	kl. 14.20	kl. 16.18	1,97	Implementasjon / koding		
24/02/2024	kl. 23.20	kl. 23.58	0,63	Implementasjon / koding	Ferdigstill pipeline frontend	
26/02/2024	kl. 10.15	kl. 12.00	1,75	Møter		
26/02/2024	kl. 12.00	kl. 13.30	1,50	Implementasjon / koding	Begynn deployment database	
28/02/2024	kl. 13.30	kl. 15.00	1,50	Møter		
29/02/2024	kl. 13.27	kl. 17.10	3,72	Implementasjon / koding	Ferdigjør database deployment	
01/03/2024	kl. 13.27	kl. 15.50	2,38	Møter	MR reviews	
04/03/2024	kl. 10.30	kl. 14.20	3,83	Implementasjon / koding		
07/03/2024	kl. 13.30	kl. 14.20	0,83	Møter		
07/03/2024	kl. 14.20	kl. 23.35	9,25	Implementasjon / koding	Frontend innlogging	
07/03/2024	kl. 23.35	kl. 23.59	0,42	Læring	Konfigurerer av frontend	
08/03/2024	kl. 00.00	kl. 01.30	1,50	Implementasjon / koding	fix frontend server (nginx)	
08/03/2024	kl. 12.00	kl. 13.00	1,00	Møter	Møte med oppdragsgiver (usecase)	
08/03/2024	kl. 13.00	kl. 15.45	2,75	Implementasjon / koding	Cleanup Login frontend	
10/03/2024	kl. 10.00	kl. 12.00	2,00	Møter	Sprint planlegging	
10/03/2024	kl. 12.00	kl. 13.00	1,00	Implementasjon / koding	Refaktorer DB Schema	
11/03/2024	kl. 10.25	kl. 12.50	2,42	Møter	Sprint Review / Planlegging	
11/03/2024	kl. 17.50	kl. 19.10	1,33	Testing	MR review	
12/03/2024	kl. 20.17	kl. 21.10	0,88	Læring	Backend og SQL	
13/03/2024	kl. 10.10	kl. 13.10	3,00	Implementasjon / koding	Refaktorer bruker-API backend	
14/03/2024	kl. 13.30	kl. 14.05	0,58	Møter	Veileder	
14/03/2024	kl. 14.15	kl. 18.07	3,87	Implementasjon / koding	Frontend	
14/03/2024	kl. 21.10	kl. 21.30	0,33	Testing	Frontend MR Review	
15/03/2024	kl. 13.00	kl. 18.50	5,83	Implementasjon / koding		
18/03/2024	kl. 04.00	kl. 04.10	0,17	Implementasjon / koding		
18/03/2024	kl. 10.15	kl. 11.51	1,60	Møter	Planlegg uken	
18/03/2024	kl. 11.51	kl. 13.10	1,32	Testing	MR Review Backend Reservasjoner	
18/03/2024	kl. 13.10	kl. 14.10	1,00	Implementasjon / koding	Frontend	
18/03/2024	kl. 14.30	kl. 16.05	1,58	Implementasjon / koding	Frontend data strukturer	
18/03/2024	kl. 16.42	kl. 16.46	0,07	Implementasjon / koding	Fix MR	
20/03/2024	kl. 22.33	kl. 23.28	0,92	Implementasjon / koding	Oppdater frontend schemaer	
21/03/2024	kl. 13.00	kl. 13.34	0,57	Implementasjon / koding	Oppsett admin page	
21/03/2024	kl. 13.40	kl. 14.05	0,42	Møter		
21/03/2024	kl. 14.15	kl. 18.20	4,08	Implementasjon / koding	Implementer registrering	
22/03/2024	kl. 12.10	kl. 13.00	0,83	Møter		
22/03/2024	kl. 13.00	kl. 16.30	3,50	Implementasjon / koding		
25/03/2024	kl. 12.00	kl. 14.20	2,33	Møter	Sprint review + planning	
28/03/2024	kl. 16.40	kl. 17.35	0,92	Testing	MR Review Backend	
30/03/2024	kl. 16.20	kl. 17.15	0,92	Implementasjon / koding	Fix frontend (equip sizes i prod)	
30/03/2024	kl. 18.45	kl. 19.05	0,33	Læring		
01/04/2024	kl. 12.00	kl. 13.00	1,00	Møter		
01/04/2024	kl. 13.00	kl. 16.30	3,50	Design	Test forskjellige design for item page	
02/04/2024	kl. 14.00	kl. 16.20	2,33	Implementasjon / koding		
02/04/2024	kl. 17.04	kl. 17.43	0,65	Implementasjon / koding	Ferdigjør frontendendringer	

02/04/2024	kl. 17.44	kl. 17.49	0,08	Dokumentasjon				
02/04/2024	kl. 17.49	kl. 18.25	0,60	Testing	test ny frontend			
02/04/2024	kl. 18.25	kl. 18.59	0,57	Dokumentasjon	dokumenter endringene			
02/04/2024	kl. 19.01	kl. 19.28	0,45	Design	diskusjon rundt orderID's			
			0,00	Ukategorisert				
04/04/2024	kl. 13.30	kl. 14.00	0,50	Ukategorisert	Møte veileder			
04/04/2024	kl. 14.00	kl. 16.30	2,50	Implementasjon / koding	Ferdigjør MR frontend			
04/04/2024	kl. 17.00	kl. 17.17	0,28	Dokumentasjon	Start diagram for CI/CD			
05/04/2024	kl. 12.00	kl. 13.00	1,00	Møter	Oppdragsgiver			
05/04/2024	kl. 14.20	kl. 14.30	0,17	Testing	Backend Rate limit review			
05/04/2024	kl. 14.40	kl. 15.20	0,67	Dokumentasjon	Lag Diagram			
08/04/2024	kl. 10.45	kl. 14.01	3,27	Implementasjon / koding				
09/04/2024	kl. 15.00	kl. 20.30	5,50	Design				
			0,00	Ukategorisert				
11/04/2024	kl. 13.30	kl. 14.10	0,67	Møter				
11/04/2024	kl. 14.20	kl. 17.00	2,67	Implementasjon / koding				
			0,00	Ukategorisert				
14/04/2024	kl. 11.30	kl. 13.30	2,00	Testing	Mr reviews			
14/04/2024	kl. 14.20	kl. 18.30	4,17	Implementasjon / koding				
15/04/2024	kl. 17.25	kl. 17.46	0,35	Dokumentasjon	Skriv frontend PR dokumentasjon			
16/04/2024	kl. 12.05	kl. 15.25	3,33	Implementasjon / koding	Jobb og merge frontend MR i main			
16/04/2024	kl. 15.25	kl. 15.55	0,50	Testing	Test i prod			
16/04/2024	kl. 15.55	kl. 17.30	1,58	Implementasjon / koding	Implementer MVP funksjonalitet			
18/04/2024	kl. 14.15	kl. 18.04	3,82	Implementasjon / koding				
19/04/2024	kl. 12.00	kl. 12.40	0,67	Møter	Gruppermøte			
19/04/2024	kl. 12.40	kl. 14.12	1,53	Design	Backend MR reviews			
19/04/2024	kl. 14.35	kl. 15.30	0,92	Design	Backend MR reviews			
22/04/2024	kl. 10.40	kl. 12.00	1,33	Møter	Sprint Review			
22/04/2024	kl. 12.00	kl. 13.45	1,75	Rapportskriving	Formattering rapport			
22/04/2024	kl. 17.00	kl. 19.15	2,25	Møter	Sprint planlegging			
22/04/2024	kl. 10.20	kl. 12.00	1,67	Testing	MR Reviews			
24/04/2024	kl. 16.40	kl. 17.08	0,47	Testing				
24/04/2024	kl. 17.20	kl. 17.33	0,22	Testing				
25/04/2024	kl. 13.30	kl. 14.20	0,83	Møter	Veiledermøte			
25/04/2024	kl. 14.25	kl. 15.55	1,50	Testing	MR Reviews			
25/04/2024	kl. 16.10	kl. 18.25	2,25	Testing				
25/04/2024	kl. 18.40	kl. 18.50	0,17	Testing				
26/04/2024	kl. 01.30	kl. 03.00	1,50	Implementasjon / koding	Fiks driftproblemer			
26/04/2024	kl. 12.00	kl. 12.50	0,83	Møter	Arbeidsgivermøte			
26/04/2024	kl. 12.50	kl. 14.30	1,67	Læring	Studer MySQL dokumentasjon			
26/04/2024	kl. 14.30	kl. 15.50	1,33	Testing	MR Reviews frontend			
29/04/2024	kl. 10.02	kl. 13.00	2,97	Møter				
29/04/2024	kl. 16.40	kl. 17.04	0,40	Testing				
29/04/2024	kl. 17.10	kl. 17.40	0,50	Rapportskriving	Formattering			
30/04/2024	kl. 14.05	kl. 14.45	0,67	Møter				
30/04/2024	kl. 14.45	kl. 17.30	2,75	Rapportskriving				
30/04/2024	kl. 17.30	kl. 18.02	0,53	Testing				
01/05/2024	kl. 10.45	kl. 11.35	0,83	Testing	MR Review User List Page			
03/05/2024	kl. 09.45	kl. 10.05	0,33	Testing	MR Review småfix			
05/05/2024	kl. 19.30	kl. 22.30	3,00	Implementasjon / koding	Verifisering skjema frontend			
06/05/2024	kl. 10.20	kl. 11.00	0,67	Møter				
06/05/2024	kl. 11.15	kl. 15.30	4,25	Møter	Sprint møte - Rapportskriving			
06/05/2024	kl. 15.30	kl. 16.30	1,00	Møter	Veiledermøte			
07/05/2024	kl. 10.20	kl. 11.00	0,67	Møter	Standupmøte			
07/05/2024	kl. 11.00	kl. 13.00	2,00	Implementasjon / koding	frontend MR			
07/05/2024	kl. 13.00	kl. 15.00	2,00	Rapportskriving	Rapportplanlegging			
07/05/2024	kl. 20.00	kl. 21.00	1,00	Læring	Les Viten i Senter			
08/05/2024	kl. 14.00	kl. 15.00	1,00	Læring				
08/05/2024	kl. 16.00	kl. 18.05	2,08	Testing	MR Review			
09/05/2024	kl. 12.20	kl. 15.05	2,75	Møter	Rapportskrivingsmøte			
09/05/2024	kl. 15.20	kl. 15.40	0,33	Rapportskriving				
			0,00	Ukategorisert				
13/05/2024	kl. 10.20	kl. 11.30	1,17	Møter				
13/05/2024	kl. 12.00	kl. 13.00	1,00	Testing	Frontend happytesting			
13/05/2024	kl. 13.00	kl. 13.20	0,33	Implementasjon / koding	fix frontend skjema			
13/05/2024	kl. 14.15	kl. 14.30	0,25	Rapportskriving				
13/05/2024	kl. 14.45	kl. 15.00	0,25	Rapportskriving				
13/05/2024	kl. 15.20	kl. 16.35	1,25	Rapportskriving				
			0,00	Ukategorisert				
15/05/2024	kl. 14.00	kl. 15.00	1,00	Møter	Møte veileder			
15/05/2024	kl. 15.00	kl. 18.00	3,00	Rapportskriving				
16/05/2024	kl. 12.00	kl. 13.40	1,67	Møter	møte arbeidsgiver			
16/05/2024	kl. 14.10	kl. 16.25	2,25	Rapportskriving	rapportskriving			
17/05/2024	kl. 12.30	kl. 13.40	1,17	Rapportskriving				
17/05/2024	kl. 21.30	kl. 23.59	2,48	Rapportskriving				
18/05/2024	kl. 00.00	kl. 01.00	1,00	Rapportskriving				
18/05/2024	kl. 11.30	kl. 14.40	3,17	Rapportskriving				
18/05/2024	kl. 16.22	kl. 16.25	0,05	Rapportskriving				
18/05/2024	kl. 16.55	kl. 17.30	0,58	Ukategorisert	Review timebruk			
18/05/2024	kl. 17.40	kl. 18.50	1,17	Rapportskriving	mer rapportskriving			
18/05/2024	kl. 20.17	kl. 21.35	1,30	Rapportskriving				
18/05/2024	kl. 22.45	kl. 23.45	1,00	Rapportskriving				

