Bacheloroppgave

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

Hansen, June Veronica Einarsen
Hjelle, Sindre Opskar
Næsvold, Leif Torbjørn

Internutleie - Communicate

Bacheloroppgave i ingeniørfag, data
Veileder: Rune Hjelsvold

Mai 2021

NTNU
Norwegian University of
Science and Technology

Hansen, June Veronica Einarsen
Hjelle, Sindre Opskar
Næsvold, Leif Torbjørn

Internutleie - Communicate

Bacheloroppgave i ingeniørfag, data
Veileder: Rune Hjelsvold
Mai 2021

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

NTNU
Kunnskap for en bedre verden

Abstract

Technology opens the possibility of automating workflows that have previously been manual, which can help streamline companies. In this project we have developed a full-fledged web application for internal rental of Communicate's cabins and apartments. The application automates tasks related to administrating the rental. As an example, allocations of bookings is done automatically based on the client's internal regulations, and it sends various automatic email alerts to both users and system administrators.

The application gives a good user experience, quality assured through user testing. We have mainly used Kanban as our development methodology, with elements from other agile methodologies. Angular is used for the frontend, .NET for the backend, and a number of relevant Azure services as part of deployment. Throughout the project the group has focused on delivering a high quality product that fulfills the client's requirements and wishes.

Sammendrag

Teknologi åpner muligheter for automatisering av arbeidsprosesser som tidligere har vært manuelle, noe som kan være med på å effektivisere bedrifter. I denne oppgaven har vi utviklet en fullverdig web-applikasjon for internutleie av Communicates feriesteder. Applikasjonen automatiserer en rekke arbeidsoppgaver knyttet til administrasjon av utleien. For eksempel gjøres tildeling av bookinger automatisk basert på oppdragsgivers interne regelverk, og det er satt opp diverse automatiske varslinger til både brukere og administratorer via epost.

Applikasjonen gir en god brukeropplevelse, som er kvalitetssikret gjennom brukertesting. Vi har hovedsaklig benyttet Kanban som utviklingsmetodikk, med elementer fra andre smidige metoder. Angular benyttes i frontend, .NET i backend, og en rekke relevante Azure-tjenester som en del av produksjonssettingen. Gjennom prosjektet har gruppen hatt fokus på å levere et produkt av høy kvalitet som oppfyller oppdragsgivers krav og ønsker.

Forord

Vi ønsker å rette en stor takk til vår veileder Rune Hjelsvold som har gitt oss god veiledning gjennom gode diskusjoner og svært nyttige tilbakemeldinger.

Vi ønsker også å takke Communicate for en interessant oppgave som har bydd på enorme mengder læring, og særlig kontaktpersonen vår Kaya Masterød Karlsen for for god oppfølging, tilgjengelighet og veiledning underveis.

Til slutt ønsker vi å takke testbrukerne for både gode og nyttige tilbakemeldinger, samt familie og venner som har stilt opp for å rettlese rapporten.

Innhold

Figurer

Tabeller

Kodelister

Akronymer

CI/CD Continuous Integration og Continuous Deployment. 18, 20, 75, 80

HTML HyperText Markup Language. 22

IDE Integrated Development Environment. 66

LTS Long-term support. 21, 81

OS Operativsystem. 21

UI User Interface. 21, 28, 70, 76, 77

WCAG Web Content Accessibility Guidelines. 16, 41, 42

Ordliste

Communicate Se Communicate Norge. iii, iv, 1, 2, 4, 5, 11, 15, 17, 18, 22, 34, 47, 65, 81, 86, 100–105, 108, 111, 113, 118

Communicate Norge Oppdragsgiver for oppgaven. Konsulentselskap som spesialiserer seg i integrasjonsløsninger og tjenesteplattformer utviklet i .NET. xv, 1, 100, 102

CORS policy Cross Origin Requests Policy, brukt for å sikre hvilke/hvor HTTP-kall kommer fra. 50, 81

Data Recieving Object Objekttypen APIet forventer å motta (forkortet DRO). 52

Data Transfer Object Objekttypen som sendes som retursvar fra APIet (forkortet DTO). xv, 52

GDPR Personvernforordningen er en forordning for å styrke personvernet ved behandling av personopplysninger i Den europeiske union.. 15, 23

mappere Utilityklasser som konverterer modelltyper, for eksempel en databasemodell til en Data Transfer Object. 29, 52

Material Design Et designsystem laget av Google som kan brukes til å forbedre designet til nettsider og apper, består av både komponenter og prinsipper. 16, 43

Node Package Manager Pakkebehandleren (package manager) benyttet i JavaScript. 44, 66

NuGet Pakkebehandleren (package manager) benyttet i .NET. 21

REST REST er en software arkitektur stil, som bruker et subset av HTTP. 15, 25

SQL Språk for å skrive til/lese fra database. 19, 44, 82

Swagger Verktøy for blant annet å teste API-endepunkter. 68, 70, 84

Table Storage NoSQL database i Azure. 19, 29, 32, 44, 52, 82

utleieobjekt En hytte eller leilighet som disponeres av selskapet til utleie for de ansatte. 2, 3, 12–14, 23, 29, 32, 34, 35, 37, 40, 41, 43, 47–49, 52, 53, 55, 56, 70, 110

W3C er en internasjonal standardiseringsorganisasjon som utvikler protokoller og standarder for World Wide Web. xvi, 50

YAML Rekursivt akronym for "YAML Ain't Markup Language". En standard for dataserialisering laget for å være vennlig for lesing av menneksker. Ofte brukt til konfigurasjonsfiler, ofte til samme type ting XML kan benyttes til. 76–78

Kapittel 1

Introduksjon

1.1 Bakgrunn

Communicate Norge er et utviklingsselskap som hovedsaklig jobber med integrasjon mellom ulike plattformer. Selskapet ble startet opp i 1996, og har kontorer i Oslo, Halden og Stavanger. Selskapet har i dag 95 ansatte, som har mulighet til å disponere tre ulike feriesteder gjennom selskapet. Alle feriestedene er tilgjengelige hele året.

1.1.1 Prosjektmål

Målet med prosjektet kan i hovedsak deles inn i tre ulike kategorier; *resultatmål*, *effektmål* og *læringsmål*. Resultatmål og effektmål er selvsagte måltyper for et utviklingsprosjekt, mens læringsmål i tillegg er med både fordi det er et hovedmål med en bacheloroppgave, og fordi Communicate har uttrykt et ønske om at oppgaven skal benyttes til læring for prosjektgruppen.

Resultatmål

Resultatmålet med prosjektet er å lage en nettside for internutleie av selskapets feriesteder. Løsningen skal som et minimum fungere på PC/tablet, og helst utvides slik at den også fungerer fra mobil-enheter. Videre funksjonalitet implementeres så langt det lar seg gjøre etter en liste prioritert etter funksjonalitet.

- Å fremstille informasjon på en ryddig og oversiktlig måte.
- Å automatisere prosessen med utleie slik at administrative kostnader knyttet til tilbudet minimeres.
- At løsningen skal utvikles og dokumenteres på en slik måte at andre utviklere senere kan gå inn å vedlikeholde eller videreutvikle den.

Effektmål

Oppgaven er gitt på bakgrunn av at et av gruppemedlemmene har fått jobb hos oppdragsgiver, hovedformålet med oppgaven er derfor at vedkommende skal bli mest mulig kjent med verktøyene som oppdragsgiver normalt bruker. Selve prosjektet skal erstatte en eksisterende løsning som er lite oversiktlig, svært utdatert, og som krever uforholdsmessig store ressurser administrativt. Den nye løsningen har derfor som mål:

- Å gi fremtidig ansatt erfaring fra verktøyene som oppdragsgiver benytter.
- Å redusere driftskostnader knyttet til dagens løsning.
- Å redusere administrative kostnader i form av at ansatte med ansvar for løsningen kan bruke ressursene sine på verdiskapning for bedriften, fremfor manuell håndtering av internutleie.

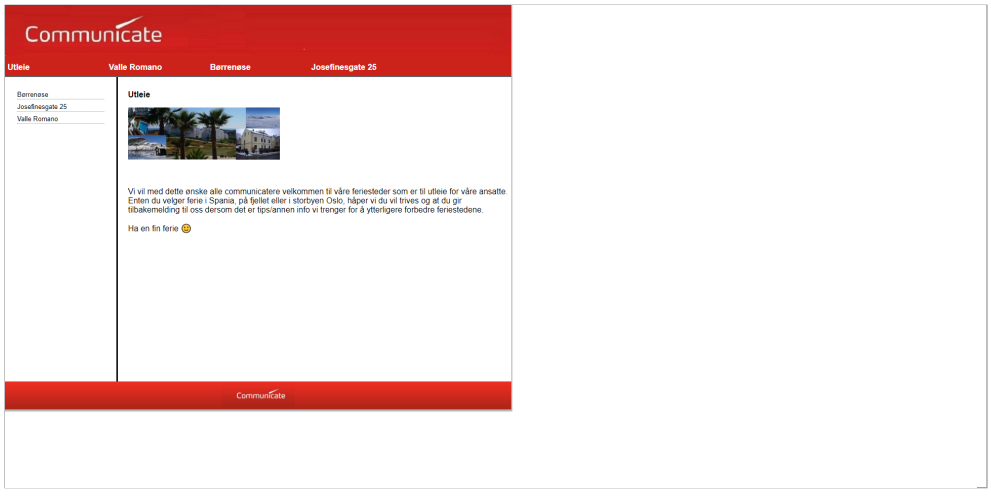
Læringsmål

Bacheloroppgaven har overordnede læringsmål som følge av emnebeskrivelsen til bacheloroppgavefaget (NTNU 2021). Samtidig som vi har fått spesifisert disse målene noe gjennom ønsker om hva vi skal lære fra Communicate - blant annet å bli kjent med .NET-rammeverket og Azure. I tillegg har gruppen selv noen mål om ønsket læring, blant annet å tilegne seg erfaring ved bruk av vår valgte utviklingsmodell, og å få bedre kjennskap til kvalitetssikring og testing. En oversikt over alle læringsmålene våre kan sees i vedlegg C.1.2.

1.2 Prosjektbeskrivelse

1.2.1 Tidligere løsning

Administrasjonen av denne internutleien gjøres per i dag også gjennom en nettside, men den er utdatert både administrativt og visuelt (se figur 1.1). Den har i praksis kun fungert som en informasjonsplattform om hvilke utleieobjekter som eksisterer og reglene for disse, og som et epostskjema.



Figur 1.1: Eksisterende løsning vist på en standard 1080p skjerm med 16:9 sideforhold (inkludert tomt areal).

Administrativt skjer bookinger gjennom at skjemaet for leie på nettsiden sender en epost til ansvarlig for administrasjonen av tjenesten. Den ansvarlige

gjør tildeling av disse bookingene manuelt, basert på spesifiserte utleieperioder for de enkelte utleieobjektene og et poengsystem som er satt opp for å forsøke å rettferdiggjøre utleien. Poengsystemet som benyttes er forklart i vedlegg F.

Den eksisterende løsningen har heller ingen antydninger til responsivt design, og ved bruk av mobil blir nettsiden lineært nedskalert til bredden av mobilskjermen, slik at brukeren må zoome inn på sidens elementer for å kunne se hva som står.

1.2.2 Ønsket løsning

Oppdragsgiver ønsker en ny nettside for drifting av internutleien. Det er ønskelig at den har en god brukeropplevelse, og at den automatiserer mye av arbeidet som mulig, slik at arbeidstimer frigjøres for dagens administrator (heretter kalt admin). Det betyr innebærer blant annet at tildeling av feriesteder (referert til videre i oppgaven som utleieobjekter) skjer automatisk basert på poengsystemet og tildelingsperioder. I tillegg til at alle brukernes poeng oppdateres på årsdag for ansettelse. Det er også ønske om at systemet automatisk lager en rapport over hvem som har booket et utleieobjekt de ulike månedene, slik at den kan sendes til lønnsavdelingen og betaling for leie trekkes fra lønningen til leietakeren. En full oversikt over krav til systemet kan sees i kapittel 3.

1.2.3 Avgrensning oppgave

Oppdragsgiver gjorde det klart at størrelsen på systemet, og kompleksiteten, gjorde at de så det som usannsynlig at vi kom til å bli ferdig med absolutt alle ønsker før utgangen av bachelorprosjektet. Derfor har vi en avtale med oppdragsgiver om at prosjektet skal ha den mest sentrale funksjonaliteten på plass, og at det skal være godt dokumentert. All funksjonalitet og annet som ikke er på plass ved slutten av bachelorperioden skal legges i backloggen som har vært brukt gjennom prosjektet slik at det er enkelt for andre å jobbe videre på backloggen og ferdigstille prosjektet.