F.1.3 Åsulv

Data hentet 20/05/2024 kl. 17

Begynner på neste side

Dato	Fra	Til	Timer	Kategori	Beskrivelse		Timebruk
			0,00	Ukategorisert		Møter	166,37
			0,00	Ukategorisert		Implementasjon / koding	68,37
			0,00	Ukategorisert		Design	3,23
15/01/2024	kl. 12.00	kl. 14.00	2,00	Møter		Dokumentasjon	0,00
16/01/2024	kl. 12.00	kl. 14.23	2,38	Møter		Testing	0,17
17/01/2024	kl. 12.05	kl. 12.20	0,25	Rapportskriving		Rapportskriving	78,88
17/01/2024	kl. 18.30	kl. 18.50	0,33	Rapportskriving		Læring	1,75
18/01/2024	kl. 13.30	kl. 14.00	0,50	Møter	Møte med veileder		0,00
18/01/2024	kl. 14.00	kl. 15.52	1,87	Møter			0,00
22/01/2024	kl. 13.15	kl. 16.30	3,25	Møter		Ukategorisert	0,00
23/01/2024	kl. 15.15	kl. 15.30	0,25	Rapportskriving			
24/01/2024	kl. 12.40	kl. 13.18	0,63	Rapportskriving	Jobb med risikoanalyse	Sammenlagt	318,77
24/01/2024	kl. 15.15	kl. 17.17	2,03	Møter			
25/01/2024	kl. 13.35	kl. 14.20	0,75	Møter	Møte med veileder		
25/01/2024	kl. 15.30	kl. 17.30	2,00	Møter			
26/01/2024	kl. 13.00	kl. 15.10	2,17	Møter	Møte med oppdragsgiver		
26/01/2024	kl. 15.10	kl. 15.30	0,33	Møter			
26/01/2024	kl. 15.55	kl. 16.25	0,50	Møter			
29/01/2024	kl. 12.50	kl. 15.00	2,17	Møter			
29/01/2024	kl. 17.10	kl. 17.41	0,52	Rapportskriving			
30/01/2024	kl. 10.00	kl. 14.00	4,00	Møter			
31/01/2024	kl. 10.14	kl. 11.30	1,27	Møter			
01/02/2024	kl. 13.35	kl. 14.05	0,50	Møter	Møte med veileder		
01/02/2024	kl. 14.10	kl. 15.24	1,23	Møter			
02/02/2024	kl. 13.30	kl. 16.22	2,87	Møter			
04/02/2024	kl. 21.10	kl. 21.30	0,33	Implementasjon / koding			
05/02/2024	kl. 10.15	kl. 11.43	1,47	Møter			
05/02/2024	kl. 13.00	kl. 13.45	0,75	Læring	Testing med JWT		
07/02/2024	kl. 15.40	kl. 15.55	0,25	Design	Ser på API design		
08/02/2024	kl. 12.00	kl. 13.10	1,17	Møter			
08/02/2024	kl. 13.35	kl. 14.00	0,42	Møter	Møte med veileder		
08/02/2024	kl. 14.05	kl. 16.00	1,92	Møter			
08/02/2024	kl. 18.50	kl. 20.10	1,33	Implementasjon / koding			
08/02/2024	kl. 20.45	kl. 21.05	0,33	Implementasjon / koding			
09/02/2024	kl. 12.03	kl. 13.37	1,57	Møter	Møte med oppdragsgiver		
09/02/2024	kl. 13.37	kl. 13.50	0,22	Møter			
12/02/2024	kl. 10.20	kl. 12.00	1,67	Møter	Start av sprint 3		
12/02/2024	kl. 17.20	kl. 18.27	1,12	Design	Design API		
13/02/2024	kl. 12.20	kl. 13.30	1,17	Design	Design API		
14/02/2024	kl. 10.10	kl. 10.58	0,80	Møter			
14/02/2024	kl. 13.10	kl. 14.40	1,50	Implementasjon / koding			
14/02/2024	kl. 15.35	kl. 16.10	0,58	Implementasjon / koding			
15/02/2024	kl. 13.33	kl. 14.13	0,67	Møter	Møte med veileder		
15/02/2024	kl. 14.23	kl. 15.55	1,53	Møter			
16/02/2024	kl. 12.00	kl. 16.05	4,08	Møter	Møte + jobbing med backend API		
19/02/2024	kl. 10.05	kl. 14.30	4,42	Møter	Møte + jobbing med backend API		
19/02/2024	kl. 17.00	kl. 17.10	0,17	Testing	Se over MR #2 (back-end)		
20/02/2024	kl. 10.05	kl. 10.30	0,42	Møter	Stand-up møte tirsdag		
21/02/2024	kl. 10.00	kl. 11.16	1,27	Møter	Stand-up møte onsdag		
21/02/2024	kl. 17.50	kl. 18.20	0,50	Implementasjon / koding			
21/02/2024	kl. 18.40	kl. 19.25	0,75	Implementasjon / koding			
21/02/2024	kl. 19.33	kl. 19.50	0,28	Implementasjon / koding			
22/02/2024	kl. 13.30	kl. 14.05	0,58	Møter	Møte med veileder		
22/02/2024	kl. 14.35	kl. 16.21	1,77	Møter			
23/02/2024	kl. 13.35	kl. 14.44	1,15	Implementasjon / koding			
26/02/2024	kl. 10.05	kl. 10.33	0,47	Implementasjon / koding			
26/02/2024	kl. 10.33	kl. 13.30	2,95	Møter			
26/02/2024	kl. 15.40	kl. 17.50	2,17	Implementasjon / koding	Inkluderer skriving av MR		
28/02/2024	kl. 10.15	kl. 10.40	0,42	Møter	"Stand-up møte"		
28/02/2024	kl. 13.40	kl. 14.05	0,42	Møter	Møte med veileder		
28/02/2024	kl. 14.05	kl. 14.40	0,58	Møter			
29/02/2024	kl. 14.05	kl. 17.00	2,92	Møter			
01/03/2024	kl. 12.13	kl. 15.25	3,20	Møter			
02/03/2024	kl. 13.10	kl. 14.35	1,42	Implementasjon / koding			
03/03/2024	kl. 17.30	kl. 19.55	2,42	Implementasjon / koding			
04/03/2024	kl. 10.00	kl. 14.12	4,20	Møter	Inkluderer jobbing på backend API		
06/03/2024	kl. 17.30	kl. 19.00	1,50	Implementasjon / koding			
07/03/2024	kl. 11.07	kl. 11.33	0,43	Implementasjon / koding			
07/03/2024	kl. 13.30	kl. 14.00	0,50	Møter	Møte med veileder		
07/03/2024	kl. 14.00	kl. 18.35	4,58	Møter	Inkluderer skriving av MR		
08/03/2024	kl. 12.00	kl. 13.30	1,50	Møter	Møte med oppdragsgiver		
08/03/2024	kl. 13.30	kl. 15.30	2,00	Møter			
11/03/2024	kl. 10.25	kl. 12.45	2,33	Møter			
12/03/2024	kl. 11.53	kl. 15.00	3,12	Implementasjon / koding			
12/03/2024	kl. 16.07	kl. 18.09	2,03	Implementasjon / koding			
13/03/2024	kl. 10.10	kl. 13.10	3,00	Møter			
13/03/2024	kl. 17.00	kl. 19.22	2,37	Implementasjon / koding			
14/03/2024	kl. 13.30	kl. 14.10	0,67	Møter	Møte med veileder		
14/03/2024	kl. 14.10	kl. 17.25	3,25	Implementasjon / koding			
15/03/2024	kl. 12.11	kl. 18.36	6,42	Implementasjon / koding			
18/03/2024	kl. 10.20	kl. 14.04	3,73	Møter			
21/03/2024	kl. 13.30	kl. 14.10	0,67	Møter	Møte med veileder		

21/03/2024	kl. 14.25	kl. 18.16	3,85	Møter			
22/03/2024	kl. 12.00	kl. 13.00	1,00	Møter	Møte med oppdragsgiver		
22/03/2024	kl. 13.00	kl. 16.30	3,50	Møter			
25/03/2024	kl. 12.00	kl. 14.10	2,17	Møter	Sprint review + planning		
25/03/2024	kl. 14.30	kl. 15.50	1,33	Implementasjon / koding			
25/03/2024	kl. 16.10	kl. 17.22	1,20	Implementasjon / koding			
26/03/2024	kl. 13.40	kl. 15.13	1,55	Implementasjon / koding			
27/03/2024	kl. 11.20	kl. 12.18	0,97	Implementasjon / koding	Mest skrivning av MR		
01/04/2024	kl. 10.05	kl. 12.35	2,50	Møter			
02/04/2024	kl. 12.00	kl. 12.42	0,70	Implementasjon / koding	Se på ClickSend		
			0,00	Ukategorisert			
02/04/2024	kl. 19.01	kl. 19.28	0,45	Design	diskusjon rundt orderID's		
03/04/2024	kl. 10.15	kl. 10.55	0,67	Møter			
03/04/2024	kl. 14.40	kl. 15.13	0,55	Implementasjon / koding	Sett på rate limiting		
04/04/2024	kl. 13.30	kl. 14.00	0,50	Møter	Møte med veileder		
04/04/2024	kl. 14.00	kl. 16.32	2,53	Implementasjon / koding			
05/04/2024	kl. 12.03	kl. 13.00	0,95	Møter	Møte med oppdragsgiver		
05/04/2024	kl. 13.00	kl. 16.06	3,10	Implementasjon / koding			
06/04/2024	kl. 10.20	kl. 11.10	0,83	Implementasjon / koding			
06/04/2024	kl. 15.30	kl. 17.17	1,78	Implementasjon / koding			
08/04/2024	kl. 10.45	kl. 14.01	3,27	Møter	Sprint review / sprint planning		
08/04/2024	kl. 16.15	kl. 18.03	1,80	Implementasjon / koding			
11/04/2024	kl. 13.30	kl. 14.00	0,50	Møter	Møte med veileder		
11/04/2024	kl. 14.25	kl. 17.00	2,58	Møter			
12/04/2024	kl. 13.16	kl. 16.33	3,28	Implementasjon / koding			
13/04/2024	kl. 15.20	kl. 17.26	2,10	Implementasjon / koding			
15/04/2024	kl. 12.05	kl. 12.48	0,72	Implementasjon / koding			
15/04/2024	kl. 13.00	kl. 13.25	0,42	Møter			
15/04/2024	kl. 13.30	kl. 15.17	1,78	Implementasjon / koding			
15/04/2024	kl. 16.10	kl. 17.23	1,22	Implementasjon / koding			
16/04/2024	kl. 12.10	kl. 17.15	5,08	Møter			
17/04/2024	kl. 12.53	kl. 15.30	2,62	Implementasjon / koding			
17/04/2024	kl. 16.00	kl. 16.15	0,25	Design			
18/04/2024	kl. 13.30	kl. 14.00	0,50	Møter	Møte med veileder		
18/04/2024	kl. 14.15	kl. 18.04	3,82	Møter			
19/04/2024	kl. 12.00	kl. 16.00	4,00	Møter			
19/04/2024	kl. 16.00	kl. 17.06	1,10	Implementasjon / koding			
21/04/2024	kl. 14.10	kl. 16.00	1,83	Implementasjon / koding			
22/04/2024	kl. 10.03	kl. 12.00	1,95	Møter	Sprint review / sprint planning		
22/04/2024	kl. 17.00	kl. 19.17	2,28	Møter	Sprint review / sprint planning		
23/04/2024	kl. 12.20	kl. 15.04	2,73	Implementasjon / koding			
24/04/2024	kl. 10.05	kl. 10.20	0,25	Møter			
24/04/2024	kl. 11.23	kl. 13.20	1,95	Implementasjon / koding	For det meste skrivning av MR		
24/04/2024	kl. 15.30	kl. 16.30	1,00	Læring	Sett igjennom andre rapporter		
25/04/2024	kl. 13.30	kl. 14.10	0,67	Møter	Møte med veileder		
25/04/2024	kl. 14.25	kl. 17.22	2,95	Møter			
26/04/2024	kl. 12.00	kl. 12.55	0,92	Møter	Møte med oppdragsgiver		
26/04/2024	kl. 12.55	kl. 15.56	3,02	Møter			
26/04/2024	kl. 16.45	kl. 17.40	0,92	Rapportskriving	Skriving av rapport + testing av glossaries (ordforklaring) i Latex		
28/04/2024	kl. 16.04	kl. 17.21	1,28	Rapportskriving			
28/04/2024	kl. 18.03	kl. 18.58	0,92	Rapportskriving			
29/04/2024	kl. 10.09	kl. 12.26	2,28	Møter			
29/04/2024	kl. 13.43	kl. 16.41	2,97	Rapportskriving			
30/04/2024	kl. 14.00	kl. 14.45	0,75	Møter	Møte med veileder		
30/04/2024	kl. 15.00	kl. 18.00	3,00	Møter			
01/05/2024	kl. 16.40	kl. 18.22	1,70	Rapportskriving			
02/05/2024	kl. 11.15	kl. 14.54	3,65	Rapportskriving			
02/05/2024	kl. 16.10	kl. 16.30	0,33	Implementasjon / koding	Fjernet HelloWorld-handler		
03/05/2024	kl. 12.10	kl. 15.30	3,33	Møter	Møte + jobb med wiki		
03/05/2024	kl. 16.35	kl. 17.26	0,85	Rapportskriving			
05/05/2024	kl. 13.09	kl. 14.47	1,63	Rapportskriving			
06/05/2024	kl. 10.10	kl. 15.25	5,25	Møter			
06/05/2024	kl. 15.33	kl. 16.13	0,67	Møter	Møte med veileder		
07/05/2024	kl. 10.21	kl. 14.56	4,58	Møter			
07/05/2024	kl. 19.20	kl. 20.52	1,53	Rapportskriving			
08/05/2024	kl. 10.30	kl. 14.00	3,50	Rapportskriving			
08/05/2024	kl. 16.00	kl. 18.26	2,43	Rapportskriving			
09/05/2024	kl. 10.31	kl. 11.08	0,62	Rapportskriving			
09/05/2024	kl. 12.00	kl. 15.00	3,00	Møter			
09/05/2024	kl. 17.10	kl. 19.01	1,85	Rapportskriving			
10/05/2024	kl. 10.45	kl. 14.22	3,62	Rapportskriving			
10/05/2024	kl. 15.38	kl. 16.18	0,67	Rapportskriving			
10/05/2024	kl. 16.47	kl. 18.32	1,75	Rapportskriving			
11/05/2024	kl. 11.55	kl. 15.30	3,58	Rapportskriving			
11/05/2024	kl. 17.21	kl. 18.05	0,73	Rapportskriving			
12/05/2024	kl. 12.30	kl. 16.03	3,55	Rapportskriving			
12/05/2024	kl. 18.13	kl. 19.28	1,25	Rapportskriving			
13/05/2024	kl. 10.19	kl. 14.00	3,68	Møter			
13/05/2024	kl. 15.42	kl. 17.18	1,60	Rapportskriving			
14/05/2024	kl. 12.15	kl. 15.20	3,08	Rapportskriving			
14/05/2024	kl. 16.30	kl. 19.03	2,55	Rapportskriving			
15/05/2024	kl. 10.11	kl. 13.09	2,97	Rapportskriving			
15/05/2024	kl. 14.00	kl. 15.00	1,00	Møter	Møte med veileder		