Avgrensning rapport

Rapporten I rapporten ønsker vi å beskrive valgene vi har tatt underveis i prosjektet. Diskutere fordeler og ulemper rundt valgene vi har gjort. Gjennom denne tilnærmingen ønsker vi å skape et helhetlig bilde av applikasjonen vi har laget, hvilken funksjonalitet som foreligger og uforutsette konsekvenser av tekniske- og designmessige valg som har kommet opp i ettertid. I tillegg ønsker vi å se prosjektet i et læringsperspektiv gjennom å trekke paralleller mot læringsprosessen i argumentasjonen.

1.3 Gruppens bakgrunn og kompetanse

Prosjektgruppen er sammensatt av studenter fra studieretningene bachelor i programmering og bachelor i ingeniørfag data. Hele gruppen hadde ved oppstart av prosjektet generell kunnskap om systemutviklingsprosssessen gjennom blant annet prosjektering, kravspesifisering og databasemodellering fra tidligere fag. I tillegg hadde programmeingsstudentene blant annet hatt kjennskap til applikasjonsutvikling, og hatt innføring i web- og cloud-teknologier. Rammeverkene .Net og Angular er derimot ukjent for hele gruppen, og ingen hadde skrevet i hverken C# eller TypeScript tidligere. Det var heller ingen av gruppemedlemmene som hadde benyttet noen av Azures tjenester, ei heller utviklet et større system eller gjennomført en produksjonssetting.

1.4 Organisering

1.4.1 Ansvarsfordeling

Ansvarsfordelingen i gruppen var som følger:

- June Hansen
  - Prosjektleder - valgt av gruppen, og var et ganske naturlig valg med tanke på tilknytning til oppgaven og Communicates kontaktperson, samt utdannelse innen administrasjon og ledelse.
  - Har arbeidet fleksibelt mellom frontend og backend, pga. ønske om å lære mest mulig. Dette ga også ekstra sikkerhet dersom en av de to andre hadde falt ut av prosjektet underveis.
- Sindre Opskar Hjelle
  - Utvikler frontend - hovedansvar for frontend med bakgrunn i tidligere erfaring med webutvikling og generell interesse for å skape gode brukeropplevelser.
  - Dokumentansvarlig - Selvpåtatt ansvar for at dokumenter leveres når de skal, inneholder det de skal osv.
- Leif Torbjørn Næsvold
  - Utvikler backend - hovedansvar for backend med bakgrunn i tidligere utviklingserfaring og interesse for logiske operasjoner.
  - Referent - Tok naturlig rollen fra første møte, og ble derfor senere valgt til referent.

1.4.2 Øvrige roller

- Rune Hjelsvold har vært gruppens internveileder fra NTNU. Gruppen har hatt faste møter med Rune hver torsdag klokka 14:00, som kun unntaksvis har vært droppet. Han har veiledet oss gjennom prosessen fra planlegging, prosjektgjennomføring og rapportskriving, og har vært en svært

nyttig sparringspartner innenfor overordnede problemstillinger gruppen har møtt på.

- Kaya Masterød Karlsen har vært gruppens kontaktperson hos oppdragsgiver. Gruppen har hatt faste møter med Kaya hver tirsdag klokka 13:00. I disse møtene har prioriteringer av oppgaver og faglige spørsmål blitt tatt opp. I tillegg har Kaya fungert som kontaktperson videre inn mot Communicate, og videreført spørsmål vi har hatt til admin, funnet testpersoner osv.

1.5 Om rapporten

Målgruppen for rapporten er først og fremst sensor og veileder for oppgaven, men kan også være nyttig for eventuelle utviklere som tar på seg videreutvikling av prosjektet i fremtiden. I tillegg kan den være nyttig for for eksempel andre studenter som som ønsker innsikt i hvordan prosjektet er utført, hvilke verktøy som er benyttet og hvorfor.

Vi har gjennomgående forsøkt å oversette engelske uttrykk for bedre lesbarhet på norsk, men for å ikke skape uklarheter er det engelske uttrykket ofte lagt i parentes første gang det nevnes i en gitt sammenheng.

I PDF-versjonen av rapporten er kapittel- og vedleggshenvisninger klikkbar, det samme er kildehenvisninger, samt utvalgte forkortelser og begreper. Man kan også navigere via innholdsfortegnelsen, og via listene over figurer, tabeller og kodelister.

Kapittel 2

Utviklingsprosess

2.1 Valg av systemutviklingsmetode

I forprosjektfasen ble det lagt stor vekt på kartlegging av behov for hele prosjektet. Hvilken utviklingsmodell som egnet seg best i vårt prosjekt ble diskutert etter at følgende elementer ble trukket frem:

- Kort utviklingsperiode på bakgrunn av prosjektperioden til bacheloroppgaven.
- Brukervennlighet og automatisering er de viktigste egenskapene ved sluttproduktet.
- Situasjonen med Koronaviruset (Covid-19) gjør at utviklingsteamet er lokalisert på ulike steder, og har begrensede og usikre muligheter for å møtes.
- Produkteier anbefalte på det sterkeste at vi samlet så mye som mulig i prosjektportalen i Azure - både repositories, bruk av Boards, Pipelines ol.
- Ukentlige statusmøter med produkteier.
- Ukentlige diskusjonsmøter med intern veileder.
- Daglige lunsjmøter innad i utviklingsteamet.

Det ble bestemt at vi som en del av kravspesifikasjons-prosessen skulle sette opp en prioritert liste over funksjonalitet. Etter ønske fra produkteier vil dette bli plassert i en Kanban-tavle slik at produkteier også skulle ha tilgang til hvordan gruppen lå an i arbeidet. Ettersom vi la opp til brukertesting, var det også forventet at det kunne komme innspill til endringer eller ny funksjonalitet underveis i prosjektet. Med bakgrunn i dette ble smidig utvikling sett på som det mest naturlige valget.

Gruppen vurderte de største smidige utviklingsmodellene. Extreme Programming virket lite aktuelt da det er mye fokus på parprogrammering, og samlokalisering som vanskeliggjøres pga. Covid-19. XP har også hovedfokus på utvikling hvor spesifikasjonene endrer seg hyppig - noe som ikke er tilfellet, selv om vi forventer noen endringer underveis (Wells 1999).

Både Scrum, iterativ utvikling og Kanban ble alle sett på som svært gode alternativer for prosjektet. Fordi oppdragsgiver ønsket at vi uansett skulle holde oversikt over oppgavene våre en Kanbantavle på bakgrunn av ønske om etterprøvbarhet og videreutvikling fra deres side, ble Kanban ble derfor en naturlig del av utviklingsprosessen. Kanban gir også et fokus på å hele tiden gjennom-

føre påbegynte oppgaver, samt fokus på å alltid ta øverst prioritert oppgave når man starter på noe nytt. På denne måten sikret vi at den viktigste oppgaven alltid skulle bli tatt hånd om og gjennomført først (Radigan 2014).

Grunnelementene ved Kanban legger ikke opp til korte daglige møter. Dette var noe gruppen hadde som ønske for å holde hverandre oppdatert både på arbeid som er ferdig, arbeid som er underveis, og arbeid som er planlagt - lignende de daglige møtene fra Scrum (Sommerville 2004). I tillegg så vi møtene som en god anledning til å gjøre parprogrammering ved behov. Parprogrammering er et verktøy som brukes aktivt i blant annet Extreme Programming. Fordi vi var tre utviklere var parprogrammering ikke mulig å gjennomføre konsekvent, men vi anså det likevel som et nyttig vertkøy for læring og debugging.

Det ble avtalt et mer omfattende møte mellom hver utviklingsperiode, hvor vi både evaluerte arbeidet som var gjort i forrige periode, og planla neste periode. Dette kan sees på som en kombinasjon av de retrospektive- og planleggingsmøtene vanlige i Scrum (Schwaber & Sutherland 2020). Dette for å sikre at gruppen er samstemt gjennom hele prosessen, samt holde kontroll på hvordan gruppen ligger an underveis i både utviklingsperiodene og prosjektet.

2.2 Gjennomføring

2.2.1 Utviklingsperioder

I løpet av forprosjektet delte vi opp prosjektet i utviklingsperioder med tilhørende milepæler. Hver utviklingsperiode har blitt innledet med en kort arbeidsfordeling, i tillegg til at gruppen har gjennomført evalueringer av arbeidet som er gjort - både utviklingsmessig og administrativt. Formålet med dette har vært at gruppen skal kunne jobbe så effektivt som mulig, og dra nytte av hverandres kunnskap. Etter kravspesifiseringen så vi for oss at prosjektet kom til å bestå av seks ulike utviklingsperioder slik figur 2.1 viser.

Figur 2.1: Gantt-skjema som viser oversikt over ulike utviklingsperioder

Som figuren viser la vi opp til seks prosjektperioder, hvorav en innledende planleggingsperiode, en avsluttende rapportskrivingsperiode og fire utviklingsperioder. Fokuset for de fire utviklingsperiodene var som følger:

1. Minimumsløsning og produksjonssetting
2. Automatisering
3. Administrasjonsside og klargjøre for brukertesting

4. Forbedring

Etter problemer særlig knyttet til produksjonssetting og oppstartsproblemer med rammeverkene vi hadde valgt (mer dyptgående diskutert i kapittel 9.2.1), havnet vi etter originalt tidsskjema i den første utviklingsperioden. Vi valgte her at to av gruppas medlemmer fortsatte utvikling på neste periode, mens sistemann fortsatte på utfordringene som hang igjen fra første periode. På denne måten var vi ajour til utgangen av andre utviklingsperiode, og fortsatte i planlagt tempo etter dette.

2.2.2 Møter

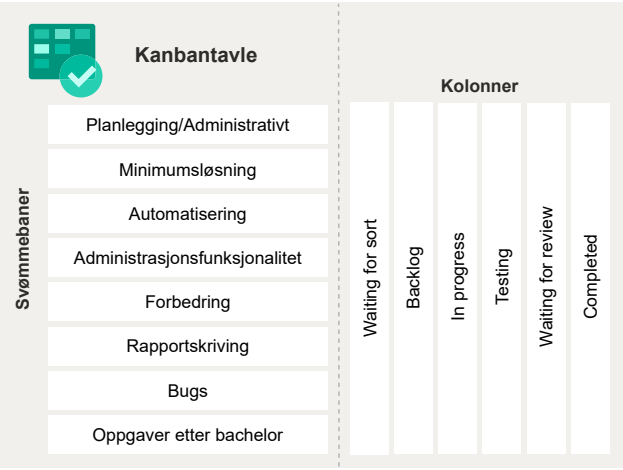
I begynnelsen av prosjektet avtalte vi ukentlige møter med både veileder og kontaktperson hos oppdragsgiver. Disse møtene fortsatte i hovedsak å foregå ukentlig gjennom prosjektet, men med noen færre møter med veileder i perioder vi utelukkende hadde fokus på utvikling. Disse møtene fungerte godt, og har vært preget av diskusjoner rundt ulike valg vi har tatt underveis i prosessen. I tillegg har vi etter hver utviklingsperiode presentert produktet til oppdragsgiver. Dette har sikret en kontinuerlig tilbakemelding, og vært nyttig i form av innspill til prioriteringer.

Vi avtalte også daglige møter internt i gruppen. Disse ble benyttet til planlegging og diskusjon, samt å hjelpe hverandre med ulike problemstillinger. Møtene ble også brukt til parprogrammering, særlig i forbindelse med debugging når et gruppemedlem hadde hatt konkrete vedvarende problemer som ikke lot seg løse på egenhånd. Dette har fungert godt, og vi brukte i gjennomsnitt ca. 3 timer hver uke til dette - altså 9 arbeidstimer totalt. Gruppen har gjennom hele prosjektet hatt god arbeidsfordeling og godt samarbeid.

2.2.3 Kanbantavlen

Oppsettet vårt av kanbantavlen kan sees i figur 2.2. Tavlen var satt opp med svømmebaner som hovedsaklig representerer de ulike utviklingsperiodene, samt en svømmebane for bugs og en for oppgaver som burde gjøres under videreutvikling av applikasjonen etter at bachelorperioden er over.