F.1.4 Ferdinand

Data hentet 20/05/2024 kl. 17

Begynner på neste side

Dato	Fra	Til	Timer	Kategori	Beskrivelse		Timebruk
15/01/2024	kl. 12.00.00	kl. 14.00.00	2,00	Møter	Møtelog 1	Møter	54,00
15/01/2024	kl. 14.00	kl. 17.30	3,50	Design		Implementasjon / koding	160,83
16.01.2024	kl. 12.00	kl. 14.30	2,50	Møter	Møtelog 2	Design	12,00
18/01/2024	kl. 14.00	kl. 16.00	2,00	Møter	Møtelog 2	Dokumentasjon	1,00
19/01/2024	kl. 13.00	kl. 17.30	4,50	Design		Testing	2,00
20/01/2024	kl. 11.30	kl. 14.00	2,50	Design		Rapportskriving	74,78
22/01/2024	kl. 12.00	kl. 13.00	1,00	Design	Jobbet først en time på skolen	Læring	5,50
22/01/2024	kl. 13.00	kl. 13.30	0,50	Møter	Endte opp med å ta møte på vei hjem		0,00
22/01/2024	kl. 17.30	kl. 19.30	2,00	Rapportskriving			0,00
23/01/2024	kl. 10.15	kl. 12.30.00	2,25	Rapportskriving	Jobbet på skole.	Ukategorisert	0,00
25/01/2024	kl. 13.00.00	kl. 14.30.00	1,50	Møter	Møte med Veileder		
25/01/2024	kl. 14.30.00	kl. 17.30.00	3,00	Rapportskriving		Sammenlagt	310,12
25/01/2024	kl. 22.00	kl. 24.00.00	2,00	Rapportskriving			
26/01/2024	kl. 00.00	kl. 02.00	2,00	Rapportskriving	late night Prosjektplan		
26/01/2024	kl. 13.00	kl. 15.00	2,00	Møter	Møte med arbeidsgiver		
26/01/2024	kl. 15.00	kl. 16.00	1,00	Dokumentasjon			
30/01/2024	kl. 10.00	kl. 14.00	4,00	Møter			
30/01/2024	kl. 18.00	kl. 19.30	1,50	Rapportskriving			
31/01/2024	kl. 10.00	kl. 14.15	4,25	Rapportskriving	Satt opp React og Frontend repo		
01/02/2024	kl. 13.30	kl. 15.00	1,50	Møter			
2/2/24	kl. 12.30	kl. 16.00	3,50	Læring	Lærer react med typescript + setter opp tailwind		
5/2/24	kl. 09.30	kl. 11.30	2,00	Implementasjon / koding			
8/2/24	kl. 11.30	kl. 16.00	4,50	Implementasjon / koding	Litt frontend og møter med arbeidsgiver		
9/2/24	kl. 09.00	kl. 11.00	2,00	Testing	Testing av konkurent BUA		
9/2/24	kl. 12.00	kl. 13.00	1,00	Møter	Møte med arbeidsgiver		
9/2/24	kl. 13.00	kl. 15.30	2,50	Rapportskriving	Skrev om devtools til frontend		
9/2/24	kl. 20.00	kl. 22.00	2,00	Læring	Mere React + Typescript		
12/2/24	kl. 10.00	kl. 12.00	2,00	Møter	Start av Sprint 3		
12/2/24	kl. 19.30	kl. 22.30	3,00	Implementasjon / koding	React Nav bar and nav tools		
13/2/24	kl. 09.30	kl. 12.30	3,00	Implementasjon / koding	Jobb med React og fil struktur		
16/2/2024	kl. 13.00	kl. 17.00	4,00	Implementasjon / koding			
19/02/2024	kl. 08.30	kl. 10.00	1,50	Implementasjon / koding	Login Meny		
20/02/2024	kl. 12.30	kl. 14.00	1,50	Implementasjon / koding			
21/02/2024	kl. 10.00	kl. 13.30	3,50	Implementasjon / koding	Items		
21/02/2024	kl. 17.15	kl. 19.30	2,25	Implementasjon / koding	Search + Tags		
22/02/2024	kl. 13.30	kl. 16.00	2,50	Møter	Møte med veileder		
26/02/2024	kl. 10.00	kl. 13.00	3,00	Implementasjon / koding	M. Møte		
27/02/2024	kl. 13.00	kl. 17.30	4,50	Implementasjon / koding	Hadde masse IDE problemer som jeg måtte fikke ()		
28/02/2024	kl. 10.15	kl. 11.15	1,00	Møter	Oppfølgings møte		
29/02/2024	kl. 13.30	kl. 16.50	3,33	Implementasjon / koding			
01/03/2024	kl. 10.00	kl. 11.00	1,00	Implementasjon / koding			
01/03/2024	kl. 12.00	kl. 15.30	3,50	Implementasjon / koding			
04/03/2024	kl. 10.00	kl. 12.00	2,00	Møter			
06/03/2024	kl. 10.00	kl. 14.00	4,00	Implementasjon / koding			
07/03/2024	kl. 13.30	kl. 14.00	0,50	Møter	møte med rådgiver		
07/03/2024	kl. 14.00	kl. 18.30	4,50	Implementasjon / koding	Item page		
08/03/2024	kl. 12.00	kl. 15.30	3,50	Implementasjon / koding	Implementasjon + Arbeidsgiver		
11/03/2024	kl. 10.15	kl. 12.00	1,75	Implementasjon / koding			
13/03/2024	kl. 12.30	kl. 14.30	2,00	Implementasjon / koding			
13/03/2024	kl. 16.00	kl. 19.30	3,50	Implementasjon / koding	Not to productive day but a day none the less		
14/03/2024	kl. 13.30	kl. 14.00	0,50	Møter	med veileder		
14/03/2024	kl. 14.00	kl. 17.30	3,50	Implementasjon / koding			
15/03/2024	kl. 12.00	kl. 17.00	5,00	Implementasjon / koding			
18/03/2024	kl. 10.15	kl. 11.45	1,50	Møter			
18/03/2024	kl. 12.00	kl. 14.00	2,00	Implementasjon / koding			
18/03/2024	kl. 17.30	kl. 19.30	2,00	Implementasjon / koding			
19/03/2024	kl. 10.15	kl. 12.15	2,00	Implementasjon / koding	Hooking up backend		
20/03/2024	kl. 11.15	kl. 16.00	4,75	Implementasjon / koding			
21/03/2024	kl. 10.45	kl. 12.00	1,25	Implementasjon / koding			
21/03/2024	kl. 13.30	kl. 17.30	4,00	Implementasjon / koding	pluss møte emd rådgiver		
22/03/2024	kl. 12.00	kl. 13.00	1,00	Møter	Møte med arbeidsgiver		
22/03/2024	kl. 13.00	kl. 17.00	4,00	Implementasjon / koding	Implementasjon av RentHistory		
25/03/2024	kl. 12.00	kl. 14.00	2,00	Møter			
25/03/2024	kl. 15.30	kl. 17.00	1,50	Implementasjon / koding			
27/03/2024	kl. 17.00	kl. 19.00	2,00	Implementasjon / koding			
28/03/2024	kl. 11.15	kl. 13.30	2,25	Implementasjon / koding			
30/03/2024	kl. 15.00	kl. 19.30	4,50	Implementasjon / koding			
31/03/2024	kl. 10.30	kl. 13.30	3,00	Implementasjon / koding			
31/03/2024	kl. 17.15	kl. 18.30	1,25	Implementasjon / koding			
03/04/2024	kl. 10.30	kl. 12.30	2,00	Rapportskriving			
04/04/2024	kl. 14.00	kl. 15.30	1,50	Møter			
05/04/2024	kl. 12.00	kl. 13.00	1,00	Møter			
05/04/2024	kl. 13.15	kl. 16.00	2,75	Implementasjon / koding			
09/04/2024	kl. 15.00	kl. 20.00	5,00	Implementasjon / koding	Merged + Running backend locally		
09/04/2024	kl. 20.00	kl. 20.30	0,50	Design	Design av Loan page		
10/04/2024	kl. 16.15	kl. 18.00	1,75	Implementasjon / koding			
11/04/2024	kl. 13.30	kl. 17.00	3,50	Implementasjon / koding	Implementasjon + Veileder møte		
12/04/2024	kl. 12.30	kl. 14.20	1,83	Implementasjon / koding			
13/04/2024	kl. 13.00	kl. 17.00	4,00	Implementasjon / koding	Hooked up a lot of backend		
13/04/2024	kl. 19.00	kl. 21.00	2,00	Implementasjon / koding	Mere hooking up of backend + todays loans		
14/04/2024	kl. 10.45	kl. 14.10	3,42	Implementasjon / koding			

F.2 Per uke

Data hentet 20/05/2024 kl. 17

Begynner på neste side

Uke 3

Ukenummer	3		Fra		15/01/2024		Til		21/01/2024	
	Erlend	Ferdinand	Noah	Åsulv	Sammenlagt					
Møter	0,00	6,50	6,75	6,75	20,00					
Implementasjon / koding	0,00	0,00	0,00	0,00	0,00					
Design	0,00	10,50	0,00	0,00	10,50					
Dokumentasjon	0,00	0,00	0,00	0,00	0,00					
Testing	0,00	0,00	0,00	0,00	0,00					
Rapportskriving	0,00	0,00	0,00	0,58	0,58					
Læring	0,00	0,00	0,00	0,00	0,00					
	0,00	0,00	0,00	0,00	0,00					
	0,00	0,00	0,00	0,00	0,00					
Ukategorisert	0,00	0,00	0,00	0,00	0,00					
Sammenlagt	0,00	17,00	6,75	7,33	31,08					

Uke 4

Ukenummer	4		Fra		22/01/2024		Til		28/01/2024	
	Erlend	Ferdinand	Noah	Åsulv	Sammenlagt					
Møter	0,00	4,00	10,20	11,03	25,23					
Implementasjon / koding	0,00	0,00	0,00	0,00	0,00					
Design	0,00	1,00	0,00	0,00	1,00					
Dokumentasjon	0,00	1,00	2,00	0,00	3,00					
Testing	0,00	0,00	0,00	0,00	0,00					
Rapportskriving	0,00	11,25	0,00	0,88	12,13					
Læring	0,00	0,00	0,00	0,00	0,00					
	0,00	0,00	0,00	0,00	0,00					
	0,00	0,00	0,00	0,00	0,00					
Ukategorisert	0,00	0,00	0,00	0,00	0,00					
Sammenlagt	0,00	17,25	12,20	11,92	41,37					

G Kodeutdrag: Backend

Her følger utdrag fra kildekoden til backend-komponenten i applikasjonen. Disse skal i hovedsak sees i sammenheng med der de er referert til i hovedteksten i rapporten.

G.1 Kodeeksempel: Handler-funksjon

Dette er et eksempel på en såkalt handlerfunksjon. Eksemplet nedenfor viser handler-funksjonen for å lage en kategori. Hvordan denne handler-funksjonen brukes er dokumentert i vedlegg J.2.

```
1 func CategoriesCreateHandler(w http.ResponseWriter, r *http.Request) {
2     LogRequestStart(r)
3
4     // Only HTTP POST requests are allowed
5     if MethodCheck(w, r, http.MethodPost) {
6         return
7     }
8
9     // Perform access control
10    res, appError := app.AccessControl(r, false, "Inventory", data.Permission{Write: true})
11    if appError != nil || !res.Allowed {
12        error_handling.LogError(w, r, error_handling.UnauthorizedError(appError))
13        return
14    }
15
16    // Decode incoming JSON
17    var req data.NewCategoryRequest
18
19    err := json.NewDecoder(r.Body).Decode(&req)
20    if err != nil {
21        error_handling.LogError(w, r, error_handling.DecodingError(err))
22        return
23    }
24
25    // Create the category
26    resp, appError := app.CreateCategory(req)
27    if appError != nil {
28        error_handling.LogError(w, r, appError)
29        return
30    }
31
32    // Return response
33    w.Header().Set("Content-Type", "application/json")
34    err = json.NewEncoder(w).Encode(resp)
35    if err != nil {
36        error_handling.LogError(w, r, error_handling.ProcessingError(err))
37        return
38    }
39 }
```

G.2 Kodeutdrag: Finn gyldige datoer

Dette er koden backend bruker for å finne gyldige datoer til hentdato og leveringsdato til en reservasjon.

Funksjonene *util.StripTime* og *util.StripDate* er hjelpefunksjoner som fjerner tid og dato fra Go *time.Time* variabler som i utgangspunktet inneholder både dato og tid, dette gjøres for at sjekkene ikke skal bli påvirket av tid eller dato der det ikke er relevant.

```

1 // Gets the allowed dates based on the weekly schedule and exception dates
2 // Start today determines if the start is today (adds check for closing time) or not (ignores start date)
3 func GetAllowedDates(startDate time.Time, startIsToday bool, maxDays int) ([]time.Time, *data.AppError) {
4     // Get weekly schedule and exception days
5     weeklySchedule, appError := db.GetWeeklySchedule()
6     if appError != nil {
7         return nil, appError
8     }
9
10    exceptions, appError := db.GetExceptionDates()
11    if appError != nil {
12        return nil, appError
13    }
14
15    allowedDays := []time.Time{}
16
17    currentDay := util.StripTime(startDate)
18    currentTime := util.StripDate(startDate)
19    dayCounter := 0
20
21    for dayCounter <= maxDays {
22        addCurrentDay := false
23
24        // Check weekly schedule
25        currentWeekday := currentDay.Weekday()
26
27        for _, scheduledDay := range weeklySchedule {
28            // Check if we have a match and the weekday is marked as open
29            if scheduledDay.Weekday == uint(currentWeekday) && scheduledDay.IsOpen {
30                // If checking today, the time must be before closing
31                if dayCounter == 0 && currentTime.Before(util.StripDate(scheduledDay.ClosingTime)) && startI
32                    addCurrentDay = true
33            } else if dayCounter != 0 {
34                addCurrentDay = true
35            }
36        }
37    }
38
39    // Check exception dates
40    for _, exceptionDate := range exceptions {
41        // Check for match
42        if util.StripTime(exceptionDate.Date).Equal(currentDay) {
43            if !exceptionDate.IsOpen {
44                addCurrentDay = false
45            } else if dayCounter == 0 && currentTime.Before(util.StripDate(exceptionDate.ClosingTime)) &
46                addCurrentDay = true
47            } else if dayCounter != 0 {
48                addCurrentDay = true

```

```
49         }
50     }
51 }
52
53     // Add day if valid
54     if addCurrentDay {
55         allowedDays = append(allowedDays, currentDay)
56     }
57
58     // Advance
59     currentDay = currentDay.Add(24 * time.Hour)
60     dayCounter += 1
61 }
62
63 return allowedDays, nil
64 }
```

G.3 Kodeutdrag: Parsing av en multipart form

Utdraget under er hentet fra handler-funksjonen som håndterer opplasting av bilder og viser hvordan *multipart formen* blir håndtert samt validering av blant annet filtypen til filen som blir lastet opp.

```
1     // Parse multipart form
2     // "<< 20" adds 1048576 to maxFileSize, this converts the maxFileSize value
3     // from megabytes to bytes
4     err = r.ParseMultipartForm(int64(maxFileSize) << 20)
5     if err != nil {
6         error_handling.ProcessingError(err)
7         return
8     }
9
10    // Get file data
11    file, handler, err := r.FormFile("image")
12    if err != nil {
13        error_handling.LogError(w, r, error_handling.DecodingError(err))
14        return
15    }
16
17    // Get picture description
18    description := r.FormValue("description")
19
20    // Get order position
21    orderPositionStr := r.FormValue("order_position")
22    orderPosition, err := strconv.Atoi(orderPositionStr)
23    if err != nil {
24        orderPosition = 0
25    }
26
27    // Validate file
28    allowedFileType := false
29    for _, allowedType := range data.ALLOWED_IMAGE_TYPES {
30        if handler.Header.Get("Content-Type") == allowedType {
31            allowedFileType = true
32        }
33    }
```

```
33     }
34
35     if !allowedFileType {
36         error_handling.LogError(w, r, &data.AppError{
37             StatusCode: http.StatusNotAcceptable,
38             Text:       "Files of this type cannot be uploaded",
39             Description: nil,
40             GoError:   errors.New("unsupported file type"),
41         })
42         return
43     }
44
45     if handler.Size > int64(maxFilesize)*1000000 {
46         error_handling.LogError(w, r, &data.AppError{
47             StatusCode: http.StatusNotAcceptable,
48             Text:       "This file is too large",
49             Description: nil,
50             GoError:   errors.New("file is too large"),
51         })
52         return
53     }
54
55     // Upload file
56     resp, appError := app.UploadPicture(uint(itemId), file, description, orderPosition, *handler)
57     if appError != nil {
58         error_handling.LogError(w, r, appError)
59         return
60     }
```

H SQL-kode: Events

Dette er SQL-koden for å starte eventene som kjører spørringene for å håndtere visse databaseoperasjoner som trengs å kjøres med et fast tidsintervall. De settes opp til å kjøres daglig kl. 2 på natten slik at det skal ha en minimal påvirkning på ytelsen når brukerne bruker applikasjonen på dagtid.

H.1 Fjerning av ugyldige sesjoner

Denne eventen har som formål å fjerne gamle og ugyldige sesjoner

```
1 DELIMITER |
2 CREATE EVENT remove_old_sessions
3 ON SCHEDULE
4 EVERY 1 DAY
5 STARTS CONCAT(DATE(NOW()+INTERVAL 1 DAY ), ' 02:00:00')
6 DO
7 BEGIN
8 DELETE FROM sessions WHERE expiry_date < CURRENT_TIMESTAMP() OR is_valid=false;
9 END |
10 DELIMITER ;
```

H.2 Fjerning av gamle unntaksdatoer

Denne eventen fjerner gamle unntaksdatoer for åpningstidene, det vil si for datoer som allerede har vært.

```
1 DELIMITER |
2 CREATE EVENT remove_old_exceptions
3 ON SCHEDULE
4 EVERY 1 DAY
5 STARTS CONCAT(DATE(NOW()+INTERVAL 1 DAY ), ' 02:00:00')
6 DO
7 BEGIN
8 DELETE FROM exception_date WHERE date < CURRENT_DATE();
9 END |
10 DELIMITER ;
```

H.3 Automatisk kansellering av reserverasjoner

Reserverasjoner som ikke blir hentet skal automatisk kanselleres slik at utstyret som er reservert frigjøres for andre som ønsker å låne utstyret. Denne er noe mer komplisert enn de andre siden to tabeller må oppdateres (tabellen for utstyrsstørrelser og tabellen for lån og reserverasjoner).

```
1 DELIMITER |
2 CREATE EVENT remove_old_reservations
3 ON SCHEDULE
4 EVERY 1 DAY
5 STARTS CONCAT(DATE(NOW()+INTERVAL 1 DAY ), ' 02:00:00')
6 DO
7 BEGIN
8 START TRANSACTION;
9
10 UPDATE equipment_size
11 INNER JOIN
12 (
13     SELECT
14     equipment_size AS loan Equip_size, equipment_type_id AS
15     loan Equip_id,SUM(equipment_quantity) AS reserved_quantity
16     FROM
17     loan
18     WHERE
19     status="reserved" AND collection_date < CURRENT_DATE()
20     GROUP BY
21     equipment_size,equipment_type_id
22 ) AS quantity_tbl
23 ON
24 (equipment_size.size,equipment_size.equipment_type_id) =
25 (quantity_tbl.loan Equip_size,quantity_tbl.loan Equip_id)
26 SET
27 equipment_size.quantity = equipment_size.quantity + quantity_tbl.reserved_quantity;
28
29 UPDATE loan SET status="not_picked_up" WHERE
30 status="reserved" AND collection_date < CURRENT_DATE();
31
32 COMMIT;
33 END |
34 DELIMITER ;
```

I Utkast til API design

Dette er et utkast til design av APIet (utkastet ble laget før databasen var ferdig designet) som ble laget den 21/02.