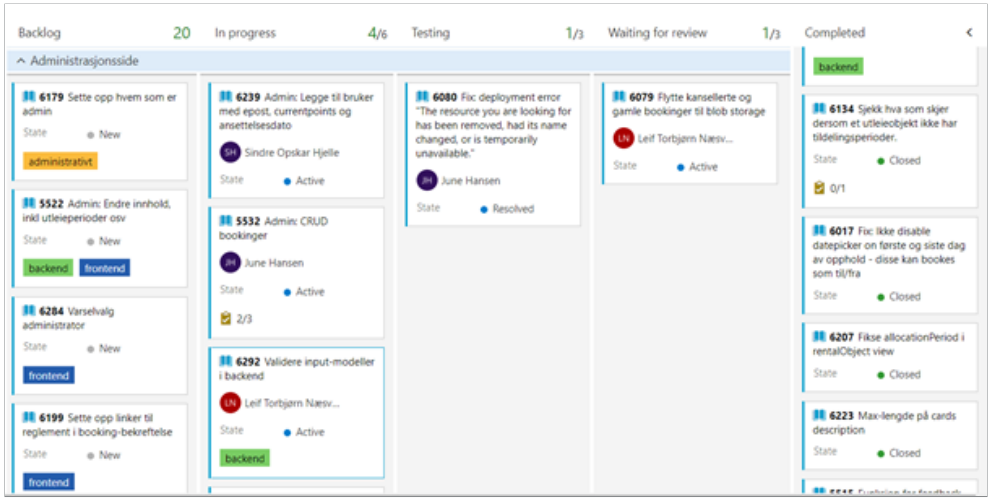
Kolonnene er satt opp slik at oppgaver som venter på å sorteres ligger i



Figur 2.2: Figur som viser oppsettet av kanbantavlen

første kolonne, oppgaver som er sortert til en periode ligger i kolonne to, oppgaver som jobbes på ligger i kolonne 3, oppgaver som testes ligger i kolonne 4, oppgaver som venter på at et annet gruppemedlem skal sjekke koden ligger i kolonne 5, og ferdige oppgaver ligger i kolonne 6. Det var også satt opp en begrensning for hvor mange oppgaver som kan ligge i hver kolonne for å oppfordre gruppemedlemmene til å gjøre ferdig hver oppgave før neste påstartes.

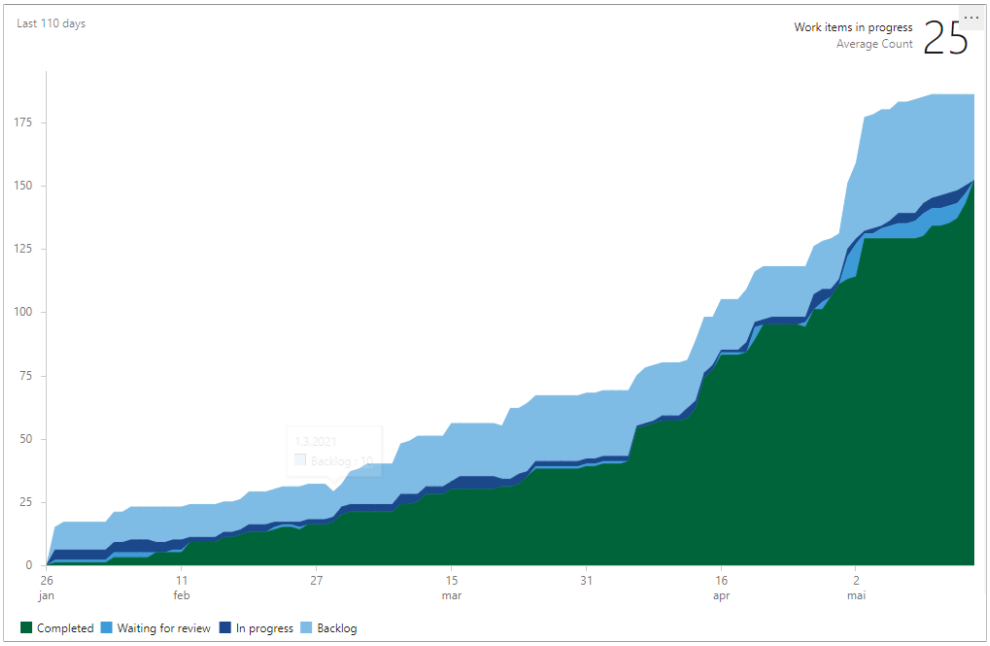
Kanbantavlen har vært kontinuerlig oppdatert etterhvert som oppgaver har blitt omprioritert, gjort eller lagt til - uavhengig om dette var etter gruppemedlemmers initiativ eller fra oppdragsgiver. Et eksempel på hvordan kanbantavlen så ut under utvikling kan sees i figur 2.3.



Figur 2.3: Skjermdump av kanbantavlen underveis i utviklingsperioden med fokus på administrasjonssiden

For å gi et innblikk i gruppens progresjon innen bruk av kanbantavlen som verktøy, samt hvordan gruppen etterhvert klarte å effektivisere progresjonen, har vi valgt å ta med en kumulativ figur (figur 2.4) som viser oversikten over antall oppgaver i de ulike kolonnene i tavlen.

Progresjonen for fullførte oppgaver (mørk grønn) økte betydelig i sluttfasen av prosjektet, og det samme gjorde antall oppgaver i backloggen (lys blå). Dette er stort sett grunnet at oppgavene stadig ble mindre etterhvert som vi jobbet, noe som naturlig gjorde at vi ble ferdig med stadig fler. Den siste store økningen i backlog og ferdige oppgaver er en kombinasjon av oppgaver som ble lagt inn etter brukertesting og raskt gjort ferdig, og rapportoppgaver som ble lagt inn i starten av rapportperioden. Økningen i backloggen, utover oppgavene som ble ferdig, var en gjennomgang av prosjektet for å legge inn oppgaver i backloggen for videreutvikling etter endt bachelorperoide.



Figur 2.4: Kumulativt diagram som viser utviklingen av antall oppgaver i de ulike kolonnene per tid i prosjektarbeidet (generert i DevOps)

Kapittel 3

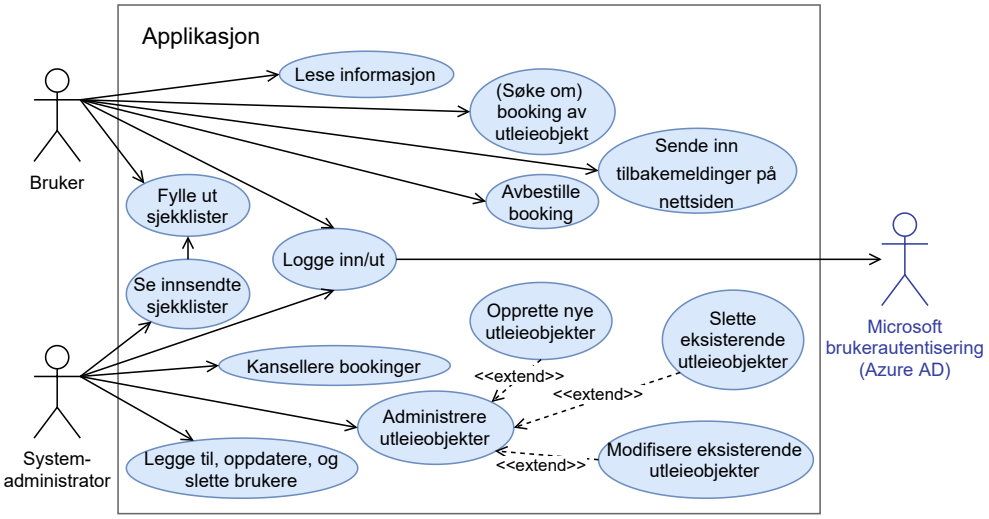
Kravspesifisering

Communicate hovedmål med oppgaven var å få en applikasjonen som utvikles kan benyttes internt, og som frigjør arbeidstid for admin ved automatisering av ting som tildeling og lignende. Likevel var det fra alle parter en forståelse for, og i stor grad forventning om, at bachelorgruppen kunne komme i en situasjon hvor det ikke ble tid til å implementere all ønsket funksjonalitet. Den opprinnelige kravspesifiseringsprosessen hadde som hensikt å kartlegge alle krav til ønsket funksjonalitet, men dette kapittelet viser kun en oversikt over krav som ble implementert. Oversikt over videre arbeid med applikasjonen kommer i avslutningen av rapporten.

3.1 Funksjonelle krav

3.1.1 Use cases

Siden use case diagrammer som hovedregel fokuserer på interaksjonene mellom eksterne aktører og systemet (Malan & Bredemeyer 2001), viser diagrammet kun interaksjoner trigget av brukerne. Andre funksjonelle krav er listet i kapittel 3.1.2.



Figur 3.1: Use Case diagram

Use case-beskrivelser

Use case: Lese informasjon
Aktører: Bruker
Mål: Brukeren kan finne nødvendig og relevant informasjon knyttet til å leie et utleieobjekt.
Beskrivelse: Brukeren navigerer seg frem til informasjonen ønsket, enten det er informasjon om et utleieobjekt, utleiereglement eller lignende, og leser aktuell informasjon.
Pre-betingelse: Brukeren er logget inn.

Use case: Søke om booking av utleieobjekt.
Aktører: Bruker
Mål: Legge inn søknad om booking av utleieobjekt.
Beskrivelse: Brukeren går inn på siden til utleieobjektet hen ønsker å booke. Brukeren velger deretter datoene hen ønsker å booke. Deretter må det bekreftes at utleiereglementet er lest ved å bekrefte søknaden.
Alternative scenarier:
- Dersom søknaden gjelder en periode hvor bookingsøknader ikke enda er tildelt, vil søknaden lagres for senere tildeling.
- Dersom søknaden gjelder en periode hvor bookingsøknader allerede er tildelt, vil søknaden automatisk bli tildelt.
Pre-betingelse: Brukeren er logget inn.

Use case: Avbestille booking
Aktører: Bruker
Mål: Avbestiller en registrert booking.
Beskrivelse: Brukeren velger en av sine eksisterende bookinger, klikker avbestill, og bekrefter avbestillingen. Dersom det er mindre enn en uke til oppstart av utleien skal leieprisen fortsatt trekkes fra lønn.
Pre-betingelse: Brukeren er logget inn.

Use case: Fylle ut sjekkliste (ved utsjekk)
Aktører: Bruker
Mål: Registrere om punktene på sjekklisten for utsjekk er gjennomført.
Beskrivelse: Brukeren mottar en epost med påminnelse om å fylle ut sjekkliste en dag før oppholdet utløper. Brukeren trykker deretter på linken, fyller ut sjekklisten, og klikker deretter på knappen for å sende den inn. Sjekklisten kan sendes inn ufullstendig og kan oppdateres senere.
Pre-betingelse: Brukeren er logget inn.

Use case: Sende inn tilbakemelding på nettsiden
Aktører: Bruker
Mål: Foreslå forbedringer til nettsiden.
Beskrivelse: Brukeren trykker på knappen for å gi tilbakemelding, skriver inn hvilken side tilbakemeldingen gjelder og selve tilbakemeldingen.
Pre-betingelse: Brukeren er logget inn.

Use case: Lese informasjon
Aktører: Bruker
Mål: Brukeren kan finne nødvendig og relevant informasjon knyttet til å leie et utleieobjekt.
Beskrivelse: Brukeren navigerer seg frem til informasjonen ønsket, enten det er informasjon om et utleieobjekt, utleiereglement eller lignende, og leser aktuell informasjon.
Pre-betingelse: Brukeren er logget inn.

Use case: Logge inn/ut
Aktører: Bruker, admin, Microsoft brukerautentisering
Mål: Logger inn eller ut av nettsiden.
Beskrivelse: Admin/bruker klikker på "logg inn" eller "logg ut"-knappen. Brukeren blir tatt til Microsofts innloggingsside, logger seg på der, og blir returnert inn- eller utlogget til nettsiden.
Pre-betingelse: Brukeren har en Microsoft-konto registrert hos Communicate.

Use case: Se innsendte sjekklister.
Aktører: Admin
Mål: Admin ser en oversikt over hvilke sjekklister som er fylt ut, og en detaljert visning over hva som er blitt fylt ut i den enkelte sjekkliste.
Beskrivelse: Admin går inn på siden for oversikt over sjekklister, og ser der hvilke ferdige bookinger som har sendt inn sjekkliste. Admin kan klikke på en sjekkliste for å se detaljer over hva som er fylt ut og hva som mangler.
Pre-betingelse: Admin er logget inn.

Use case: Kansellere bookinger.
Aktører: Admin
Mål: Admin kansellerer en booking.
Beskrivelse: For å kansellere/slette en booking går admin inn på oversikt over alle bookinger, velger bookingen som skal kanselleres, og bekrefter at hen faktisk ønsker å kansellere den.
Pre-betingelse: Admin er logget inn.

Use case: Administrere utleieobjekter
Aktører: Admin
Mål: Admin legger til, endrer, eller sletter utleieobjekter.
Beskrivelse: Se detaljer under beskrivelsene til de ulike extend use casene som følger under.
Pre-betingelse: Admin er logget inn.

Use case: Opprette nye utleieobjekter.
Aktører: Admin
Mål: Admin oppretter et nytt utleieobjekt.
Beskrivelse: Admin går inn i skjemaet for å opprette et nytt utleieobjekt, fyller inn nødvendig informasjon samt ønsket ekstra informasjon, og velger lagre. Når dette er gjennomført dukker utleieobjektet opp som mulig å booke for alle brukere.
Pre-betingelse: Admin er logget inn.

Use case: Slette eksisterende utleieobjekter.
Aktører: Admin
Mål: Admin sletter et utleieobjekt.
Beskrivelse: Admin går inn på utleieobjektet hen ønsker å slette, trykker på knappen for å slette, og bekrefter at hen virkelig ønsker å slette utleieobjektet.
Pre-betingelse: Admin er logget inn.

Use case: Modifisere eksisterende utleieobjekter.
Aktører: Admin
Mål: Admin modifiserer et utleieobjekt.
Beskrivelse: Admin går inn på utleieobjektet hen ønsker å modifisere, trykker på knappen for å modifisere, og endrer ønsket innhold. Alt innholdet til et utleieobjekt kan endres, både beskrivelser, bilder, søknadsfrister og perioder, samt sjekklister osv.
Pre-betingelse: Admin er logget inn.

3.1.2 Andre funksjonelle krav

- Applikasjonen skal automatisk tildele bookinger, dette gjøres basert på:
  - Et poengsystem ment for å gjøre at utleien blir forholdsvis rettferdig fordelt mellom de ansatte. Det er en del teknikaliteter om hvordan dette skal fungere, som er utdypende forklart i notatene i vedlegg F.
  - Tildelingsperioder med tilhørende søknadsfrister. Søknader som kommer inn før søknadsfrist tildeles automatisk når søknadsfrist er passert, basert på poengsystemet nevnt i punktet over. Søknader som kommer inn midt i en tildelingsperiode tildeles automatisk ettersom de kommer inn.
- Applikasjonen skal automatisk varsle brukere via epost. Ønskede varslinger er:
  - Varsel om at søknadsfrist nærmer seg.
  - Varsel/bekreftelse om at booking er innvilget.
  - Bekreftelse på kansellering av booking.
  - Månedlig varsel til admin om hvilke bookinger det skal trekkes lønn for.
- Applikasjonen skal automatisk øke en brukers poeng på årsdag for ansettelse.
- Applikasjonen skal benytte brukernes eksisterende Microsoft-kontoer for innlogging.

3.2 Krav til sikkerhet

- Det er krav til at uautoriserte applikasjoner ikke skal ha tilgang til å gjennomføre REST-kall mot tjenestene, slik at det unngås å eksponere data til ukjente kilder. Det betyr at kun dette systemet skal kunne benytte funksjonaliteten som eksister.
- Som følge av GDPR og et ønske om å beskytte sine ansatte skal personvern ivaretas. Det betyr i hovedsak at det skal lagres så lite personidentifiserende data som mulig, samt at det beskyttes på en god måte der det må lagres.

3.3 Andre krav

- Da dette er en interntjeneste for selskapet, ønsker Communicate at tjenesten skal ha så lave driftskostnader som mulig. Det betyr i praksis at hvor mulig velges Azure-ressursene med lavest kostnad som kan benyttes samtidig som applikasjonen har den funksjonaliteten som trengs.
- Tjenesten skal ha en god brukeropplevelse, og gjøre det enkelt for brukeren å finne frem og utføre ønskede operasjoner. Selskapet hadde ikke

noe mer spesifikt krav enn dette, så vi tar utgangspunkt i Web Content Accessibility Guidelines (WCAG) basert på hva selskapet har behov for, og designprinsipper fra Material Design - et designsystem laget av Google.

- Tjenesten er tenkt benyttet av selskapet etter at utviklingsperioden er ferdig, og vil vedlikeholdes og sannsynlig videreutvikles av interne utviklere i selskapet. Det betyr at applikasjonen må være godt dokumentert slik at det er lett for nye utviklere å begynne (videre)utvikling av tjenesten.
- Oppetid og lignende håndteres automatisk av Azure. Det skal likevel settes opp overvåkning av tjenesten slik at det varsles dersom problemer oppstår.
- Tjenesten skal som et minimum fungere på nettleserne Chrome og Firefox. Det er ønskelig at det også fungerer på Edge, men dette er ikke et absolutt krav. Det er ønskelig at tjenesten fungerer generelt på mobil, men med ønske om at sjekklisten er mobilvennlig.

Kapittel 4

Teknologier

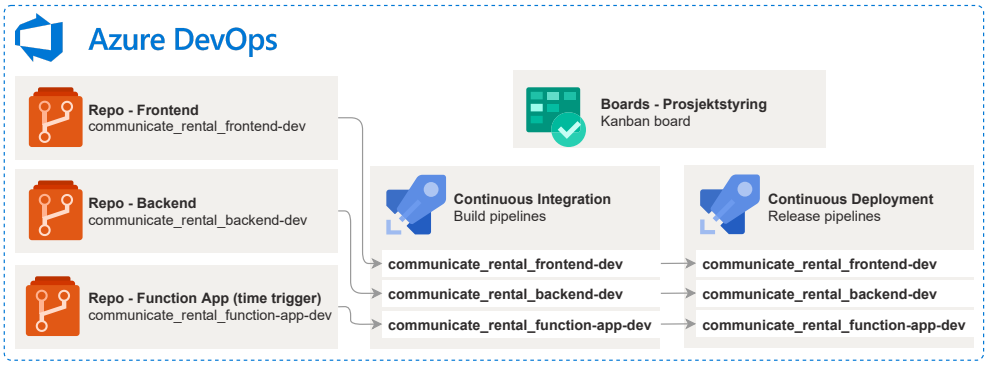
Vi ble tidlig fortalt at Communicate sin plan for prosjektet var å ferdigstille nettsiden for bruk etter at vårt bachelorprosjekt var ferdigstilt. Siden selskapet spesialiserer seg i .NET-utvikling, var det derfor naturlig at det var krav til bruk av teknologier i .NET-sfæren. Nevnt i oppgaveteksten (vedlegg A) var det krav om bruk av .NET med C#, Azure, Azure DevOps, samt Visual Studio/Visual Studio Code og "andre relevante Microsoft-løsninger og tjenester". Frontend-rammeverk sto vi fritt til å velge selv.

4.1 Azure- og .NET-sfæren

4.1.1 Azure

Azure er skyplattformen til Microsoft, og gir tilgang til tjenester knyttet til alt fra prosjektstyring til produksjon, overvåkning av tjenester og lagring (Microsoft 2019). Videre følger en oversikt over hvilke tjenester vi har benyttet, og hvorfor.

DevOps: Azure DevOps



Figur 4.1: Oversikt over hvordan gruppen har benyttet DevOps (egenlaget).

Å benytte Azure DevOps var klart spesifisert som et krav i oppgaven. Azure DevOps er et utviklingsverktøy som fokuserer på smidig utvikling gjennom samarbeid, bygging, testing og utgivelse (deployment) av kode (Microsoft 2021*m*). Det er en skyplattform fokusert på versjonskontroll i likhet med GitHub, BitBucket og andre lignende tjenester. Gruppen benyttet følgende verk-

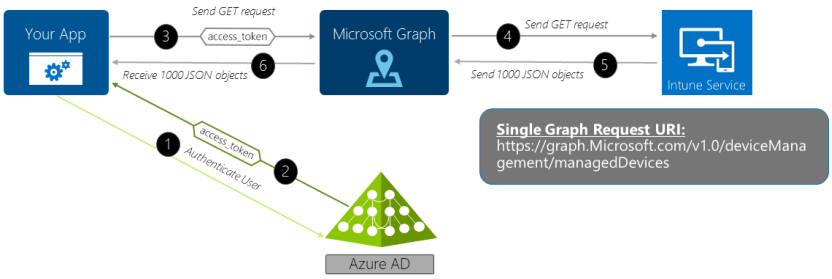
tøy som en del av Azure DevOps: Kanban-tavlen og planleggingsverktøyene i Azure Boards, Azure Repos hvor vi hadde repositories med koden i, samt Azure Pipelines for Continuous Integration og Continuous Deployment (CI/CD) ved bruk av build- og release-pipelines. En oversikt over dette kan sees i figur 4.1.

Brukerhåndtering: Azure Active Directory

Azure Active Directory (heretter Azure AD) er Microsofts tjeneste for identitets- og tilgangshåndtering. Azure AD gjør det enklere å konfigurere hvilke brukere som skal ha tilgang til ulike tjenester, gjør det blant annet mulig å benytte innlogging via Microsoft-kontoer (Microsoft 2020*j*). Applikasjonen vår benytter Azure AD for pålogging og håndtering av hvilke brukere som har tilgang til hva i APIet (se 4.1.1), samt den tilhørende tjenesten Microsoft Graph.

Brukerhåndtering: Microsoft Graph

Microsoft Graph (MS Graph) tillater henting av brukerinformasjon (som ligger lagret hos Microsoft) for brukere som ligger innenfor de definerte grensene i Azure AD (Microsoft 2021*e*). I vårt tilfelle var de definerte grensene brukere tilknyttet Communicate. Når oppdragsgiver registrerer en ny bruker tilknyttet selskapet blir dette oppdatert i Azure AD, og det blir mulig for oss å hente ut informasjonen som ligger lagret i det aktuelle endepunktet (Microsoft 2021*a*). For å hente informasjonen må en tilgangsnøkkel (access token) foreligge. Dette kan hentes gjennom en forespørsel til selskapets Azure AD, som sjekker at kallet kommer fra en godkjent kilde (punkt 1 og 2 i figur 4.2).



Figur 4.2: Oversikt over Microsoft Graph (Hornbech 2021).

Etter at en tilgangsnøkkel foreligger, kan man begynne å gjøre GET-kall på brukerinformasjonen tilknyttet selskapets brukere (punkt 4 og 5 i figuren).

API-håndtering: Azure API Management

Kontaktpersonen vår i Communicate foreslo også at vi satt opp Azures API Management (APIM) for tilgangsstyring i APIet. På denne måten ble APIet beskyttet mot kall fra uvedkommende. APIM tillater enkel - men omfattende -

konfigurasjon av APIer, hvilke typer kall det skal motta, mulighet til å verifisere API-nøkler, mulighet til oppsett slik at kun enkelte brukertyper (satt opp i Azure AD) kan benytte ulike endepunkter etc. (Microsoft 2017*a*).

Vi har satt opp backend på en slik måte at det kun er kall som kommer fra APIM som tillates i backend, mens APIM er satt opp slik at det kun tillates kall fra tjenesten som kjører frontend og Function Appen (mer om denne videre i kapittelet). Vi benytter altså APIM som et mellomlag for verifisering.

Databaser: Azure Storage Account

Applikasjonen vår hadde naturlig nok behov for database-lagring. I tillegg var det klart at vi ville ha behov for lagring av objekter med ulik kompleksitet og av ulike typer. Gruppen vurderte det som lite sannsynlig at det ville bli vesentlige datamengder i systemet. Ettersom oppgaven spesifiserte bruk av andre relevante Microsoft-løsninger og tjenester fokuserte vi på database-tilbudene som var tilgjengelige innen Azure-sfæren. Azure har tre databasetyper de fokuserer på - Azure SQL database, Azure Storage Account og Azure Cosmos DB. Kort beskrevet er:

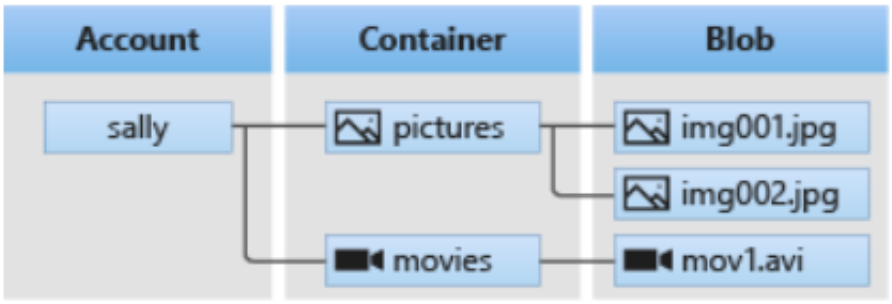
- Azure SQL er en tradisjonell relasjonsdatabase (Microsoft 2021*o*)
- Storage Accounts er en noSQL database som tillater ulike typer lagring relativt billig, men med begrensede filtrerings- og spørringsmuligheter (Microsoft 2021*j*)
- Cosmos DB ligner Storage Accounts men er dyrere, har høyere respons og flere spørringsmuligheter (Microsoft 2021*b*)

Azure SQL database kunne nok vært et godt alternativ når det kommer til dataobjekter med slik kompleksitet som beskrevet (Microsoft 2021*l*). Ulempen med en relasjonsdatabase kan i noen tilfeller være komplekse spørringer for å hente ut data, samtidig som spørringene - dersom ikke sikret godt nok - gir mulighet for SQL-injection.

En annen mulighet var Azure Table Storage - en tjeneste som leveres som en del av Azure Storage Account. Table Storage er en kostnadseffektiv noSQL database som regnes som en rimeligere løsning enn SQL-lagring (Microsoft 2021*h*), men som gir noen utfordringer i forhold til nestede objekter. Table Storage benytter LINQ-queries, som enten i stor grad kan ligne på SQL-queries, eller benytte lambda-uttrykk slik at man kan skrive kode som filtrerer hvilke elementer som skal hentes basert på sammenligning av parametre - mye likt det man gjør i en standard if-sjekk (Shet 2017) (se eksempler i kapittel 6.2.2).