Begynner på neste side

API utkast

Dette var et utkast til hvordan APIen kunne se ut ved begynnelsen av prosjektet. Det er utdatert i forhold til nåværende implementasjon, for oppdatert dokumentasjon se [API-dokumentasjon.md](#)

UtstyrsAPI

I eksemplene har jeg bare gjettest meg fram til hvordan de bør se ut mtp. at vi ikke har endelig databasestruktur, det vil derfor nok gjøres justeringer. Vi vil kanskje også trenge flere endepunkter i fremtiden. Sånn jeg ser det er det størst usikkerhet knyttet til håndteringen av størrelser, statistikk og revisjonsloggen.

Autentisering

/api/auth/login - Logg inn med egen løsning (tlf.nr. + passord) [POST]

Sender innloggingsdetaljer med JSON, eksempel:

```
{
  "phone_number": 12345678,
  "password": "passord1234"
}
```

(siden passordet sendes i klartekst til APIen bør vi bruke HTTPS på APIen vår så fort som mulig)

Svar (ved suksess)

"Logged in as \name\ successfully!" (kanskje som JSON?)

/api/auth/vipps - Logg inn med Vipps (OAUTH)?

Usikker på hvordan denne skal fungere, men her regner jeg med vi får en login/auth token fra Vipps som vi bruker for å dobbeltsjekke at brukeren er logget inn hos Vipps samt en unik verdi som knytter Vipps-brukeren dems med vår løsning. Dermed får bruker Access og Refresh JWT tokens som vanlig for å benytte tjenesten vår.

Svar (ved suksess)

"Logged in as \name\ using Vipps successfully!"

/api/auth/refresh - Hent en ny aksess token med en gyldig refreshtoken [GET]

Dersom brukeren har en gyldig session (via en gyldig refreshtoken) får brukeren en ny aksess token som brukes til resten av tjenesten.

Svar (ved suksess)

"Access token refreshed successfully!"

/api/auth/logout - Logg ut [POST]

Aktiv session blir ugyldiggjort og token cookies blir resatt hos klient. Dersom klient tar vare på sin aksess token kan bruker i teorien ha tilgang i noen minutter ekstra.

Svar (ved suksess)

"Logged out successfully!"

/api/auth/status - Status på autentisering [GET]

All data om autentisering til en bruker vises. Data lagret i JWT tokens (om de har gyldige tokens). I tillegg vises data lagret om session i database

Eksempel:

```
{
  "access_token": {
    "iss": "UtstyrsbankenAPI",
    "sub": "ab444ad2-c98a-11ee-bff5-5413795c52b3",
    "exp": 1707756943,
    "iat": 1707756643
  },
  "valid_access_token": true,
  "refresh_token": {
    "iss": "UtstyrsbankenAPI",
    "sub": "ab444ad2-c98a-11ee-bff5-5413795c52b3",
    "exp": 1708943557,
    "iat": 1707733957,
    "jti": "83d4787b-487c-4b1e-b936-6770f0061e27"
  },
}
```


API utkast

about:sredoc

```
"valid_refresh_token": true,
"session": {
  "token_id": "83d4787b-487c-4b1e-b936-6770f0061e27",
  "user_id": "ab444ad2-c98a-11ee-bff5-5413795c52b3",
  "issued_at": "2024-02-12T10:32:37Z",
  "expiry": "2024-02-26T10:32:37Z",
  "is_valid": true
},
"authenticated": true
}
```

Først får vi detaljene i en evt. gyldig aksesstoken (iss = issuer, sub = subject/bruker-ID, exp = expiry i Unix-sekunder, iat = issued at i Unix-sekunder), "valid_access_token" er `true` om aksesstoken er gyldig, `false` om aksesstoken er ugyldig (signatur, utløpsdato osv.) eller `null` dersom en aksesstoken ikke ble medsendt i forespørselen. Deretter følger samme informasjon om refresh token og tilslutt får vi se dataene om session i databasen (skal samsvare med refresh token). Sist får vi "authenticated" som viser om systemet anser brukeren som autentisert eller ikke (gyldig aksesstoken eller gyldig session)

Brukere

/api/users/register - Registrer bruker via egen løsning [POST]

Detaljer om ny bruker sendes som JSON:

```
{
  "name": "Ola Nordmann",
  "phone_number": 12345678,
  "password": "password"
}
```

(siden passordet sendes i klartekst til APIen bør vi bruke HTTPS på APIen vår så fort som mulig)

Bruker blir deretter satt inn i databasen med hashet passord, men som ikke aktivert. Bruker må aktiveres med SMS før den kan brukes. Dette endepunktet vil trenge en form for rate limiting siden sending av SMS koster penger.

Svar (ved suksess):

"User \id\ created successfully, but needs verification"

SMS med kode for verifisering sendes på oppgitt telefonnummer

/api/users/sendcode?user=<user-id> - Send aktiveringskode på SMS [GET]

Sender ny aktiveringskode på SMS, dersom f.eks. den gamle er ugyldig. Dette endepunktet vil trenge en form for rate limiting.

/api/users/verify - Verifiser bruker med kode fra SMS [PATCH]

Verifikasjonsdetaljer sendes som JSON:

```
{
  "phone_number": 12345678,
  "code": 124421
}
```

Dersom verifikasjonskoden stemmer med telefonnummeret/bruker som skal verifiseres blir bruker aktivert

Svar (ved suksess):

"User with number \phone_number\ verified and activated!"

/api/users/vippsreg - Registrer med Vipps?

Igjen usikker, men regner med det funker at backend får login/aksesstoken fra Vipps etter at bruker har gjennomført login-prosessen hos Vipps, samt en unik ID knyttet til Vipps-brukeren. Vi bruker så Vipps API til å hente nødvendig data (fullt navn, telefonnummer). Deretter registreres brukeren med IDen og de innhentede dataene (i tillegg må Vipps-flag være satt til true og passord er null)

/api/users/user/<user-id> - Hent detaljer om bruker med ID user-id [GET]

Detaljer om bruker med ID \user-id\ hentes. Vil antagelig kreve admin-privilegier.

Eksempel på svar:

```
{
  "id": 8,
}
```

API utkast

about:srcdoc

```
"name": "Ola Nordmann",
"phone_number": 12345678,
"password": "$2a$13$Bwgn/rqQRce17KBgTwNAukaMmSoC2hNlzN0ZH7iiSpgcm0TrfqvS",
"vipps": false,
"vipps_id": null,
"created": "2024-02-12T17:10:39Z",
"activation_code": 124421,
"activated": true
}
```

Dette endepunktet kan utvides til redigering via PATCH og sletting av brukere via DELETE.

Inventar

/api/inventory/create - Lag ny utstyrstype i inventaret [POST]

Detaljer om den nye utstyrstypen sendes som JSON, eksempel:

```
{
  "name": "Skistaver",
  "description": "Staver til langrennsski",
  "category": 3,
  "image": "/path/to/image.jpg",
  "size_type": "Integer",
  "size_unit": "cm"
}
```

Igjen, her er jeg usikker på hvordan vi endte opp med å håndtere størrelser.

Svar (ved suksess):

```
"Item \item-id\ created successfully!"
```

/api/inventory/list - Liste over alt inventar? [GET]

Detaljer om alt utstyr i en stor JSON-array? Vi kan evt. vurdere å utelate info om størrelser og heller kreve at frontend/klient kjører request for en spesifikk utstyrstype for denne informasjonen.

```
[
  {
    "id": 11
    "name": "Skistaver",
    "description": "Staver til langrennsski",
    "category": 3,
    "image": "/path/to/image.jpg",
    "size_type": "Integer",
    "size_unit": "cm",
    "sizes": [
      {
        "id": 48,
        "size_int": 80,
        "size_string": null,
        "size_sort": 80,
        "amount_avaliable": 3,
        "total_amount": 4
      },
      {
        "id": 49,
        "size_int": 90,
        "size_string": null,
        "size_sort": 90,
        "amount_avaliable": 2,
        "total_amount": 2
      },
      {
        "id": 50,
        "size_int": 100,
        "size_string": null,

```

3 of 9

07/05/2024, 19:58

API utkast

about:srcdoc

```
    "size_sort": 100,
    "amount_avaliable": 0,
    "total_amount": 5
  }
]
}
```

/api/inventory/list?category=<category-id> - Liste over alt inventar i kategori med ID category-id [GET]

Dataene man får vil være den samme som forrige endepunkt, bare filtrert etter kategori

/api/inventory/item/ - Hent detaljer om gjenstand/utstyrstype med ID item-id [GET]

Dataene man får vil være den samme som forrige endepunkt, bare denne gang, kun om en spesifikk utstyrstype. Dette kan være nyttig om vi låser detaljer om størrelser og antall tilgjengelig bak dette endepunktet og ikke inkluderer det i list-endepunktet. Dessuten kan dette endepunktet utvides til å kunne brukes til modifikasjon (PATCH) og sletting utstyrtyper (DELETE).

/api/inventory/item/<item-id>/new - Legg inn en ny størrelse til utstyrstypen [POST]

Data om den nye størrelsen sendes inn som JSON:

```
{
  "size_int": null,
  "size_string": "M",
  "size_sort": 20,
  "total_amount": 3
}
```

Siden dette er en ny størrelse vil f.eks. "amount_avaliable" settes til det samme som "total_amount" (dette må nok endres basert på databasestrukturen til Erlend?)

Svar (ved suksess):

"Size \size-id\ for item \item-id\ created successfully!"

/api/inventory/item/<item-id>/ - Endre/modifiser dataene om en størrelse [PATCH]

Dersom f.eks. mer utstyr i samme størrelse skal legges inn kan dette oppdateres med en PATCH-forespørsel med samme typen data som for å lage nye størrelser.

Kategorier (til inventaret)

/api/categories/create - Lag ny kategori [POST]

Lag en ny kategori, dataene sendes som JSON:

```
{
  "name": "Skiutstyr",
  "parent_category": 4
}
```

I eksemplet over ser jeg for meg at vi har kategorier i et hierarki og "parent_category" inneholder kategori-ID til kategorien over denne.

Svar (ved suksess):

"Category \category-id\ created successfully!"

/api/categories/list - Liste over alle kategorier [GET]

Lister opp alle kategorier og sender det i JSON-format, noe ala dette:

```
[
  {
    "id": 1,
    "name": "Ballsport",
    "subcategories": [
      {
        "id": 2,
        "name": "Fotball",
        "subcategories": null
      },
      {
```

4 of 9

07/05/2024, 19:58

API utkast

about:sredoc

```
    "id": 3,
    "name": "Håndball",
    "subcategories": null
  }
],
},
{
  "id": 4,
  "name": "Vinterutstyr",
  "subcategories": [
    {
      "id": 5,
      "name": "Ski",
      "subcategories": [
        {
          "id": 6,
          "name": "Langrenn",
          "subcategories": null
        },
        {
          "id": 7,
          "name": "Slalom",
          "subcategories": null
        }
      ]
    }
  ]
}
]
```

Litt usikker på hvordan databasen skal se ut og hvilke SQL-spøringer jeg skal bruke om vi i det hele tatt skal ha en slik struktur? Men, man skal ihvertfall få ut alle kategoriene.

/api/categories/category/<category-id> - Hent detaljer om kategori med ID category-id [GET]

Litt samme som over, bare med utgangspunkt i en av kategoriene. Sånn f.eks. om forespørselen er for kategori med ID 4:

```
{
  "id": 4,
  "name": "Vinterutstyr",
  "parent_category": null,
  "subcategories": [
    {
      "id": 5,
      "name": "Ski",
      "subcategories": [
        {
          "id": 6,
          "name": "Langrenn",
          "subcategories": null
        },
        {
          "id": 7,
          "name": "Slalom",
          "subcategories": null
        }
      ]
    }
  ]
}
```

Kan utvides til å støtte endringer (PATCH) og sletting (DELETE) av kategorier. Utstyr under disse kategoriene skal ikke fjernes, kategorien blir satt til `null`?

Reservasjon

/api/reservations/create - Lag ny reservasjon [POST]

Lager ny reservasjon på noe utstyr, detaljene sendes som JSON

```
{
  "user_id": 8,
  "from_date": "2024-02-20T00:00:00Z",
  "to_date": "2024-02-27T00:00:00Z",
  "items": [
    {
      "item_id": 11,
      "size_id": 49,
      "amount": 2
    },
    {
      "item_id": 12,
      "size_id": 58,
      "amount": 2
    }
  ]
}
```

(man sender med bruker ID da ansatte kan lage reservasjoner for andre brukere? selvfølgelig kan vanlige brukere kun lage reservasjoner for egen bruker)

Systemet vil sjekke om datoene er gyldige (innenfor kravet om maks antall dager frem i tid, samt gjeldene åpningstider) og evt. krav om maks antall reservasjoner, før den legges inn i databasen.

Svar (ved suksess):

"Reservation \reservation-id\ registered on user \user-id\ successfully!"

/api/reservations/reservation/<reservation-id> - Hent detaljer om en reservasjon med ID reservation-id [GET]

Detaljene om reservasjonen sendes til klient som JSON og blir noe sånt som dette

```
{
  "reservation_id": 83,
  "user_id": 8,
  "from_date": "2024-02-20T00:00:00Z",
  "to_date": "2024-02-27T00:00:00Z",
  "items": [
    {
      "item_id": 11,
      "size_id": 49,
      "amount": 2
    },
    {
      "item_id": 12,
      "size_id": 58,
      "amount": 2
    }
  ],
  "picked_up": false,
  "cancelled": false
}
```

/api/reservations/list?user=<user-id> - Liste over reservasjoner registrert på bruker med ID user-id [GET]

Her listes alle aktive reservasjoner (ikke "picked_up" eller "cancelled") som en bruker har for øyeblikket. Dataene ser ut som i endepunktet over.

Lån

/api/loans/create - Registrer nytt lån [POST]

Lager nytt lån på noe utstyr, detaljene sendes som JSON

```
{
  "user_id": 8,
  "from_date": "2024-02-20T00:00:00Z",
  "to_date": "2024-02-27T00:00:00Z",
}
```

API utkast

about:sredoc

```
"items": [
  {
    "item_id": 11,
    "size_id": 49,
    "amount": 2
  },
  {
    "item_id": 12,
    "size_id": 58,
    "amount": 2
  }
]
```

Svar (ved suksess):

"Loan \loan-id\ created successfully for user \user-id"

/api/loans/register - Registrer nytt lån fra reservasjon [POST]

Denne er ganske enkel, man sender kun følgende som JSON:

```
{
  "reservation_id": 81
}
```

Så vil systemet hente opp reservasjonen og lage et lån basert på dette, reservasjonen vil også få `picked_up` satt til `true`.

Svar (ved suksess):

"Loan \loan-id\ from reservation \reservation-id\ created successfully for user \user-id"

/api/loans/loan/<loan-id> - Hent detaljer om lån med ID loan-id [GET]

Dataene sendes som JSON og vil være ganske lik en reservasjon

```
{
  "loan_id": 49,
  "user_id": 8,
  "from_date": "2024-02-20T00:00:00Z",
  "to_date": "2024-02-27T00:00:00Z",
  "items": [
    {
      "item_id": 11,
      "size_id": 49,
      "amount": 2
    },
    {
      "item_id": 12,
      "size_id": 58,
      "amount": 2
    }
  ],
  "extended": false
}
```

/api/loans/list?user=<user-id> - Liste over lån registrert på bruker med ID user-id [GET]

Lister alle aktive/nåværende lån på brukeren, dataene vil se ut som i eksemplet over

/api/loans/history?user=<user-id> - Se over lånehistorikk for bruker med ID user-id [GET]

Lister alle historiske/tilbakeleverte lån på brukeren, dataene ser ut som eksemplet for ett spesifikt lån.