Et annet aspekt ved lagringen er at bilder tilhørende feriestedene må lagres. Dette har Azures SQL database støtte for, men dette ansees som en kostbar løsning da datamengden bestemmer prisen for tjenesten (Microsoft 2021*f*). Table Storage har ikke støtte for dette, men som en del av Azure Storage Account tilbys Blob Storage - en lagringsmetode som er laget for filer. Figur 4.3 viser hvordan Blob Storage muliggjør lagring av blob-er (filer) i ulike containere, slik at de kan kategoriseres. I tillegg gir dette mulighet for henting av URLer fra

databasen (Microsoft 2021*c*), slik at de kan hentes gjennom et HTTP-kall mot backenden, som gjør visning av bildefilene enklere å håndtere i frontend.



Figur 4.3: Blob Storage (Microsoft 2020*b*).

Cosmos DB ble valgt bort som følge av kostnad, samtidig som vurderingen av Azure SQL og Azure Storage Accounts tilsa at begge disse løsningene ville være gode nok for systemet.

Kombinasjonen SQL-/Blob-storage ville redusert kostnaden noe, men vil da kreve både tradisjonell SQL spørring i tillegg til mapping for å holde blob-databasen oppdatert. SQL-databaser kunne fungert greit for oss, men ble valgt bort grunnet

- ønske fra oppdragsgiver om lavest mulig driftskostnader,
- at programmet vårt ikke har veldig komplekse behov innen sortering av innformasjonen som blir hentet,
- at Blob Storage tillater henting av URLer som kan vises direkte eksternt,
- at både Table Storage og Blob Storage er en del av det forholdsvis rimelige Storage Account-tilbudet som gjorde at all informasjon kunne hentes på omtrent samme måte med samme nøkler.

Vi valgte derfor å benytte oss av Azure Storage Accounts - og dermed kombinasjonen Table og Blob Storage - fremfor Cosmos DB og SQL.

Deployment: Azure App Services

Oppdragsgiver ønsket også at CI/CD i Azure ble benyttet gjennomgående gjennom prosessen, derfor ble dette satt opp allerede i oppstart av prosjektet. Fordelen med dette er at de ulike delene av applikasjonen har vært tilgjengelig i Azure fra tidlig i prosessen. Den største hovedtjenesten Azure tilbyr innen deployment er App Services. Innenfor App Service-kategorien finnes det en rekke ulike tjenester, som har ulikt kostnadsnivå og funksjonalitet (Rajendran 2020).

Undersøkelser viste at det billigste er en tjeneste som kjører kun når den trigges - Function Apps. Disse kan trigges av ulike faktorer. Eksempler på slike faktorer er et HTTP-kall, filoppdatering eller på et spesifikt klokkeslett.

Function Apps er laget for å reagere på en trigger, kjøre kode, og deretter stenge ned igjen. Det er mulig å lage APIer med Function Apps, hvor hvert endepunkt er en Function, men fordi den stenger ned automatisk kan det føre til treghet ved lasting av siden fordi man får cold starts som krever ekstra ressurser i form av tid (Microsoft 2020*c*).

Et annet design som Azure tilbyr er Web-App-er. Web App kjører kontinuerlig og unngår dermed utfordringen med cold starts. Det er også verdt å nevne at dette er det eneste av de tre alternativene som har mulighet for å støtte UI-applikasjoner utviklet i eksternt rammeverk. Azure har også en tjenestetype som heter Azure Web API - dette er en tjenestetype som henger igjen fra tidligere og som i praksis er lik Web Apps (Rajendran 2020).

Den dyreste tjenesten er Logic Apps, som er laget for at ikke-programmerere enkelt skal kunne sette opp automatiserte arbeidsflyter (workflows) hvor integrasjon mellom ulike plattformer er fokus (Microsoft 2021*n*) - for eksempel å trigge at en epost skal sendes hvis et databaseobjekt oppdateres. Fordi hovedpoenget med prosjektet vårt var å skrive kode, var dette en tjeneste som var uaktuell for oss.

Etter en diskusjon kom vi frem til at det beste for oss var å sette opp en Function App som gjør kall til backend basert på en tidstrigger. Function Appen kan på denne måten styre de logiske operasjonene i systemet. I tillegg ble det bestemt at løsningen skulle utgis som to ulike Web App-er - en for frontend og en for backend. En annen løsning som ble vurdert var å kjøre backend som en Function App, som ville blitt noe billigere enn en Web App. Vurderingen tilsa at dette ville komplisert prosessen og gitt oss problematikk knyttet til cold starts under informasjonsinnhentingen. Derfor endte vi på en Web-App også for backend (se kapittel 9 for ytterligere diskusjon rundt valget).

4.1.2 .NET 5

Selv om det var et krav om å benytte .NET med C#, er det en omfattende sfære av ulike plattformer og rammeverk som er laget av Microsoft. Gruppen måtte derfor ta et valg knyttet til hvilken versjon vi ønsket å benytte.

Orginalt var fokuset til .NET på Framework-versjonen, som ga muligheten til å utvikle applikasjoner til Windows. Senere kom .NET Core som er en cross-platform versjon av .NET som kan kjøres på alle tre store OSene. I tillegg er det en rekke mindre versjoner til spesifikke plattformer (Warren 2018). I november 2020 kom .NET 5, som er et forsøk på å samle de ulike .NET-rammeverkene til et, slik at NuGet-pakker blir kompatible på alle plattformer som benytter .NET i fremtiden (Lander 2019).

Gruppen landet på å benytte .NET 5 som er den nyeste versjonen, til tross for at det ikke er planlagt langtidsstøtte (LTS) (Lander 2019), da vi anså at det sannsynligvis er enklere å konvertere fra .NET 5 til en senere LTS-versjon, enn fra en tidligere versjon.

4.2 Angular 11

I frontend sto vi friere til å benytte de verktøyene vi ønsket. Gruppen fikk av kontaktpersonen vår i Communicate en anbefaling om Angular - et TypeScript-basert rammeverk som ble initialisert av Google og er et åpen kildekode-prosjekt. Denne anbefalingen ga hun fordi hun selv hadde kunnskap til rammeverket, og derfor kunne hjelpe oss ved behov. Gruppen hadde allerede kjennskap til at Angular var et mye brukt rammeverk, og den ble dermed tatt med i en vurderingen. I tillegg vurderte gruppen Vue og React - andre rammeverk som også er mye brukt.

To av gruppens medlemmer har fra tidligere erfaring med Polymer/LitElement-rammeverket, som også er et åpen kildekode-prosjekt initialisert av Google. Polymer er bygget opp mye likt som Angular i form av en komponentbasert tilnærming som har en struktur som sterkt lener seg mot bruk av MVC-patternet (mer om dette i 5.4.1). Forskjellen på de to er at Polymer i stor grad fokuserer på å være et lite rammeverk/bibliotek for å utvikle web-komponenter, mens Angular er utviklet for å fungere som et fullverdig rammeverk for applikasjonsutvikling. Likevel deler de to mange likheter i fokus på komponenter og måten filstrukturen fungerer på (Joshi 2016).

Som følge av at gruppen hadde fått anbefaling om bruk av Angular, samt at to gruppemedlemmer allerede hadde kjennskap til Polymer - men ønsket å utvide kunnskapen sin-, landet gruppen på å bruke Angular. Angular er et komplekst rammeverk og kan dermed være tidkrevende å lære. Samtidig kan det beskrives som et modent og velutviklet rammeverk-, som et resultat er problemstillinger ofte beskrevet av ulike kilder (AltexSoft 2020). Derfor anså vi dette som overkommelig.

Angular legger opp til bruk av TypeScript, et strengt syntaktisk supersett av JavaScript som er utviklet og vedlikeholdt av Microsoft, samt templates basert på HTML med Angular-spesifikk syntaks og funksjonalitet.

Styling

Ren HTML gir lite i forhold til utseende i brukergrensesnittet, så CSS-styling må til for å få til et godt design. For CSS-styling valgte vi å benytte SASS - en preprosessor til CSS. I tillegg til vanlig CSS-syntaks gir SASS blant annet mulighet for delte variabler på tvers av flere SASS-filer (W3Schools 2019), noe som gjorde det enklere for oss å holde farger, størrelser og lignende konsekvente mellom ulike komponenter - noe som igjen resulterte i et mer helhetlig designuttrykk.

Da ingen på gruppen har erfaring innen web-design, UX eller lignende, valgte vi også å benytte en del ferdige komponenter og verktøy for å gjøre arbeidet enklere, blant annet Bootstrap. Bootstrap er et utbredt rammeverk som gjør det enklere å utvikle responsive nettsider, med både ferdiglagde komponenter og utility-klasser (Bootstrap 2021). I tillegg benytter vi enkelte andre ferdiglagde komponenter.

Kapittel 5

Design

I kapittel 5.1 gis en oversikt over brukertyper som legger grunnlaget for mye av designet. I kapittel 5.2 beskrives personverndesignet i prosjektet, og i kapittel 5.3 diskuteres det hvordan applikasjonen er designet og den overordnede strukturen, hvilke designmønster som er benyttet i de ulike delene og lignende. Videre diskuteres brukergrensesnittet og dets progresjon i kapittel 5.5.

5.1 Brukertyper

5.1.1 Vanlig bruker

I denne applikasjonen vil en vanlig bruker tilsi en bruker uten adminprivilegier. En bruker skal som beskrevet i de forskjellige use case-senariene ha tilgang til ulike informasjonssider, ha mulighet til å søke om booking av utleieobjekt, samt avbestille godkjente bookinger, sende tilbakemeldinger på nettsiden og kunne se alle sine aktive søknader og bookinger. En vanlig bruker har ikke tilgang til noen av de administrative rettighetene.

5.1.2 Administrator

Admin vil ha samme muligheter som en vanlig bruker, samt ha en del tilleggsfunksjonksjonalitet. Admin skal også kunne endre på alle informative tekster, samt sjekklisten for utsjekk av utleieobjekt og datoene for søknadsperioder. Hen skal kunne endre, slette og opprette bookinger om dette skulle være nødvendig. De vil være ansvarlige for å holde informasjonen i applikasjonen oppdatert.

5.2 Personvern

Norge har en personopplysningslov som i 2018 ble styrket av at Personvernforordningen (GDPR) trådte i kraft. Som følge av dette har alle bedrifter som behandler persondata en rekke plikter i forbindelse med håndteringen av disse (Datatilsynet 2019*a*). Datatilsynet har laget en oversikt over hvilke prinsipper som må tas hensyn til (Datatilsynet 2019*b*). Disse følger i en liste, med forklaring på hvordan de ulike prinsippene er designet håndtert i denne applikasjonen:

Lovlig, rettferdig og gjennomsiktig: For å lovlig behandle personopplysninger må det eksistere et aktuelt behandlingsgrunnlag. Et slikt behandlingsgrunnlag er samtykke - noe organisasjonens admin har gitt i Azure, slik at vi har tilgang til å behandle brukerne.

Gjennomsiktighet blir også dekket på denne måten, ettersom det beskrives hvilken data som benyttes. Rettferdighet kan i denne sammenheng beskrives som prinsippet om at behandlingen av personopplysninger skjer i respekt for brukerens interesser og rimelige forventninger. Dette dekkes ved at det i applikasjonen blir informert over hvilke data som lagres og hvordan den benyttes.

Formålsbegrensning: Dette prinsippet handler om å kun benytte dataene til det formålet den er hentet inn for i utgangspunktet. Dataene hentes inn basert på tilganger knyttet til applikasjonen som gis i en Microsoft-portal som gir adgangsnøkler, slik at dette ikke kan hentes inn i andre applikasjoner.

Dataminimering: Prinsippet handler om å begrense mengden innsamlede personopplysninger til det som er nødvendig for å realisere formålet med innsamlingen. Dataene som lagres om en bruker er årsdag for ansettelse og nåværende poeng, slik at tildeling kan skje rettferdig.

I tillegg lagres Id-en til brukerens Microsoft-konto for å holde styr på brukerne. Lagring av Id fremfor feks e-post har vi valgt som et resultat av at vi mener at id-en er vanskeligere å knytte opp mot en identitet. En brukers ID kan kun hentes gjennom organisasjonen, som brukeren tilhører, sitt Azure AD - da med godkjennelse fra organisasjonens admin. Dette gjøres gjennom å godkjenne at en applikasjon kan lese sin egen profil, eller ved å hente den selv. Det betyr at visning av navn, samt utsending av epost, gjøres ved å hente den nødvendige brukerinformasjonen fra MS Graph (se kapittel 4.1.1 for info om MS Graph) basert på IDen.

Riktighet: Personopplysninger som behandles skal være korrekte, og om nødvendig oppdateres. En bruker kan selv oppdatere egen informasjon ved å endre den gjennom sin Microsoft-bruker. I tillegg kan innformasjon lagret i databasen rettes gjennom admin dersom feil blir oppdaget.

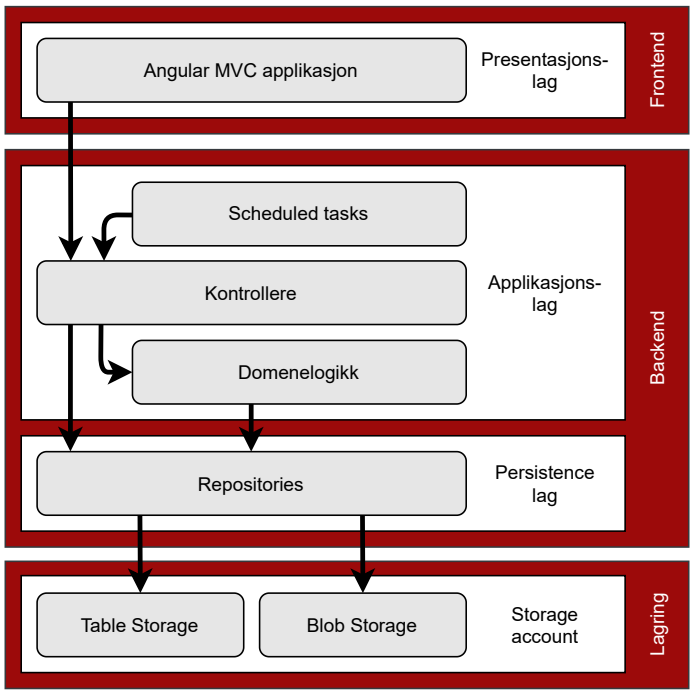
Lagringsbegrensning: Dette prinsippet handler om at personopplysninger skal slettes eller anonymiseres når de ikke lenger er nødvendige for formålet. Dette vil delvis skje automatisk når en Microsoft- bruker slettes, slik at det ikke lenger er mulig å hente personopplysninger basert på den lagrede dataene i systemet. Admin har også mulighet til å slette brukerdataene som lagres i databasen dersom det er ønskelig, - uavhening av Micrsoft -brukeren.

Integritet og konfidensialitet, ansvarlighet: Disse prinsippene handler om at de som behandler personopplysninger må iverksette tiltak mot utilsiktet og ulovlig ødeleggelse, tap og endringer av personopplysninger, samt å ikke bare ha ansvar - men ta ansvar - for behandling av personopplysninger. Dette gjøres her ved å kun gi tilgang til endring av data av en selv eller admin, ved å lagre så lite som mulig om en bruker, og å sikre at ikke uvedkommende har tilgang til dataene - noe vi har brukt APIM til (se kapittel 4.1.1). Et godt prinsipp å forholde seg til er å la store aktører som har budsjett og tid til å

håndtere sikkerhet gjøre det for en (Maddox 2021), da få har ressurser til å faktisk gjennomføre de omfattende sikkerhetsprosedyrene som er nødvendige for god håndtering (OWASP 2021).

5.3 Overordnet arkitektur - lagdelt

Prosjektet har flere ulike deler, og det er derfor nødvendig med et overordnet arkitekturmønster. Både lagdelt arkitektur og Clean Architecture ble vurdert som gode alternativer for prosjektet (Microsoft 2020*a*). Clean Architecture ble valgt bort med bakgrunn i at vi ikke har god nok kjennskap til denne modellen. Et annet moment som var førende for valget var at det naturlig ville bli en fom for lagdelt modell uansett ettersom det benyttes REST-kall mellom de ulike delene av prosjektet. Gruppen landet på en versjon av en lagdelt arkitektur, hvor enkelte lag er splittet opp til ulike del-tjenester. Dette er kun den overordnede modellen, da både frontend og backend bruker egne, mer spesifikke, designmønster internt.



Figur 5.1: Overordnet lagdelt struktur

Den lagdelte modellen er et mye utbredt arkitekturmønster (Richards 2015), også for bruk i Single Page Applications (Microsoft 2020*a*) slik denne applikasjonen er (se kapittel 5.4.1). Vi valgte denne arkitekturen fordi det er en utbredt arkitektur ved bruk av REST-kall mellom komponenter. Vi kunne også valgt en microservicearkitektur, men fordi applikasjonen ikke er av voldsom størrelse anså vi det som mer overhead enn nyttig, særlig siden ingen på gruppen

tidligere har benyttet arkitekturen.

I tillegg til å skape et fremtidsrettet design gjennom uavhengige komponenter som er mulig å bytte ut, vil vår løsning i Azure (tre separerte App Services som samhandler - se kapittel 4.1.1) gjøre at ulemper knyttet til å re-publisere hele applikasjonen på nytt ved endringer bli minimal. Det samme gjelder skalerbarhet som ofte pekes på som en potensiell utfordring. Dette er har vi valgt å løse gjennom metoden vi benytter for produksjonssetting av applikasjonen. Dermed får vi fordelene ved enkel utvikling og gode testmuligheter uten veldig store negative konsekvenser (Richards 2015). Hovedutfordringen ligger først og fremst i og designe uavhengige lag (Sommerville 2004).

En oversikt over hvordan lagene er bygget opp i vår applikasjon kan sees i figur 5.1.

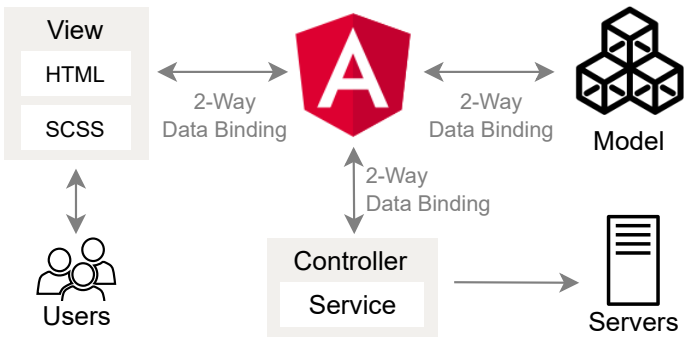
5.4 Systemdesign

5.4.1 Frontend

SPA

Angular-rammeverket er fokusert rundt Single Page Application (heretter SPA) metoden for utvikling av frontend. Hovedfordelen med denne metoden er at applikasjonen blir lastet en gang, deretter endres innholdet uten at siden lastes på nytt. Det eneste som lastes inn er eventuell data som må hentes for å fylle den nye visningen, men ikke selve strukturen til nettsiden. SPA støtter muligheten for lav kobling mellom backend og frontend, som oppfyller kravet som omhandler videreutvikling av systemet gjennom "Separation of Concerns"-prinsippet (mer om dette i kapittel 5.4.2). En annen fordel ved SPA er at denne typen midlertidig lagring begrenser ressursbruken mot backend, noe som igjen antas at vil gjøre applikasjonen mer responsiv.

MVC

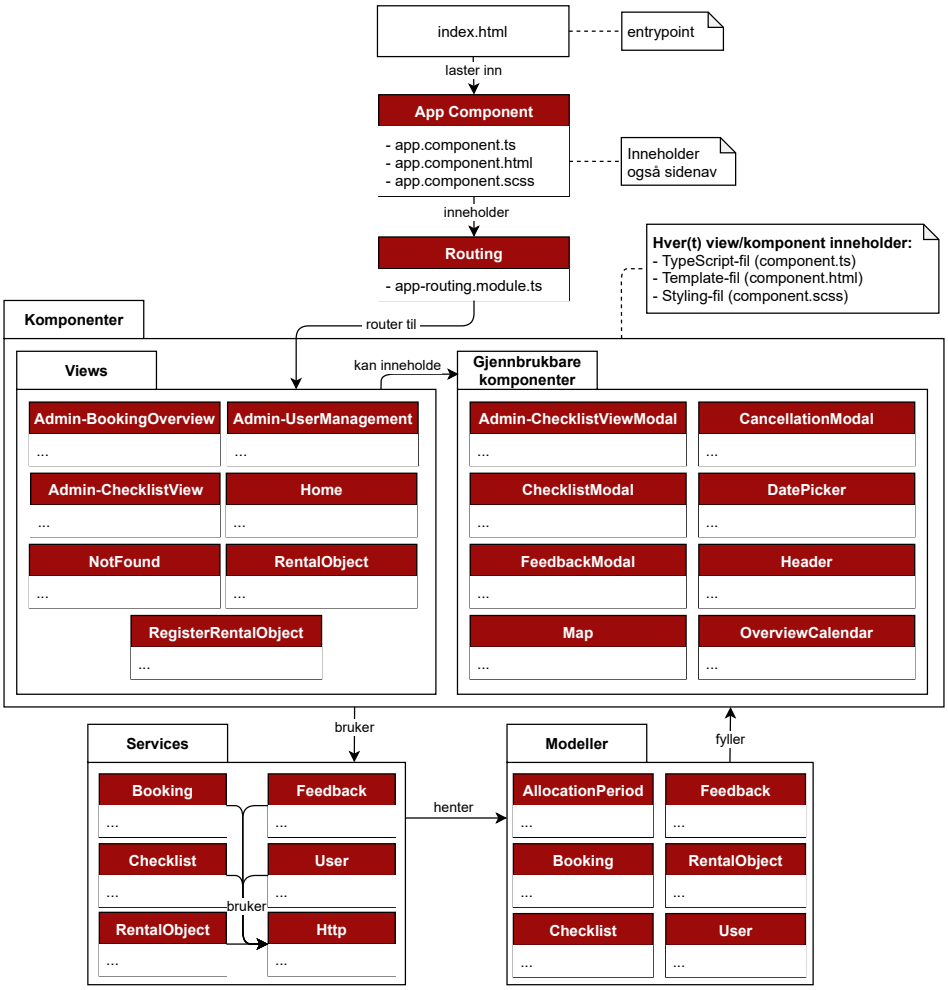


Figur 5.2: Oversikt over Angular og MVC (adapsjon av Kumar (2020)).

Angular er i stor grad bygget opp til at man følger et komponentbasert Model-View-Controller designmønster (heretter MVC). Som illustrert i figur 5.2 legges det opp til at bruk av komponenter som inneholder et view, som igjen består av (Angular-beriket)HTML og SCSS. I tillegg eksisterer det modeller som ligger separat fra disse komponentene, samt kontroller-komponenter (internt kalt Services). Kontroller-komponentene har hovedansvar for http-kall og dermed håndterer data som skal fylle modellene og brukes av viewsene (Caballero 2019).

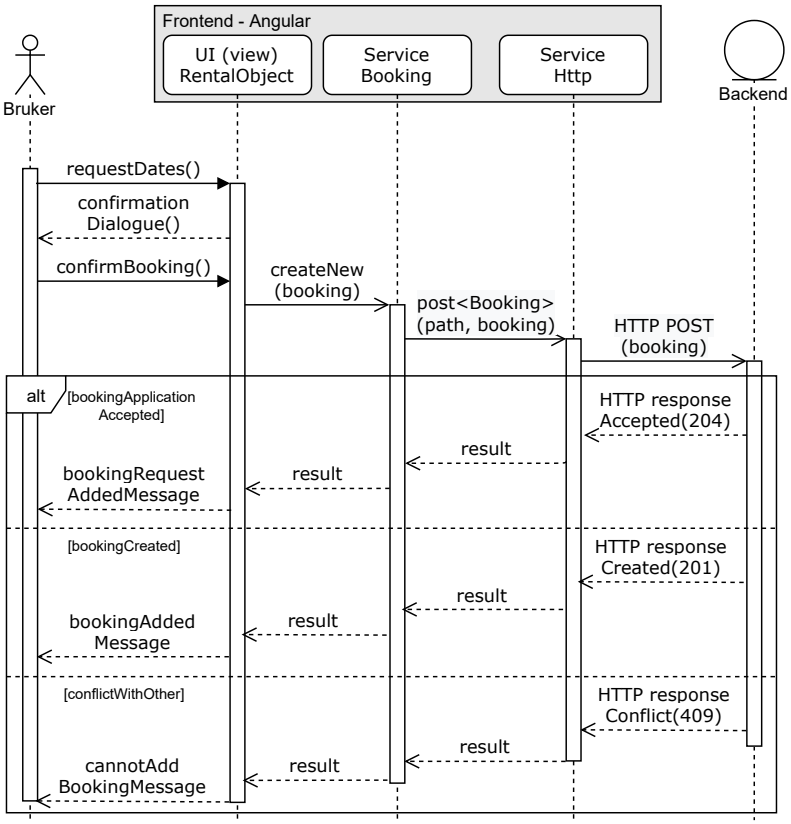
Dette designet gjør det enkelt å endre utseendet til en komponent uten å miste funksjonalitet. Samtidig er det en metode som gjør det enkelt å legge til ny funksjonalitet og nye sider underveis, noe som gjør designet ideelt for smidig utvikling.