/api/loans/return?loan=<loan-id> - Registrer lånet som levert tilbake [PATCH?]

Registrerer lånet med ID \loan-id\ som tilbakelevert, dvs. sletting av lånet i den vanlige tabellen og opprettelsen av en kopi i lånehistorikktabellen, samt oppdatering av antall tilgjengelig av alt utstyr i lånet.

Svar (ved suksess):

"Loan \loan-id\ has been returned successfully!"

7 of 9

07/05/2024, 19:58

Statistikk & annet

/api/status - Hent statusinfo (opetid, aktive tokens?, forbindelse med Vipps-API?, databaseforbindelse) [GET]

Returnerer status for tjenesten, til de som drifter tjensten, eksempel:

```
{
  "uptime_seconds": 85482,
  "active_sessions": 98,
  "vipps_status": 200,
  "database": "OK"
}
```

/api/stats/item?id=<item-id>&from=<from-date>&to=<to-date> - Hent statistikk om gjenstand med ID item-id i perioden from-date - to-date [GET]

Henter lånestatistikk for gjenstanden (gjenstand inkludert i lån) som JSON, eksempel:

```
{
  "loans": 81,
  "sizes": [
    {
      "size_id": 49,
      "loans": 30,
    },
    {
      "size_id": 50,
      "loans": 21,
    },
    {
      "size_id": 51,
      "loans": 13,
    },
    {
      "size_id": 52,
      "loans": 17,
    }
  ]
}
```

Her er det nok ønskelig å inkludere data om gjenstanden/utstyrstypen, men dette kan hentes gjennom eget endepunkt

/api/stats/category?id=<category-id>&from=<from-date>&to=<to-date> - Hent statistikk om kategori med ID category-id i perioden from-date - to-date [GET]

Samme som forrige, bare på kategorinivå istedet, man vil også kunne se antall lån for subkategorier i stedet for størrelser som for utstyrstyper.

/api/log?from=<from-date>&to=<to-date> - Hent revisjonsloggen for perioden from-date - to-date [GET]

Henter opp revisjonsloggen for den gitte perioden, vil se noe ala sånn her ut

```
[
  {
    "entry_id": 95201,
    "timestamp": "2024-02-13T12:10:49Z",
    "user_id": 4,
    "interaction_id": 49,
    "interaction_type": "loan",
    "description": "User 4 created loan 49"
  },
  {
    "entry_id": 95202,
    "timestamp": "2024-02-13T12:17:11Z",
    "user_id": 4,
    "interaction_id": 50,
    "interaction_type": "loan",
    "description": "User 4 created loan 50"
  }
]
```

API utkast

about:srcdoc

```
}  
|
```

Usikker på om databasestrukturen kan tillate en slik struktur på loggen, men i dette tilfellet refererer `interaction_id` til hva som skjer med, f.eks. ved et nytt lån vil dette være IDen til det nye lånet, ved sletting av en reservasjon vil dette være IDen til reservasjonen osv.

/api/log?limit=<limit> - Hent de nyeste lines oppføringer fra revisjonsloggen [GET]

Samme som forrige, bare med de nyeste oppføringene i loggen istedet.

J API dokumentasjon

Her dokumenteres alle endepunktene som finnes i APIen. Dette inkluderer URL, HTTP-metoder, hvilke parametre som kan brukes, struktur på eventuell JSON-data samt eksempler på responser. Endepunktene er sortert:

- Autentisering (`/api/auth`): Se J.1
- Inventar (`/api/inventory`) og kategorier (`/api/category`): Se J.2
- Reservasjoner (`/api/reservations`) og lån (`/api/loans`): Se J.3
- Brukerhåndtering (`/api/users`): Se J.4
- Bildehåndtering (`/api/images`): Se J.6
- Diverse (åpningstider, konfigurable verdier, statistikk, rolleinfo): Se J.5

J.1 Autentisering

Autentisering

about:srcdoc

Autentisering

/api/auth/login [POST] Innlogging

Send detaljer via JSON, noe sånt som:

```
{
  "phone_number": 12344321,
  "password": "Passord123"
}
```

Dersom det finnes en bruker med det spesifiserte telefonnummeret og passordet vil bruker logges inn. En sesjon blir opprettet i databasen og brukeren får to cookies med "AccessToken" og "RefreshToken".

Respons (ved suksess) fra dette endepunktet ser ut som dette:

```
{
  "text": "Login successful!",
  "access_token_expiry": "2024-02-26T17:21:34.782353065+01:00",
  "session_expiry": "2024-03-11T17:16:34.782353578+01:00",
  "user_id": 9,
  "first_name": "Ola",
  "last_name": "Nordmann",
  "role_id": 0,
  "role_name": "User"
}
```

/api/auth/logout [POST] Utlogging

Utlogging skjer ved å sende en tom POST-forespørsel mot dette endepunktet. Da bes klient om å fjerne cookieene for "AccessToken" og "RefreshToken". Sesjon tilknyttet "RefreshToken" blir også ugyldiggjort.

Respons (ved suksess) fra dette endepunktet ser ut som dette:

```
{
  "text": "User 9 logged out successfully!",
  "user_id": 9
}
```

/api/auth/refresh [GET] Generering av nye "AccessTokens"

Dersom man har en gyldig "RefreshToken" (og den tilknyttede sesjonen er gyldig) kan man bruke den til å generere nye "AccessTokens". Dette vil erstatte eventuelt eksisterende cookie med den nye. Den nye "AccessToken" vil være gyldig i fem minutter fra da den ble generert.

Respons (ved suksess) fra dette endepunktet ser ut som dette:

```
{
  "text": "Access token refreshed successfully!",
  "access_token_expiry": "2024-02-26T17:28:25.267488238+01:00"
}
```

/api/auth/whoami [GET] Hent detaljer om innlogget bruker (via medsendt **AccessToken** cookie)

PERM: Dette endepunktet krever ingen tilganger

Henter detaljer (bruker-ID, navn, rolleinfo) om innlogget bruker dvs. fra medsendt **AccessToken**

Svar (ved suksess):

```
{
  "user_id": 1,
  "first_name": "Åsulv",
  "last_name": "Vå",
  "role_id": 3,
  "role_name": "Administrator",
  "is_default": false,
}
```

```
{
  "is_guest": false,
  "is_employee": false,
  "is_admin": true
}
```

/api/auth/whoami?strict=true [GET] Hent detaljer om innlogget bruker (via medsendt **RefreshToken** cookie)

PERM: Dette endepunktet krever ingen tilganger

Samme som endepunktet over, men bruker **RefreshToken** cookie for å hente detaljer istedetfor **AccessToken** .

/api/auth/resetpw?number={phone_number} [POST] Start passord reset

Perms: Krever ingen ekstra tilganger.

Limiter: SMS-limiter: En IP-adresse kan sende opptil 3 forespørsler hvert 3. minutt (kan konfigureres, dette er bare defaults)

Dette endepunktet starter passord reset-prosessen som beskrevet over.

Svar (ved suksess):

```
{
  "text": "Started password reset for user with number {phone_number}"
}
```

/api/auth/resetpw?number={phone_number} [PATCH] Utfør passord reset

Perms: Krever ingen ekstra tilganger.

Limiter: SMS-limiter: En IP-adresse kan sende opptil 3 forespørsler hvert 3. minutt (kan konfigureres, dette er bare defaults)

Dette endepunktet setter nytt passord til brukeren ved hjelp av koden fra SMS-en. Kode og nytt passord sendes med som JSON:

```
{
  "code": 611774,
  "new_password": "nYtt pa$$0rD"
}
```

Svar (ved suksess):

```
{
  "text": "Password reset for user with number {phone_number} successfully!"
}
```

J.2 Inventar og kategorier

Inventar og kategorier

about:srcdoc

Inventar og kategorier

/api/inventory/create [POST] Lag ny gjenstand i inventaret

PERM: Krever "write"-tilgang på ressursen "Inventory"

Detaljer om den nye gjenstanden sendes med som JSON:

```
{
  "name": "Langrensski",
  "description": "Ski til langrenn.",
  "max_quantity": 6,
  "order_position": 10,
  "categories": [3,4]
}
```

Feltet "unit" er størrelsesenheten til utstyrstypen. Feltet "max_quantity" (maks antall en bruker kan reservere av denne utstyrstypen) er valgfri, dersom det ikke sendes med antas det at gjenstanden ikke har en "max_quantity". Feltet "categories" er en array av IDer til kategoriene gjenstanden hører til, dersom det ikke sendes med antas det at gjenstanden er ukategorisert.

Svar (ved suksess):

```
{
  "text": "Item Langrensski with ID 7 created successfully!",
  "item": {
    "id": 7,
    "name": "Langrensski",
    "description": "Ski til langrenn.",
    "unit": "cm",
    "max_quantity": 6,
    "removed": false,
    "order_position": 10,
    "categories": [
      3,
      4
    ]
  }
}
```

/api/inventory/list [GET] Hent alle gjenstander i inventaret

PERM: Krever "view"-tilgang på ressursen "Inventory"

Alle gjenstander/utstyrstyper i databasen returneres (data om størrelsene er ikke inkludert)

```
[
  {
    "id": 1,
    "name": "Langrensski",
    "description": "",
    "max_quantity": 6,
    "removed": false,
    "order_position": 10,
    "categories": [
      3
    ]
  },
  {
    "id": 2,
    "name": "Skisko",
    "description": "Sko med binding som passer til langrensski",
    "max_quantity": 6,
    "removed": false,
    "order_position": 11,
    "categories": [
      3,

```

```
    4
  ]
},
{
  "id": 3,
  "name": "Skistaver",
  "description": "Staver til bruk for langrenn",
  "max_quantity": 6,
  "removed": false,
  "order_position": 12,
  "categories": [
    3,
    4
  ]
},
{
  "id": 7,
  "name": "Langrennsski",
  "description": "Ski til langrenn.",
  "max_quantity": 6,
  "removed": false,
  "order_position": 13,
  "categories": [
    3,
    4
  ]
}
]
```

/api/inventory/item/ Gjenstand-ende punkt

/api/inventory/item/{gjenstand-id} **[GET]** Hent detaljer om en gjenstand

PERM: Krever "view"-tilgang på ressursen "Inventory"

Detaljene om gjenstanden/utstyrstypen sendes til klient (inkluderer størrelser og antallet tilgjengelig av disse):

```
{
  "id": 1,
  "name": "Langrennsski",
  "description": "",
  "max_quantity": null,
  "removed": false,
  "order_position": 10,
  "categories": [],
  "sizes": [
    {
      "size": "150 cm",
      "quantity": 3,
      "order_id": 150,
      "removed": false
    },
    {
      "size": "160 cm",
      "quantity": 2,
      "order_id": 160,
      "removed": false
    },
    {
      "size": "170 cm",
      "quantity": 4,
      "order_id": 150,
      "removed": false
    }
  ]
}
```

/api/inventory/item/{gjenstand-id} [PATCH] Endre detaljer om en gjenstand

PERM: Krever "edit"-tilgang på ressursen "Inventory"

Dataene som skal endres sendes som JSON:

```
{
  "description": "ny beskrivelse"
  "categories": [4]
}
```

Kun feltene som sendes med blir endret, resten beholder sin nåværende verdi. Følgende felt er mulig å redigere via dette endepunktet: "name","description","max_quantity","order_position","categories"

I tillegg kan man sende med feltet "reset_max_quantity" satt til `true` for å sette "max_quantity" til `NULL` dersom man ønsker å fjerne "max_quantity".

Svar (ved suksess):

```
{
  "text": "Item Skisko with ID 2 modified successfully!",
  "item": {
    "id": 2,
    "name": "Skisko",
    "description": "ny beskrivelse",
    "max_quantity": 6,
    "removed": false,
    "order_position": 11,
    "categories": [
      4
    ]
  }
}
```

/api/inventory/item/{gjenstand-id} [DELETE] Slett en gjenstand fra inventaret

PERM: Krever "delete"-tilgang på ressursen "Inventory"

Gjenstanden "slettes", dvs. flagget "removed" blir satt til `true` ("soft-deletion")

Svar (ved suksess):

```
{
  "text": "Item with ID 7 removed successfully!"
}
```

/api/inventory/item/{gjenstand-id}/new [POST] Lag ny størrelse til en gjenstand/utstyrstype

PERM: Krever "write"-tilgang på ressursen "Inventory"

Detaljene om den nye størrelsen sendes med som JSON:

```
{
  "size": "170 cm",
  "quantity": 4,
  "order_id": 170
}
```

Svar (ved suksess):

```
{
  "text": "Created size 170 cm for item with ID 1 successfully!",
  "item_size": {
    "size": "170 cm",
    "quantity": 4,
    "order_id": 170,
    "removed": false
  }
}
```

```
}
```

/api/inventory/item/{gjenstand-id}/{størrelse} [PATCH] Endre detaljer om en størrelse til en gjenstand/utstyrstype

PERM: Krever "edit"-tilgang på ressursen "Inventory"

Feltene som skal endres sendes med som JSON:

```
{  
  "quantity": 5  
}
```

Felter som ikke sendes med endres ikke, følgende felter kan endres: "quantity", "order_id" (ideen er at denne kan utvides i framtiden om størrelser får flere felter)

Svar (ved suksess):

```
{  
  "text": "Modified size 170 cm for item with ID 1 successfully!",  
  "item_size": {  
    "size": "170 cm",  
    "quantity": 5,  
    "order_id": 170,  
    "removed": false  
  }  
}
```

/api/inventory/item/{gjenstand-id}/{størrelse} [DELETE] Slett en størrelse til en gjenstand/utstyrstype

PERM: Krever "delete"-tilgang på ressursen "Inventory"

Størrelsen slettes fra databasen ("hard-delete")

Svar (ved suksess):

```
{  
  "text": "Size 170 cm of item with ID 1 deleted successfully"  
}
```

/api/categories/create [POST] Lag ny kategori

PERM: Krever "write"-tilgang på ressursen "Inventory"

Detaljene om den nye kategorien sendes som JSON:

```
{  
  "name": "Ballsport"  
}
```

Svar (ved suksess):

```
{  
  "text": "Category Ballsport with ID 5 created successfully!",  
  "category": {  
    "id": 5,  
    "name": "Ballsport"  
  }  
}
```

/api/categories/list [GET] Se liste over alle kategoriene

PERM: Krever "view"-tilgang på ressursen "Inventory"

Kategoriene sendes tilbake som JSON:

```
[  
  {
```

```
    "id": 3,
    "name": "Langrenn"
  },
  {
    "id": 4,
    "name": "Skitilbehør"
  },
  {
    "id": 5,
    "name": "Ballspport"
  }
]
```

/api/categories/category/ Kategori-endepunktet

/api/categories/category/{kategori-id} [PATCH] Endre detaljer om en kategori

PERM: Krever "edit"-tilgang på ressursen "Inventory"

Detaljene som skal endres sendes med som JSON:

```
{
  "name": "Fotball"
}
```

Kun feltene som sendes med endres, feltet som kan endres er: "name" (ideen er at denne kan utvides i framtiden om kategorier får flere feltet)

Svar (ved suksess):

```
{
  "text": "Category Fotball with ID 5 modified successfully!",
  "category": {
    "id": 5,
    "name": "Fotball"
  }
}
```

/api/categories/category/{kategori-id} [DELETE] Slett en kategori

PERM: Krever "delete"-tilgang på ressursen "Inventory"

Kategorien slettes fra databasen ("hard-delete"), alle linker mellom kategorien og gjenstander/utstyrstyper fjernes også automatisk pga. CASCADE

Svar (ved suksess):

```
{
  "text": "Category with ID 5 deleted successfully!"
}
```


J.3 Lån og reserverasjoner

Lån og reserverasjoner

about:srcdoc

Lån og reserverasjoner

/api/Loans/create [POST] Opprett nytt lån

PERM: Krever write-tilgang på Loan-ressursen

Data om det nye lånet sendes med som JSON:

```
{
  "loaner_id": 2,
  "return_date": "2024-03-21T00:00:00Z",
  "equipment_type_id": 4,
  "equipment_size": "180 cm",
  "equipment_quantity": 1
}
```

loaner_id er IDen til bruker som lånet skal registreres på

Da vil backend sjekke om det finnes utstyr slik at lånet kan utføres og deretter redusere antallet tilgjengelig av utstyr i den spesifiserte størrelsen og sette inn lånet.