Struktur



Figur 5.3: Forenklet oversikt over frontend

Figur 5.3 viser kodestrukturen i frontendprosjektet, kodestrukturen blir autogenerert i Angular ved bruk av CLI-kommandoen `ng new`. Index.html-filen er inngangspunktet brukeren lander på når hen åpner siden. Denne filen laster modulen som heter App Component, som inneholder sidenavigeringen, samt benytter header-komponenten og routeren. Routeren har ansvaret for å rendre views-ene. Gjenbrukbare komponenter er brukt til mindre deler av et view, slik som modaler, datovelger, headeren og kart. Disse kan bli rendret inn i de views-ene som måtte trenge dem, og kan benyttes av flere views. Dataen som vises hentes av service-ene, som fyller modellene som komponentene bruker.



Figur 5.4: Sekvensdiagram for flyten ved ny booking i frontend

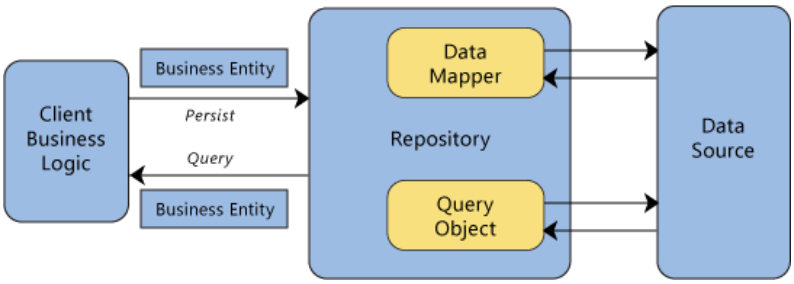
Et eksempel på flyt mellom komponentene benyttes kan sees i figur 5.4. Diagrammet viser frontend-prosessen fra en booking blir forespurt av brukeren til hen får respons på forespørselen. Her ser vi hvordan views blir brukt i samhandling med en service for bookinger ("Service Booking" i figuren) for å lage en ny bookingforespørsel. Servicen for bookinger kaller deretter servicen som oppretter Http-kall mot backend ("Service Http" i figuren). HTTP-responsen opprettes i backend, og en passende melding blir returnert til brukeren via UIet.

5.4.2 Backend

Repository pattern

Som nevnt i kapittel 4.1.1 benytter systemet en storage account med to typer lagring - Blob Storage og Table Storage. Backend inneholder naturlig nok logikk for å kunne hente data fra begge databasene, mens frontend har behov for å motta objekter som inneholder både bilder og data fra tabellagring (f.eks. data om et utleieobjekt inkludert bilder). Det er derfor nødvendig å aggregere data fra de ulike lagringsmetodene, før det sendes som et kombinert objekt til frontend.

På bakgrunn av dette, samt ønske om høy testabilitet (testability), vurderte gruppen at å benytte et repository designmønster i backend ville være hensiktsmessig. Dette mønstret legger til repository-ene som mellomlag mellom kontrollerne og datakildene, og disse kan enten deles etter business objekter, eller basert på databaseobjekter (Smith 2020). Som en del av dette designmønstret benyttes mappere. Disse separerer dataen som mottas til formen den lagres på i databasen, og kombinerer den igjen for å sende den ut. Et eksempel på dette kan sees i figur 5.5.

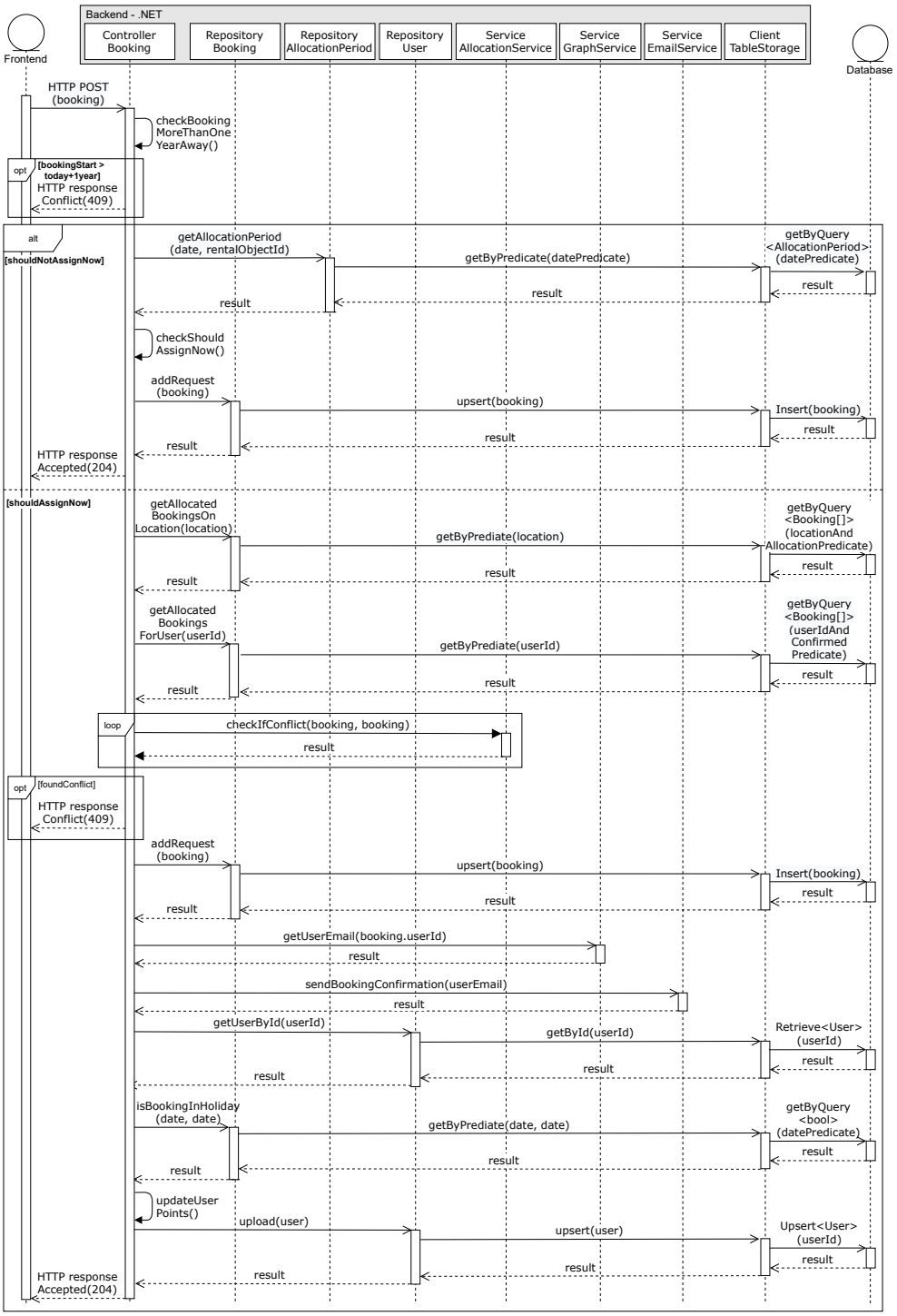


Figur 5.5: Oversikt over flyten i et repository designmønster (Kudchikar 2019)

I tillegg har vi valgt å benytte generiske klienter som repository-ene kaller, slik at det kun er disse som trengs å byttes ut dersom databasetypen byttes (Kudchikar 2019).

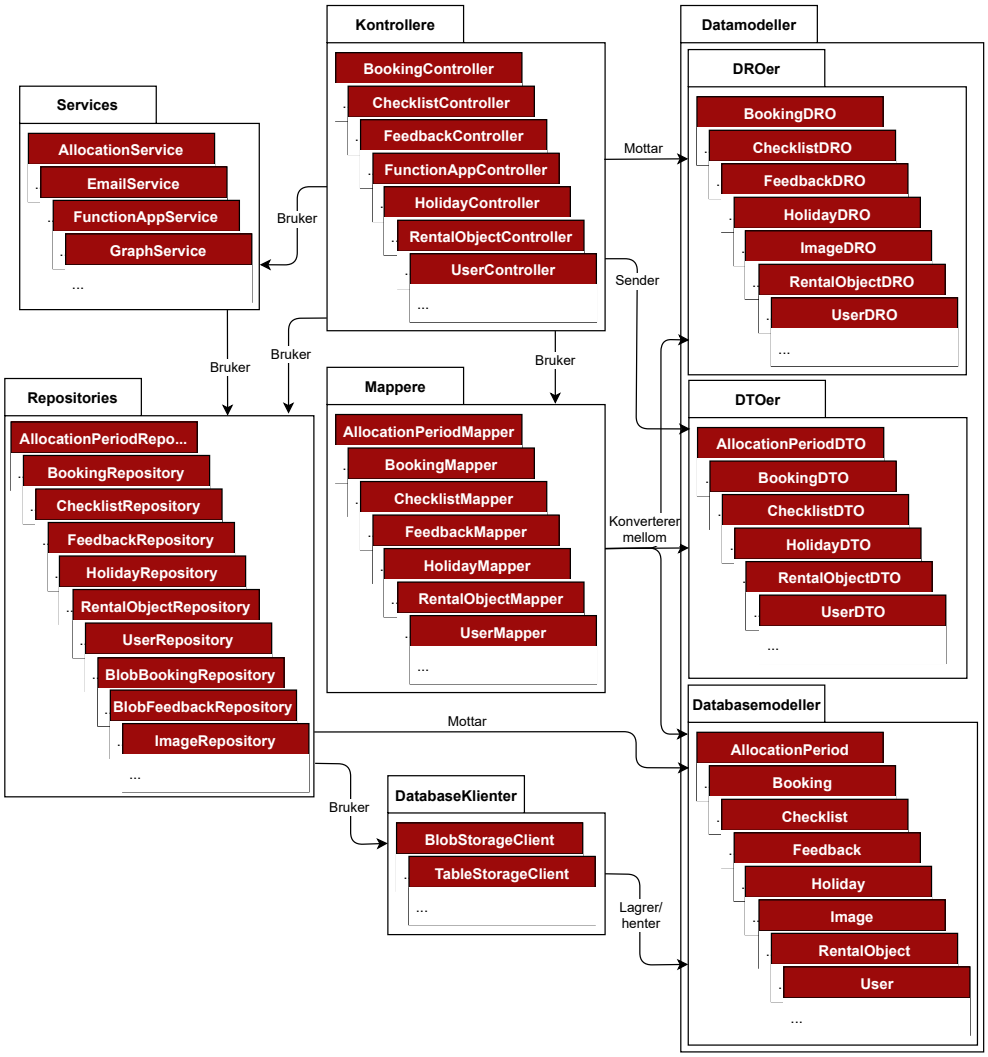
Dette mønsteret tillater konstruksjon av uavhenige komponenter mellom mapping og domenelogikk - noe som er sentralt for designprinsippet "Separation of Concerns". Mønsteret gjør det også lett å teste hvert lag separat. Samtidig gjør dette at man senere kan endre lagringsmetode uten å måtte endre API-endepunkter. Mønsteret gir et design som fører til at nettsiden kan kjøre så billig som mulig, samtidig som endring av databasetype ved behov på et senere tidspunkt kun vil kreve spesifikke lokale endringer knyttet til kommunikasjon med databasen.

Et annet viktig moment innen temaet "Separation of Concerns" er hvordan vi beskytter databasemodellene. Dette har vi løst gjennom et design der vi lager egne modeller for transporteringen av data til og fra APIet i tillegg til



Figur 5.6: Sekvensdiagram for flyten ved ny booking i backend

databasemodellene. Beskyttelsen av modellene gjøres med en modellendring i form av mapping som benyttes i kontrolleren. Objektet blir deretter videresendt til sitt repository, så lagt til, oppdatert, slettet eller hentet fra databasen. Et eksempel på denne prosessen er vist i figur 5.6. Figuren viser hva som skjer i backend når en bruker legger til en ny booking. Her vises tydelig hvordan kallet går via kontrolleren, så mappes, for deretter å benytte bookingsrepositoryet for samhandling med databasen, og det utføres logikk basert på dataen som hentes.