HUSK: Dette endepunktet kjøres når et lån blir laget dvs. utstyret blir fysisk lånt ut til brukeren, derfor settes collection_date til nåværende dato.

Svar (ved suksess):

```
{
  "text": "Loan (current) with ID 19 created for user with ID 2 successfully!",
  "loan": {
    "id": 19,
    "loaner_id": 2,
    "reserved_date": "2024-03-15T15:17:59.429534059+01:00",
    "collection_date": "2024-03-15T00:00:00Z",
    "return_date": "2024-03-21T00:00:00Z",
    "status": "current",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "180 cm",
    "equipment_quantity": 1
  }
}
```

Lag lån fra reserverasjon

Du kan også passere en eksisterende reserverasjon med dette endepunktet. For å gjøre poster du ID på eksisterende reserverasjon mot dette samme endepunktet med følgende body:

Eksempel:

```
{
  "from_reservation": 3, // Reserverasjons ID
  "use_reservation": true
}
```

Svar vil være på samme format

Lag lån og fjern reserverasjonen

Dette brukes dersom lånet avviker fra reserverasjonen (eks. bruker ønsker å låne flere ski enn i reserverasjonen). Da må man sende alle detaljer om lånet samt reserverasjons IDen til reserverasjonen som skal kanselleres i from_reservation og use_reservation satt til false.

Eksempel:

```
{
  "loaner_id": 2,
  "return_date": "2024-03-21T00:00:00Z",
  "equipment_type_id": 4,
  "equipment_size": "180 cm",
  "equipment_quantity": 1,
  "from_reservation": 3,
  "use_reservation": false
}
```

```
"from_reservation": 3, // Reserverasjons ID
"use_reservation": false
}
```

Svar vil være på samme format

/api/loans/history?user={bruker-id} [GET] Hent lånehistorikken til en bruker

PERM: Krever view-tilgang på LoanSelf om man ser på egne lån, krever view-tilgang på Loan om man ser på andres lån.

Henter lånehistorikken til en bruker dvs. alle ferdigleverte lån.

Svar (ved suksess):

```
[
  {
    "id": 17,
    "loaner_id": 2,
    "reserved_date": "2024-01-01T00:00:00Z",
    "collection_date": "2024-01-02T00:00:00Z",
    "return_date": "2024-01-09T00:00:00Z",
    "status": "finished",
    "delivered_date": "2024-01-09T00:00:00Z",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "180 cm",
    "equipment_quantity": 1
  },
  {
    "id": 18,
    "loaner_id": 2,
    "reserved_date": "2024-02-10T00:00:00Z",
    "collection_date": "2024-02-15T00:00:00Z",
    "return_date": "2024-02-20T00:00:00Z",
    "status": "finished",
    "delivered_date": "2024-02-20T00:00:00Z",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "180 cm",
    "equipment_quantity": 1
  }
]
```

/api/loans/list?user={bruker-id} [GET] Hent alle aktive lån til en bruker

PERM: Krever view-tilgang på LoanSelf om man ser på egne lån, krever view-tilgang på Loan om man ser på andres lån.

Henter alle aktive lån til en bruker.

Svar (ved suksess):

```
[
  {
    "id": 19,
    "loaner_id": 2,
    "reserved_date": "2024-03-15T00:00:00Z",
    "collection_date": "2024-03-15T00:00:00Z",
    "return_date": "2024-03-21T00:00:00Z",
    "status": "current",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "180 cm",
    "equipment_quantity": 1
  }
]
```

/api/loans/loan/ Lån-ende punkt

Brukes til å samhandle med et enkelt lån

/api/loans/loan/{lån-id} [GET] Se på et enkeltlån

PERM: Krever view-tilgang på LoanSelf om man ser på egne lån, krever view-tilgang på Loan om man ser på andres lån.

Detaljer om lånet sendes som JSON:

```
{
  "id": 18,
  "loaner_id": 2,
  "reserved_date": "2024-02-10T00:00:00Z",
  "collection_date": "2024-02-15T00:00:00Z",
  "return_date": "2024-02-20T00:00:00Z",
  "status": "finished",
  "delivered_date": "2024-02-20T00:00:00Z",
  "times_delayed": 0,
  "equipment_type_id": 4,
  "equipment_size": "180 cm",
  "equipment_quantity": 1
}
```

MERK: Alle typer lån kan hentes med dette endepunktet, dette inkluderer reserverasjoner siden internt regnes det som en type lån.

/api/loans/loan/{lån-id} [PATCH] Endre på/utvid et lån

Man kan kun endre på aktive lån, dvs. status er `current`

Som ansatt/admin (med edit-tilgang på ressursen `Loan`)

Følgende verdier ved lånet kan endres: `return_date`, `loaner_id`

MERK: Siden man er ansatt/admin kan man fritt sette `return_date` uavhengig av lengde på lånet og om Utstyrsbanken er registrert som åpen, `times_delayed` vil heller ikke telles opp. Eneste krav for `return_date` er at den ikke er i fortida og etter `collection_date`.

Eksempel, flytt lånet til bruker med ID 2 og utvid lånet til torsdag 25.04.

```
{
  "loaner_id": 2,
  "return_date": "2024-04-25T00:00:00Z"
}
```

Svar (ved suksess):

```
{
  "text": "Successfully modified loan with ID 22!",
  "loan": {
    "id": 22,
    "loaner_id": 2,
    "reserved_date": "2024-04-18T00:00:00Z",
    "collection_date": "2024-04-18T00:00:00Z",
    "return_date": "2024-04-25T00:00:00Z",
    "status": "current",
    "times_delayed": 0,
    "equipment_type_id": 6,
    "equipment_size": "180 cm",
    "equipment_quantity": 1
  }
}
```

Som vanlig bruker (med edit-tilgang på ressursen `LoanSelf`)

Dersom man redigerer et lån som en vanlig bruker vil det være for å utvide lånet, dvs. man kun kan redigere `return_date`, dette vil også telle opp lånets `times_delayed`. Følgende krav stilles for å kunne utvide et lån:

- Lånets `times_delayed` er ikke høyere enn den konfigurable verdien `loan_max_times_delayed`, dvs. at brukeren har ikke utvidet lånet mer enn tillatt.
- Lånet har ikke forfalt dvs. `return_date` er i dag eller i fremtida.
- Den nye `return_date` er etter den gamle `return_date`

- Den nye `return_date` er ikke i fortiden
- Den nye `return_date` er en dag der Utstyrsbanken er åpen samt er innenfor maks dager man kan utvide lånet dvs. den konfigurable verdien `loan_extension_max_days`

Struktur på JSON som sendes med samt svar ligner på det over.

/api/Loans/loan/{lån-id} [DELETE] Slett et lån

PERM: Krever delete-tilgang på ressursen `Loan`

Sletter lånet fra databasen, dette er IKKE det samme som å returnere lånet:

- Lånet blir fjernet fra databasen, ikke satt til `finished`
- Antall tilgjengelig utstyr av størrelsen som ble lånt økes ikke

Use casen er dersom det oppstår desync mellom lån og antall tilgjengelig utstyr i databasen

Svar (ved suksess):

```
{
  "text": "Loan with ID 21 deleted successfully!"
}
```

/api/Loans/overdue [GET] Se en liste over alle lån som har forfalt

PERM: Krever view-tilgang på `Loan`.

Sender alle lån som har forfalt (lån som låner ikke har levert i tide):

```
[
  {
    "id": 22,
    "loaner_id": 2,
    "reserved_date": "2024-03-04T00:00:00Z",
    "collection_date": "2024-03-05T00:00:00Z",
    "return_date": "2024-03-12T00:00:00Z",
    "status": "current",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "190 cm",
    "equipment_quantity": 1
  }
]
```

/api/Loans/return?loan={lån-id} [PATCH] Returner et lån

PERM: Krever edit-tilgang på `Loan`.

Registrerer et lån som returnert/ferdig og øker antallet tilgjengelig for utstyrstypen i inventaret. Nåværende dato registreres som levert tilbake dato (`delivered_date`) for lånet.

Man må sende JSON til dette endepunktet. Her kan man sende `unrecoverable_items` som er antall gjenstander som ikke kunne returneres av ulike grunner (ødelagt, mistet osv.). `unrecoverable_items` vil da trekkes fra antallet som legges til størrelsen av utstyrstypen ved leveringen. Dersom man ikke sender med `unrecoverable_items` antas denne lik 0 dvs. at alle gjenstandene kunne returneres.

Eksempel dersom alle gjenstandene kunne returneres (tomt JSON objekt):

```
{}
```

Eksempel dersom 2 gjenstander ikke kunne returneres:

```
{
  "unrecoverable_items": 2
}
```

- Dersom `unrecoverable_items` er satt til et antall høyere enn antallet i lånet vil det antas at ingen gjenstander i lånet kunne returneres
- Alt annet ved endepunktet er som før, inkludert respons

Svar (ved suksess):

```
{
  "text": "Loan with ID 19 registered as returned and finished successfully!"
}
```

/api/loans/today [GET] Se en liste over alle lån som skal returneres idag

PERM: Krever view-tilgang på Loan.

Sender en liste med lån som skal returneres/leveres tilbake idag, dvs. return_date = nåværende dato. Dataene vil ha samme struktur som de andre endepunktene som henter en liste over lån.

/api/loans/issues [GET] Hent alle lån med potensielle problemer

PERM: Krever view-tilgang på ressursen [Loan](#)

Med problemer så menes følgende:

- return_date er satt til en dato der Utstyrskontrollen holder stengt
- Lengden på lånet er lengre enn det som i utgangspunktet er tillatt

Denne infoen kan være nyttig etter man f.eks. har oppdatert åpningstidene.

Svar (ved suksess):

```
[
  {
    "id": 5,
    "loaner_id": 2,
    "reserved_date": "2024-04-15T00:00:00Z",
    "collection_date": "2024-04-18T00:00:00Z",
    "return_date": "2024-04-25T00:00:00Z",
    "status": "current",
    "times_delayed": 0,
    "equipment_type_id": 2,
    "equipment_size": "170 cm",
    "equipment_quantity": 1,
    "valid_collection_date": true,
    "valid_return_date": false,
    "loan_length": 7
  }
]
```

I eksempelet over har lånet ikke en gyldig return_date.

/api/reservations/create [POST] Lag ny reserverasjon

PERM: Krever write-tilgang på ReservationSelf dersom man lager en reserverasjon til egen bruker, krever write-tilgang på Reservation dersom man lager en reserverasjon til en annen bruker.

Detaljene om den nye reserverasjonen sendes med som JSON:

```
{
  "loaner_id": 1,
  "collection_date": "2024-03-19T00:00:00Z",
  "return_date": "2024-03-21T00:00:00Z",
  "equipment_type_id": 4,
  "equipment_size": "200 cm",
  "equipment_quantity": 1
}
```

MERK: Dersom en bruker lager en reserverasjon til egen bruker vil systemet sjekke om datoene er gyldige (Utsstyrskontrollen er åpen de dagene, innenfor maksimumsgrensene på hvor langt det er i fremtiden) og om det er innenfor maksimumsgrensningene på antall av utstyrstypen som kan lånes/reserveres samtidig. Dersom en bruker lager en reserverasjon til en annen bruker blir ikke disse sjekkene gjort. Dette er for å gjøre det mulig for de ansatte på Utsstyrskontrollen å kunne lage reserverasjoner som ellers ville blitt brutt med reglene (f.eks. om en person vil låne flere ski en det som vanligvis er tillatt).

Svar (ved suksess):

```
{
  "text": "Loan (reserved) with ID 23 created for user with ID 1 successfully!",
  "loan": {
    "id": 23,
    "loaner_id": 1,
    "reserved_date": "2024-03-15T17:16:39.999173553+01:00",
    "collection_date": "2024-03-19T00:00:00Z",
    "return_date": "2024-03-21T00:00:00Z",
    "status": "reserved",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "200 cm",
    "equipment_quantity": 1
  }
}
```

/api/reservations/list?user={bruker-id} [GET] Liste over alle reserverasjoner til en bruker

PERM: Krever view-tilgang på **ReservationSelf** dersom man ser på reserverasjonene til egen bruker, krever view-tilgang på **Reservation** dersom man ser på reserverasjonene til en annen bruker.

Svar (ved suksess):

```
[
  {
    "id": 23,
    "loaner_id": 1,
    "reserved_date": "2024-03-15T00:00:00Z",
    "collection_date": "2024-03-19T00:00:00Z",
    "return_date": "2024-03-21T00:00:00Z",
    "status": "reserved",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "200 cm",
    "equipment_quantity": 1
  }
]
```

Reserverasjons-endepunktet **/api/reservations/reservation**

Dette endepunktet brukes for å samhandle med en enkelt reserverasjon

/api/reservations/reservation/{reserverasjons-id} [GET] Se på en enkelt reserverasjon

Svar (ved suksess):

```
{
  "id": 23,
  "loaner_id": 1,
  "reserved_date": "2024-03-15T00:00:00Z",
  "collection_date": "2024-03-19T00:00:00Z",
  "return_date": "2024-03-21T00:00:00Z",
  "status": "reserved",
  "times_delayed": 0,
  "equipment_type_id": 4,
  "equipment_size": "200 cm",
  "equipment_quantity": 1
}
```

PERM: Krever view-tilgang på **ReservationSelf** dersom man ser på en reserverasjon til egen bruker, krever view-tilgang på **Reservation** dersom man ser på en reserverasjon til en annen bruker.

/api/reservations/reservation/{reservation_id} [PATCH] Endre på en reserverasjon

PERM: Krever edit-tilgang på ressursen **Reservation**

Man kan kun endre på aktive reserverasjoner, dvs. status er **reserved**

Følgende verdier ved reservasjonen kan endres: `collection_date`, `return_date`, `loaner_id`

MERK: Siden man er ansatt/admin kan man fritt sette `return_date` og `collection_date` uavhengig av lengde på reservasjonen og om Utstyrsbanken er registrert som åpen, `times_delayed` vil heller ikke telles opp. Eneste krav for `return_date` er at den ikke er i fortida og etter `collection_date`.

Verdiene man ønsker å endre sendes som JSON, i eksemplet her redigerer vi reservasjonen med ID 23 og flytter den til 23/04-30/04:

```
{
  "collection_date": "2024-04-23T00:00:00Z",
  "return_date": "2024-04-30T00:00:00Z"
}
```

Svar (ved suksess):

```
{
  "text": "Successfully modified reservation with ID 23!",
  "loan": {
    "id": 23,
    "loaner_id": 1,
    "reserved_date": "2024-04-18T00:00:00Z",
    "collection_date": "2024-04-23T00:00:00Z",
    "return_date": "2024-04-30T00:00:00Z",
    "status": "reserved",
    "times_delayed": 0,
    "equipment_type_id": 6,
    "equipment_size": "200 cm",
    "equipment_quantity": 1
  }
}
```

`/api/reservations/reservation/{reservasjons-id}` [DELETE] Slett/kanseller en reservasjon

PERM: Krever delete-tilgang på `ReservationSelf` dersom man kansellerer en reservasjon til egen bruker, krever delete-tilgang på `Reservation` dersom man kansellerer en reservasjon til en annen bruker.

Setter status på reservasjonen til `not_picked_up` og øker antallet tilgjengelig av det reserverte utstyret slik at andre brukere kan låne og reservere utstyret igjen.

Svar (ved suksess):

```
{
  "text": "Reservation with ID 23 cancelled successfully!"
}
```

Liste over alle reservasjoner som skal plukkes opp idag `/api/reservations/today` [GET]

PERM: Krever view-tilgang på ressursen `Reservation`

Svar (ved suksess):

```
[
  {
    "id": 24,
    "loaner_id": 1,
    "reserved_date": "2024-03-15T00:00:00Z",
    "collection_date": "2024-03-15T00:00:00Z",
    "return_date": "2024-03-19T00:00:00Z",
    "status": "reserved",
    "times_delayed": 0,
    "equipment_type_id": 4,
    "equipment_size": "200 cm",
    "equipment_quantity": 1
  }
]
```

`/api/reservations/issues` [GET] - Hent alle reservasjoner med potensielle problemer

PERM: Krever view-tilgang på ressursen `Reservation`

Med problemer så menes følgende:

- `collection_date` er satt til en dato der Utstyrskassen holder stengt
- `return_date` er satt til en dato der Utstyrskassen holder stengt
- Lengden på reserveringen er lengre enn det som i utgangspunktet er tillatt

Denne infoen kan være nyttig etter man f.eks. har oppdatert åpningstidene.

Svar (ved suksess):

```
[
  {
    "id": 4,
    "loaner_id": 2,
    "reserved_date": "2024-04-15T00:00:00Z",
    "collection_date": "2024-04-16T00:00:00Z",
    "return_date": "2024-04-24T00:00:00Z",
    "status": "reserved",
    "times_delayed": 0,
    "equipment_type_id": 2,
    "equipment_size": "170 cm",
    "equipment_quantity": 1,
    "valid_collection_date": true,
    "valid_return_date": false,
    "loan_length": 8
  }
]
```

I eksempelet over har reserveringen en gyldig `collection_date`, men ikke en gyldig `return_date`. Reserveringen er dessuten for langt, maks tillatt er 7 dager som standard, reserveringen i eksempelet er 8 dager langt.

J.4 Brukerhåndtering

Brukerhåndtering

about:srcdoc

Brukerhåndtering

/api/users/register [POST] Registrering av nye brukere

PERM: Krever write tilgang på ressursen RegisterUser

For å registrere en ny bruker sendes detaljer om den nye brukeren slik:

```
{
  "phone_number": 12123434,
  "first_name": "Test",
  "last_name": "Bruker",
  "password": "V3rY$3curEpa$sw0rd"
}
```

Detaljene om den nye brukeren vil så bli lagret i databasen, avhengig av konfigurasjon vil det kreves verifisering av den nye brukeren ved å taste inn aktiveringskoden. Denne sendes på SMS til det registrerte telefonnummeret.

Respons (ved suksess) fra dette endepunktet ser ut som dette:

```
{
  "text": "Registered user 12123434 : Test Bruker with ID 10 successfully!",
  "user": {
    "id": 10,
    "first_name": "Test",
    "last_name": "Bruker",
    "phone_number": 12123434,
    "role_id": 0,
    "activation_status": "activated",
    "activation_expiration": "2024-02-26T16:30:51Z"
  }
}
```

For testing sin del er verifisering skrudd av, så brukeren er aktivert med en gang.

/api/users/resend?number={telefonnummer} [PATCH] Send ny aktiveringskode

Sender en ny aktiveringskode for uaktivererte brukere. Denne kan brukes f.eks. om den opprinnelige aktiveringskoden er utløpt.

Svar (ved suksess):

```
{
  "text": "Sent new activation code for user with number {phone_number}"
}
```

/api/users/verify [POST] Verifisering/aktivering av brukere

For å aktivere en bruker sender man telefonnummeret og aktiveringskoden:

```
{
  "phone_number": 12123435,
  "activation_code": 500889
}
```

Respons (ved suksess) fra dette endepunktet ser ut som dette:

```
{
  "text": "User with number 12123435 verified successfully!"
}
```

/api/users/guest/create [POST] Opprett en ny gjestebruker

PERM: Krever write-tilgang på User

Data om den nye gjestebrukeren sendes med som JSON:

```
{
  "first_name": "Roger",
  "last_name": "Halvorsen",
  "phone_number": 21219898
}
```

Svar (ved suksess):

```
{
  "text": "Created guest user 21219898 : Roger Halvorsen with ID 17!",
  "user": {
    "guests": [
      {
        "id": 17,
        "first_name": "Roger",
        "last_name": "Halvorsen",
        "role_id": 0,
        "phone_number": 21219898
      }
    ]
  }
}
```

/api/users/user?id={bruker-ider} [GET] Hent data om bruker(e)

PERM: Krever view-tilgang på UserSelf for egen bruker, view-tilgang på User for å se data om andre brukere

Henter all data (med unntak av passordhash) om brukerne spesifisert med IDer. Dersom man ønsker å hente data om flere brukere må IDene være på formen: 1,2,3

Eksempel: /api/users/user?id=1,2,4

```
{
  "registered": [
    {
      "id": 1,
      "first_name": "Åsulv",
      "last_name": "Vaa",
      "role_id": 3,
      "phone_number": 12345678,
      "activation_code": 0,
      "activation_status": "activated",
      "activation_expiration": "2024-03-25T13:36:02Z"
    },
    {
      "id": 2,
      "first_name": "Jonas",
      "last_name": "Smitt",
      "role_id": 1,
      "phone_number": 11223344,
      "activation_code": 0,
      "activation_status": "activated",
      "activation_expiration": "2024-03-25T13:36:30Z"
    }
  ],
  "guests": [
    {
      "id": 4,
      "first_name": "Kari",
      "last_name": "Smitt",
      "role_id": 0,
      "phone_number": 22334455
    }
  ]
}
```

Eksempel på respons på forespørselen `/api/users/user?id=2`:

```
{
  "registered": [
    {
      "id": 2,
      "first_name": "Jonas",
      "last_name": "Smitt",
      "role_id": 1,
      "phone_number": 11223344,
      "activation_code": 0,
      "activation_status": "activated",
      "activation_expiration": "2024-03-25T13:36:30Z"
    }
  ]
}
```

`/api/users/user?number={telefonnummer}` [GET] Hent data om bruker(e) som deler telefonnummeret

PERM: Krever view-tilgang på `UserSelf` for egen bruker, view-tilgang på `User` for å se data om andre brukere

Henter all data (med unntak av passordhash) om brukerne som deler spesifisert telefonnummer.

Eksempel på respons på forespørselen `/api/users/user?number=22334455`:

```
{
  "registered": [
    {
      "id": 3,
      "first_name": "Kari",
      "last_name": "Smitt",
      "role_id": 1,
      "phone_number": 22334455,
      "activation_code": 0,
      "activation_status": "activated",
      "activation_expiration": "2024-03-25T13:36:52Z"
    }
  ],
  "guests": [
    {
      "id": 4,
      "first_name": "Kari",
      "last_name": "Smitt",
      "role_id": 0,
      "phone_number": 22334455
    }
  ]
}
```

`/api/users/user?id={bruker-id}` [PATCH] Rediger en bruker

MERK: Endepunktet støtter kun 1 bruker av gangen, endepunktet støtter ikke bruk av telefonnummer da det kan være flere brukere knyttet til et og samme telefonnummer.

PERM: Krever edit-tilgang på `UserSelf` om man redigerer sin egen bruker, edit-tilgang på `User` dersom man redigerer noen andre sin bruker.

Følgende verdier kan endres: `first_name`, `last_name`, `phone_number`, `role_id` (kun admin/med edit-tilgang på `User`)

MERK: Dersom man endrer `phone_number` må brukeren aktiveres på nytt, brukeren vil få tilsendt SMS med aktiveringskode på sitt nye telefonnummer. Administrator kan sende med `no_activation` satt til `true` for å omgå dette.

Man sender enkelt med de verdiene som man ønsker å endre i JSON:

Eksempel, endre etternavn til bruker med ID 1: `/api/users/user?id=1`

```
{
  "last_name": "Vå"
}
```

Svar (ved suksess):

```
{
  "text": "Modified user with ID 1 successfully!",
  "user": {
    "registered": [
      {
        "id": 1,
        "first_name": "Åsulv",
        "last_name": "Vå",
        "role_id": 3,
        "phone_number": 95205721,
        "activation_code": 0,
        "activation_status": "activated",
        "activation_expiration": "2024-04-18T14:40:54Z"
      }
    ]
  }
}
```

/api/users/user?id={bruker-id} [DELETE] Slett en bruker

MERK: Endepunktet støtter kun 1 bruker av gangen, endepunktet støtter ikke bruk av telefonnummer da det kan være flere brukere knyttet til et og samme telefonnummer.

PERM: Krever delete-tilgang på User

MERK: Brukere med aktive lån kan ikke slettes, lånene må slettes/fullføres først før man sletter brukeren.

Ved sletting av brukeren skjer følgende:

- Alle brukerens reservasjoner kanselleres umiddelbart
- Brukerdata anonymiseres ved å sette telefonnummer til **NULL** samt navn til **Slettet bruker**
- **Registrert bruker:** Alle aktive sesjoner blir ugyldige, **activation_status** settes til **deleted**

Svar (ved suksess):

```
{
  "text": "User with ID 3 deleted successfully!"
}
```

/api/users/list [GET] List opp alle brukere

PERM: Krever "view"-tilgang på ressursen User

Henter opp en liste over alle registrerte brukere og alle gjestebukere.

Svar (ved suksess):

```
{
  "registered": [
    {
      "id": 1,
      "first_name": "Åsulv",
      "last_name": "Vaa",
      "role_id": 3,
      "phone_number": 12345678,
      "activation_code": 0,
      "activation_status": "activated",
      "activation_expiration": "2024-03-25T13:36:02Z"
    },
    {
      "id": 2,
```

```
    "first_name": "Jonas",
    "last_name": "Smitt",
    "role_id": 1,
    "phone_number": 11223344,
    "activation_code": 0,
    "activation_status": "activated",
    "activation_expiration": "2024-03-25T13:36:30Z"
  },
  {
    "id": 3,
    "first_name": "Kari",
    "last_name": "Smitt",
    "role_id": 1,
    "phone_number": 22334455,
    "activation_code": 0,
    "activation_status": "activated",
    "activation_expiration": "2024-03-25T13:36:52Z"
  }
],
"guests": [
  {
    "id": 4,
    "first_name": "Kari",
    "last_name": "Smitt",
    "role_id": 0,
    "phone_number": 22334455
  }
]
}
```

J.5 Diverse

Diverse

about:srcdoc

Diverse

/api/schedule [GET] Se informasjon om de vanlige åpningstidene samt unntaksdatoer

PERM: Krever view-tilgang på ressursen Schedule

Inkluderer også innstillingene for hvor langt i fremtiden en reservasjon kan være.

Svar (ved suksess):

```
{
  "weekly_schedule": [
    {
      "weekday": 2,
      "start_time": "2000-01-01T17:00:00Z",
      "closing_time": "2000-01-01T19:00:00Z",
      "is_open": true
    },
    {
      "weekday": 4,
      "start_time": "2000-01-01T17:00:00Z",
      "closing_time": "2000-01-01T19:00:00Z",
      "is_open": true
    }
  ],
  "exceptions": [
    {
      "date": "2024-03-18T00:00:00Z",
      "is_open": true,
      "opening_time": "2000-01-01T14:00:00Z",
      "closing_time": "2000-01-01T17:00:00Z"
    }
  ],
  "reservation_start_max_days": 14,
  "reservation_end_max_days": 7
}
```

Ukedagene følger denne konverteringen:

Verdi	Dag
0	Søndag
1	Mandag
2	Tirsdag
3	Onsdag
4	Torsdag
5	Fredag
6	Lørdag

MERK: Datoenes lagres uten tidspunkt i databasen, tidspunktene for åpning og stenging lagres uten dato. I backend blir dette gjort om til Go sin `time.Time` som alltid inkluderer både dato og tid. Derfor endre vi opp med at datoene får midnattstidspunkt og tidspunktene får datoen 01.01.2000. Tidssonen vil alltid bli UTC.

/api/schedule [PATCH] Endre åpningstidene til Utstyrsbanken

Via dette endepunktet kan man endre åpningstidene til Utstyrsbanken og dermed hvilke dager som er gyldige for reservasjoner og lån. Man kan endre både de faste ukentlige åpningstidene samt unntaksdager med andre åpningstider. Et eksempel på en forespørsel som setter åpningstidene til tirsdag 17-19, torsdag 17-19 samt at Utstyrsbanken er åpen førstkommande fredag (19.04) fra 16-18 og stengt torsdag (25.04) neste uke:

```
{
  "weekly_schedule": [
    {
```

```

        "weekday": 2,
        "start_time": "2000-01-01T17:00:00Z",
        "closing_time": "2000-01-01T19:00:00Z",
        "is_open": true
    },{
        "weekday": 4,
        "start_time": "2000-01-01T17:00:00Z",
        "closing_time": "2000-01-01T19:00:00Z",
        "is_open": true
    }
],
"exceptions": [
    {
        "date": "2024-04-19T00:00:00Z",
        "opening_time": "2000-01-01T16:00:00Z",
        "closing_time": "2000-01-01T18:00:00Z",
        "is_open": true
    },{
        "date": "2024-04-25T00:00:00Z",
        "opening_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z",
        "is_open": false
    }
]
}

```

- For tidspunktene `start_time`, `opening_time`, `closing_time` ignoreres dato
- For datoen `date` ignoreres tidspunkt
- HUSK: `weekday` er fra 0-6 og starter med søndag, ikke mandag
- Det forventes at ukedagene finnes i databasen allerede slik at det bare er å oppdatere de, se endringene i DB-fila.
- Man kan spesifisere kun `weekly_schedule` eller `exceptions` for å redigere kun de faste åpningstidene eller kun unntaksdager.
- Ukedager som er utelatt i forespørslen får beholde de verdiene de nå har, dersom man spesifiserer `exceptions` i forespørselen vil alle nåværende `exceptions` overskrives med de nye. Med andre ord dersom det finnes et unntak 18.04. og denne ikke er med når man vil legge inn et unntak 19.04. så blir unntaket den 18.04. slettet.

Svar (ved suksess):

```

{
  "text": "Successfully updated schedule! Note, this may cause certain loans and/or reservations to have invalid colle",
  "schedule": {
    "weekly_schedule": [
      {
        "weekday": 0,
        "start_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z",
        "is_open": false
      },
      {
        "weekday": 1,
        "start_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z",
        "is_open": false
      },
      {
        "weekday": 2,
        "start_time": "2000-01-01T17:00:00Z",
        "closing_time": "2000-01-01T19:00:00Z",
        "is_open": true
      },
      {
        "weekday": 3,
        "start_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z",
        "is_open": false
      },
      {

```

```

        "weekday": 4,
        "start_time": "2000-01-01T17:00:00Z",
        "closing_time": "2000-01-01T19:00:00Z",
        "is_open": true
    },
    {
        "weekday": 5,
        "start_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z",
        "is_open": false
    },
    {
        "weekday": 6,
        "start_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z",
        "is_open": false
    }
],
"exceptions": [
    {
        "date": "2024-04-19T00:00:00Z",
        "is_open": true,
        "opening_time": "2000-01-01T16:00:00Z",
        "closing_time": "2000-01-01T18:00:00Z"
    },
    {
        "date": "2024-04-25T00:00:00Z",
        "is_open": false,
        "opening_time": "2000-01-01T00:00:00Z",
        "closing_time": "2000-01-01T00:00:00Z"
    }
],
"reservation_start_max_days": 0,
"reservation_max_length_days": 0
}
}

```

- `reservation_start_max_days` og `reservation_max_length_days` kommer med uten sine reelle verdier siden en struct blir gjenbrukt.

`/api/stats?from={fra-dato}&to={til-dato}` [GET] Henter lånestatistikken for den gitte perioden.

PERM: Krever view-tilgang på Stats

Henter lånestatistikken for perioden basert på alle fullførte og aktive lån med `collection_date` i perioden.

Man vil få antall lån i perioden, antall lånt utstyr i perioden samt gjennomsnittlig antall lånt utstyr per lån. I tillegg vil man få samme statistikk for hver utstyrstype som ble lånt ut i perioden samt for hver kategori. Statistikk for de ulike størrelsene for hver utstyrstype er også inkludert.

Dato er på format: YYYY-MM-DD (dvs. uten tidspunkt)

Eksempel, periode 01/02/2024 - 29/02/2024, med testdata satt inn

GET på `/api/stats?from=2024-02-01&to=2024-02-29`

```

{
  "start": "2024-02-01T00:00:00Z",
  "end": "2024-02-29T00:00:00Z",
  "number_of_loans": 8,
  "quantity_loaned": 10,
  "avg_quantity_loaned": 1.25,
  "equipment": [
    {
      "id": 3,
      "name": "Skisko",
      "number_of_loans": 3,
      "quantity_loaned": 3,
      "avg_quantity_loaned": 1,
      "sizes": [

```



```
    {
      "size": "39",
      "number_of_loans": 2,
      "quantity_loaned": 2
    },
    {
      "size": "40",
      "number_of_loans": 1,
      "quantity_loaned": 1
    }
  ]
},
{
  "id": 6,
  "name": "Slalomski",
  "number_of_loans": 1,
  "quantity_loaned": 2,
  "avg_quantity_loaned": 2,
  "sizes": [
    {
      "size": "160 cm",
      "number_of_loans": 1,
      "quantity_loaned": 2
    }
  ]
},
{
  "id": 7,
  "name": "Slalomsko",
  "number_of_loans": 1,
  "quantity_loaned": 2,
  "avg_quantity_loaned": 2,
  "sizes": [
    {
      "size": "40",
      "number_of_loans": 1,
      "quantity_loaned": 2
    }
  ]
},
{
  "id": 2,
  "name": "Langrennski",
  "number_of_loans": 3,
  "quantity_loaned": 3,
  "avg_quantity_loaned": 1,
  "sizes": [
    {
      "size": "150 cm",
      "number_of_loans": 1,
      "quantity_loaned": 1
    },
    {
      "size": "170 cm",
      "number_of_loans": 2,
      "quantity_loaned": 2
    }
  ]
}
],
"categories": [
  {
    "id": 1,
    "name": "Vinter",
    "number_of_loans": 8,
    "quantity_loaned": 10,
```

```
    "avg_quantity_loaned": 1.25
  },
  {
    "id": 3,
    "name": "Langrenn",
    "number_of_loans": 6,
    "quantity_loaned": 6,
    "avg_quantity_loaned": 1
  },
  {
    "id": 4,
    "name": "Alpint",
    "number_of_loans": 2,
    "quantity_loaned": 4,
    "avg_quantity_loaned": 2
  },
  {
    "id": 5,
    "name": "Skitilbehør",
    "number_of_loans": 4,
    "quantity_loaned": 5,
    "avg_quantity_loaned": 1.25
  }
]
}
```

/api/config [GET] Hent nåværende verdi til de fire konfigurable verdiene

PERM: Krever view-tilgang på ressursen `Config`

Henter de nåværende verdiene til de konfigurable verdiene

Svar (ved suksess):

```
{
  "loan_max_times_delayed": 2,
  "reservation_start_max_days": 14,
  "reservation_max_length_days": 7,
  "loan_extension_max_days": 7
}
```

/api/config [PATCH] Endre en eller flere av de fire konfigurable verdiene

PERM: Krever edit-tilgang på ressursen `Config`

Send med en eller flere av de konfigurable verdiene som du ønsker å endre:

```
{
  "loan_max_times_delayed": 1
}
```

Svar (ved suksess):

```
{
  "text": "Successfully updated configurable values!",
  "config": {
    "loan_max_times_delayed": 1,
    "reservation_start_max_days": 14,
    "reservation_max_length_days": 7,
    "loan_extension_max_days": 7
  }
}
```

/api/roles [GET] Hent info om roller

PERM: Krever view-tilgang på Roles

Dette endepunktet er tiltenkt bruk dersom man ønsker å endre rollen til en bruker, da vil man trenge IDen til rollen.

Svar (ved suksess):

```
{
  "default_role_id": 1,
  "guest_role_id": 0,
  "employee_role_id": 2,
  "admin_role_id": 3,
  "Roles": [
    {
      "id": 3,
      "name": "Administrator"
    },
    {
      "id": 0,
      "name": "guest"
    },
    {
      "id": 1,
      "name": "User"
    },
    {
      "id": 2,
      "name": "Employee"
    }
  ]
}
```

J.6 Bildehåndtering

Bildehåndtering

about:srcdoc

Bildehåndtering

/api/images/upload?item={item-id} [POST] Last opp et nytt bilde

PERM: Krever write-tilgang på Inventory

Laster opp bildet til botta på Minio serveren og setter inn metadata som knytter bildet til utstyrstypen som er spesifisert (item-id).

Følgende krav/sjekker blir gjort på bildefila:

- Filtypen er en av følgende: jpeg, gif, png, bmp, avif, webp
- Filen er ikke større enn maks filstørrelse (konfigurerbar, default 10 MB)

MERK: Det er enkelt å omgå sjekken på filtype, nettleser vil da prøve og feile å vise det

Dette endepunktet bruker en multipart-form istedet for JSON i request body

Nøkkel	Verdtype	Påkrevd?	Standardverdi	Beskrivelse
image	Fil	Ja		Bildefilen som lastes opp
description	Tekst	Nei	""	Beskrivelse av bildet
order_position	Tekst (tall)	Nei	0	Sorteringsverdi når man viser flere bilder for en utstyrstype

Svar (ved suksess):

```
{
  "text": "Picture with ID 11 for equipment type with ID 5 uploaded successfully!"
}
```

/api/images/image/{picture-url} [GET] Hent et bilde

PERM: Krever view-tilgang på Inventory

Henter bildet fra Minio. Bildet sendes til klient/nettleser i original størrelse med korrekt metadata for Content-Type.

Svar (ved suksess): Kun bildet

/api/images/image/{picture-url} [DELETE] Slett et bilde samt tilhørende metadata

PERM: Krever delete-tilgang på Inventory

Sletter bildet fra Minio, tilhørende metadata slettes også fra databasen.

Svar (ved suksess):

```
{
  "text": "Deleted picture with 'URL' 331d54b1-e0cd-4825-a9f1-89a907cd3ad6! Associated metadata is also deleted"
}
```

/api/images/meta/{picture-id} [GET] Hent metadata om et bilde

PERM: Krever view-tilgang på Inventory

Henter metadata til bildet med ID picture-id dvs. picture_id i tabellen picture i databasen

Svar (ved suksess):

```
{
  "id": 10,
  "description": "j",
  "url": "10b9c0c7-b495-4eb0-a98c-2109d701d90d",
  "equipment_type_id": 5,
  "order_position": 1
}
```

/api/images/meta/{picture-id} [PATCH] Modifiser metadata om et bilde

PERM: Krever edit-tilgang på Inventory

Modifiserer metadata til bildet med ID `picture-id`

Følgende verdier kan endres: `description`, `order_position`

Verdiene man ønsker å endre, sendes med som JSON

```
{
  "description": "ny beskrivelse"
}
```

Svar (ved suksess):

```
{
  "text": "Modified picture metadata with ID 10 successfully!",
  "metadata": {
    "id": 10,
    "description": "ny beskrivelse",
    "url": "10b9c0c7-b495-4eb0-a98c-2109d701d90d",
    "equipment_type_id": 5,
    "order_position": 1
  }
}
```

`/api/images/meta/{picture-id}` [DELETE] Slett metadata samt tilhørende bilde

PERM: Krever delete-tilgang på Inventory

Sletter metadata om bildet samt bildet i Minio som er assosiert med metadataen.

Svar (ved suksess):

```
{
  "text": "Deleted picture metadata with ID 10 successfully! Associated picture with 'URL' 10b9c0c7-b495-4eb0-a98c-2109d701d90d"
}
```

Andre endringer i `images`-branchen

- For å laste opp og hente bilder må backend ha en forbindelse til en Minio-instans. Konfigurasjon av dette både med fil og med miljøvariabler er lagt til i README. Husk at man må lage en bøtte manuelt, samt få tak i en **Access Key** og tilhørende **Secret Key**, dette kan gjøres via WebUI til Minio.
- Bilder får en URL i form av en UUID som da kan legges til base URL: `/api/images/image/{uuid}` for å hente bildet, denne UUIDen kan også refereres `picture-url`.
- Metadata til bilder knyttet til utstyrstyper blir nå sendt som en array for hver utstyrstype i `/api/inventory/list` samt ved `/api/inventory/item/{item-id}`.

Eksempel:

```
{
  "id": 6,
  "name": "Slalomski",
  "description": "Slalomski til bruk i alpinbakken",
  "max_quantity": 3,
  "removed": false,
  "order_position": 4,
  "categories": [
    1,
    4
  ],
  "pictures": [
    {
      "id": 7,
      "description": "Slalomski/alpinski, faktisk ski du får låne kan avvike fra typen på bildet",
      "url": "c695691d-f8f8-4c6a-82a7-b1597cf599de",
      "equipment_type_id": 6,
      "order_position": 1
    }
  ]
}
```

```
    },  
    {  
      "id": 8,  
      "description": "Slalomski/alpinski, faktisk ski du får låne kan avvike fra typen på bildet",  
      "url": "8c3b9327-21e7-4631-a78a-4651f9e7f722",  
      "equipment_type_id": 6,  
      "order_position": 2  
    }  
  ]  
}
```

K Oppgavebeskrivelse

Oppgavebeskrivelsen som denne bacheloroppgaven er basert på. Teksten er skrevet av oppdragsgiver, altså Utstyrskontrollen.

Begynner på neste side

Oppgave 33

BPROG

2-4 stk

Oppgavetittel: Tidsriktig utlånsordning av aktivitetsutstyr: Digital løsning for Utstyrskassen i Vestre Toten

Bedrift: Venstre Toten Frivilligsentral
Kontaktperson: Gunnhild Andersen
E-post: post@vestretoten.frivilligsentral.no
Telefon: 41646475
Lokasjon: Raufoss

Beskrivelse av oppgaven

Digital løsning for booking og oversikt over utstyrskassen til Utstyrskassen i Vestre Toten. Per tid føres det manuelle lister over utlån, som i realiteten gir lite eller ingen oversikt over hva som er tilgjengelig, uten at man må være fysisk i lokalet. Man har heller ikke enkel oversikt over hva som er utlånt utover den avtalte fristen, uten å bla seg gjennom permer med utlånskjemaer.

Løsningen skal bidra til å gjøre det enkelt for låner å reservere utstyr, få visuell oversikt over hva som finnes av utstyr og få påminnelser om å levere utstyr.

For utlåner skal den skape oversikt over tilgjengelig og utlånt beholdning av utstyr, det skal være enkelt å registrere utlån inn og ut og løsningen skal kunne lage statistikk om bruk.

Se vedlegg for en detaljert oppgavebeskrivelse.

Oppgave 33

BPROG

2-4 stk

Digital løsning for Utstyrsbanken i Vestre Toten kommune.**Bestiller:**

Utstyrsbanken i Vestre Toten kommune er en sentral for utlån av aktivitetsutstyr til alle årstider. Utlån av utstyr er gratis og hvem som helst kan låne aktivitetsutstyr der. Utstyrsbanken er organisert under Vestre Toten Frivilligsentral.

Bestilling:

Digital løsning for booking og oversikt over utstyrsbeholdning. (BUA sin løsning kan være til inspirasjon.) Per tid føres det manuelle lister over utlån, som i realiteten gir lite eller ingen oversikt over hva som er tilgjengelig, uten at man må være fysisk i lokalet. Man har heller ikke enkel oversikt over hva som er utlånt utover den avtalte fristen, uten å bla seg gjennom permer med utlånsskjemaer.

Løsningen skal bidra til å gjøre det enkelt for låner å reservere utstyr, få visuell oversikt over hva som finnes av utstyr og få påminnelser om å levere utstyr.

For utlåner skal den skape oversikt over tilgjengelig og utlånt beholdning av utstyr, det skal være enkelt å registrere utlån inn og ut og løsningen skal kunne lage statistikk om bruk.

Innhold:**For låner:**

Fra startside: Jeg vil låne utstyr Logg inn – opprett bruker. Innlogging med epost og passord.

Brukervennlig fra alle skjermer (mobil, PC, tablet).

Mulighet for sortering alfabetisk, i årstider- søkefelt

Bra størrelse på bilder med beskrivelse – trykke på bilde for å komme inn til valg på størrelser – trykke rett på størrelser – ulik farge på tilgjengelig/ ikke tilgjengelig størrelse. Utstyret må være inne, for å kunne reserveres.

Mulighet for reservasjon til ønsket dato (bare mulig å velge åpningsdatoer)

Mulighet for å søke om å forlenge utlånet

Må kunne melde inn feil og skader på utstyret.

For ansatt:

Må være enkelt å få oversikt over:

- Dette er reserverasjoner til dagens dato
- Dette er utstyr som ikke er levert til avtalt dato: Knapp for å sende utfylt purre sms

Oppgave 33

BPROG

2-4 stk

- Dette er utstyr som forventes levert i dag:

Løsningen sender automatisk purre-sms til låner dagen før avtalt returdag.

Det må være enkelt å sjekke beholdning – hvor mange par ski har vi inne? Fx sitte på kontor annet sted og ta imot ønske om å reservere for en stor gruppe.

Enkel registrering av utlån og retur. Kvittering på levert utstyr sendes automatisk på epost.

Registrering av utlån må ha støtte for QR scanning, da utstyr skal QR merkes.

Må kunne lage opprette bruker manuelt.

Annet:

Menyvalg for å legge inn diverse informasjon om åpningstider + kom i kontakt utenom åpningstid – andre avtaler ang utstyr – informasjon om henger mm

Opplæring til ansatt

Enkelt å ta ut statistikk på antall totale lån, utlån per årstidskategori, per utstyr og per størrelse.

Teknisk:

- Bør bygges som en progressiv web app (PWA). Ønsker ikke egne apper på AppStore/GooglePlay.
- Fritt valg av frontend og backend rammeverk. Benytt gjerne så mye hyllevare og godt kjente rammeverk; Svelte, Vue, React e.l. for frontend. Og PHP, .net C# e.l. for backend. Samme også for stil/layout; Bootstrap, Tailwind, Bulma e.l.
- Drift av løsningen kan gruppen selv velge. Dette løses av oppdragsgiver selv på et senere tidspunkt.
- Innlogging bør gjøres med Open ID Connect (OIDC) slik at BankID identifisering enkelt kan legges til senere. Eksempel på ferdig løsning som kan benyttes er Auth0 (www.auth0.com).