Figur 5.7: Forenklet klassediagram som viser en oversikt i backend

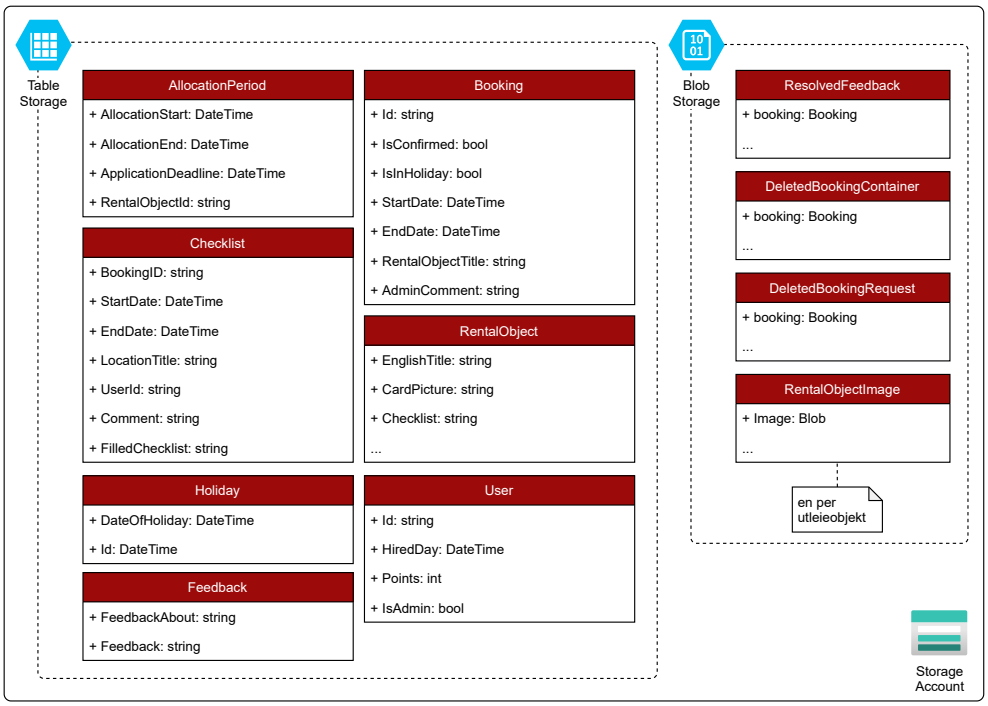
Kommunikasjonen mellom de ulike klassene er også vist i figur 5.7. Merk at enkelte av koblingene i figuren - slik som mellom ”Repositories” og ”Databasemodeller” - er ment å vise hvilke objekter som benyttes, de forsøker ikke å vise at komponentene gjør en direkte handling med et objekt

Bookinger langt frem i tid

En problemstilling som dukket opp var muligheten til å legge til en booking langt frem i tid, og hvordan dette skulle håndteres med tanke på tildelingsperioder og automatisert tildeling. Dette løste vi ved at tildelingsperiodene blir lagt til i databasen for inneværende år og neste år, slik at et utleieobjekt vil hele tiden ha en tildelingsperiode for mer enn ett år frem i tid. Oppdragsgiver godtok dette som en rimelig lengde frem i tid for å kunne legge inn bookinger, så vi la deretter inn sjekker i både klient og API for å ikke tillate bookinger mer enn ett år frem i tid.

5.4.3 Database

I tillegg til personvernsprinsippene som påvirket lagring av data, diskutert i kapittel 5.2, så vi på ulike problemstillinger innenfor datalagring gjennom blant annet responsivitet og lagringskapasitet.



Figur 5.8: Largringen fordeles på Table Storage og Containere som holder blob-er

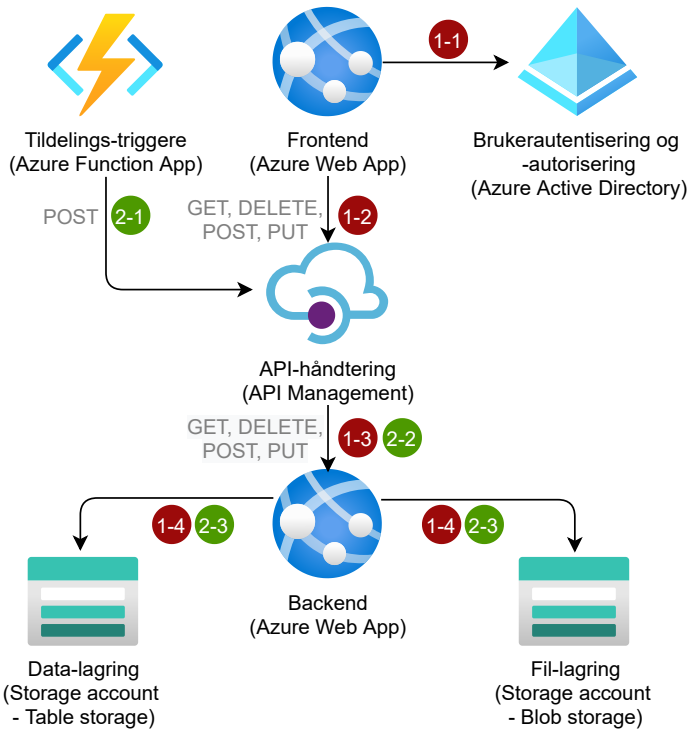
Databasedesignet ble satt opp slik at data som skal aksesseres hovedsaklig ble lagt i Table Storage, mens bildefiler lagres i Blob Storage. Underveis kom også oppdragsgiver med ønske om at både gamle og ubrukte bookinger skulle lagres slik at det kunne brukes til statistikk. Vi gjorde da en vurdering på hvordan vi burde håndtere større datamengder, og kom etter diskusjon med oppdragsgiver frem til at dataene som lagres for statistikk også blir lagt over i

Blob Storage i egne beholdere. En oversikt over det endelige databasedesignet kan sees i figur 5.8.

Årsaken til dette valget er at objekter som lagres i Table Storage er lettere å håndtere som komplekse objekter, mens det er enklere å håndtere data som ikke aktivt skal brukes i Blob Storages fordi det ikke spiller noen rolle om modellene endrer seg over tid - dataene lagres som filer uansett - en oversikt over oppsettet vårt kan sees i figur 5.8. Samtidig unngår vi at queries mot databasen må lete gjennom alt for mange entries i databasen for å finne relevante bookinger, og det var lite ressurskrevende å implementere. For å oppfylle personvernsretninslinjer blir informasjon om brukeren tilhørende bookingen fjernet.

5.4.4 Design i Azure

For å gi et helhetlig overblikk over hvordan de ulike Azure komponentene (beskrevet i 4.1.1) kommuniserer, har vi valgt å ta med en overordnet modell som viser nettopp dette designet i figur 5.9. Her er begge hovedflytene våre vist. Flyten fra Function Appen som trigges automatisk er markert i grønt med "2-x", mens flyten som initieres av en bruker er markert i rødt med "1-x". En detaljert beskrivelse av flyten finnes i vedlegg J.2.



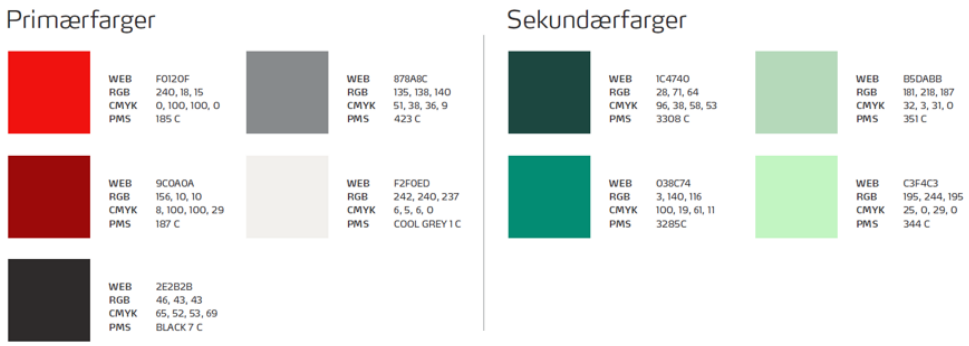
Figur 5.9: Design i Azure

Figuren viser også hvordan APIM, som er konfigurert til kun å tillate kall fra Function Appen og frontend-applikasjonen, kontrollerer alle kall til APIet.

5.5 Brukergrensesnitt

5.5.1 Fargevalg

I utgangspunktet la oppdragsgiver ingen begrensninger på fargevalg, men for å skape sammenheng med resten av selskapets tjenester, tok vi en avgjørelse der det ble bestemt at vi skulle benytte de samme fargene som Communicate benytter på sin egen hjemmeside. Dette innebærer en kombinasjon av mørke gråtoner i kombinasjon med hvitt og rødt, som er bedriftens egne farger. Fargene ble hentet fra selskapets stilguide, som figur 5.10 er et utsnitt av.



Figur 5.10: Oversikt av farger fra Communicates stilguide.

Sekundærfargene ble ikke benyttet, selv om vi hadde behov for en grønnfarge. Etter utprøving med Communicates sekundærfarger opplevde vi at ingen av disse fargene hadde en bekreft/success-nyanse fordi innholdet av blått var for høyt. En god bekreft/sukkess-nyanse var noe vi hadde behov for til en del knapper. Derfor valgte vi en ekstra grønnfarge med hex-kode **#006800** - en ren grønnfarge med god kontrast mot hvit.

5.5.2 Sidedesign

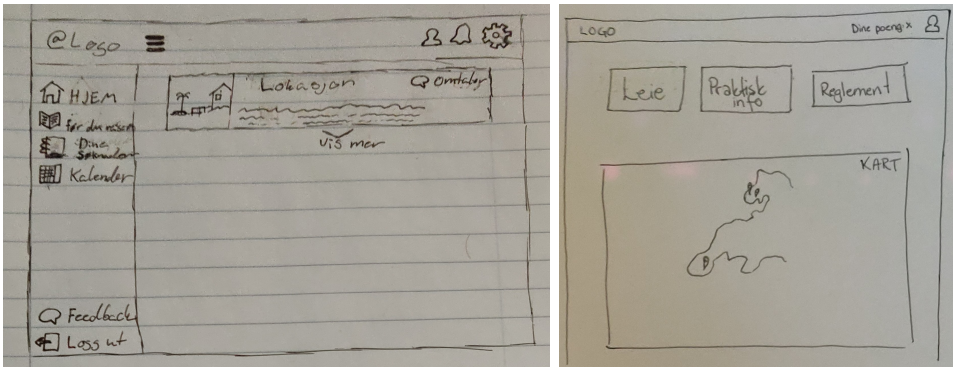
Design av brukergrensesnitt ble bestemt i fellesskap etter at hvert av gruppemedlemmene hadde laget ulike forslag. Inspirasjon ble hentet fra ulike løsninger som gruppen allerede vurderer som velfungerende, deriblant AirBnB og diverse reisebyråer som alle benytter seg av ulike bookingløsninger.

Felles for hele web-applikasjonen er at gruppen ønsket et helhetlig design som ligger tett opp mot designet oppdragsgiver allerede benytter på sine hjemmesider. Dette ble også diskutert med oppdragsgiver i påfølgende møte. I tillegg vil muligheten for å legge til/fjerne utleieobjekter vektlegges slik at en utvikler ikke skal måtte involveres i denne prosessen.

Forside

På forsiden har vi vektlagt at utleieobjektene skal være lett tilgjengelig for brukeren både visuelt og interaktivt. Figur 5.11 viser to ulike iterasjoner fra designprosessen.

Den første iterasjonen (5.11a) har en meny funksjon til venstre som åpnes via en hamburgermeny ved siden av logoen. Menyen inneholder snarveier til systemets viktigste funksjoner. Samtidig får man et viseult overblikk over de ulike utleieobjektene, da det er ment at disse listes nedover som sidens hovedobjekter. Iterasjonen viser kun ett objekt, her er det viktig å understreke at i en slik løsning ville alle objektene ligget nedover som figuren viser. Øverst til høyre ligger ikoner for innstillinger, varsler og brukerprofil.

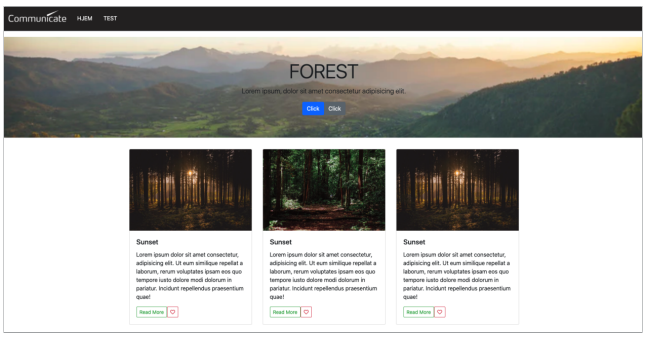


(a) Forsideskisse 1 **(b)** Forsideskisse 2

Figur 5.11: Eksempler på skisser av opprinnelige forslag til design forside

Den andre iterasjonen (5.11b) har et større fokus på informasjonen som er felles for de ulike objektene. Hovedobjektet på denne siden er et kart med nåler som viser lokasjonen til de ulike utleieobjektene. I tillegg ligger det et profilikon i menyen øverst til høyre, samt en visning for opptjente poeng i headeren. Her er hensikten at profilikonet har en dropdown-meny som lar en bruker se egne søknader/bookinger, se varsler, gi feedback og logge ut.

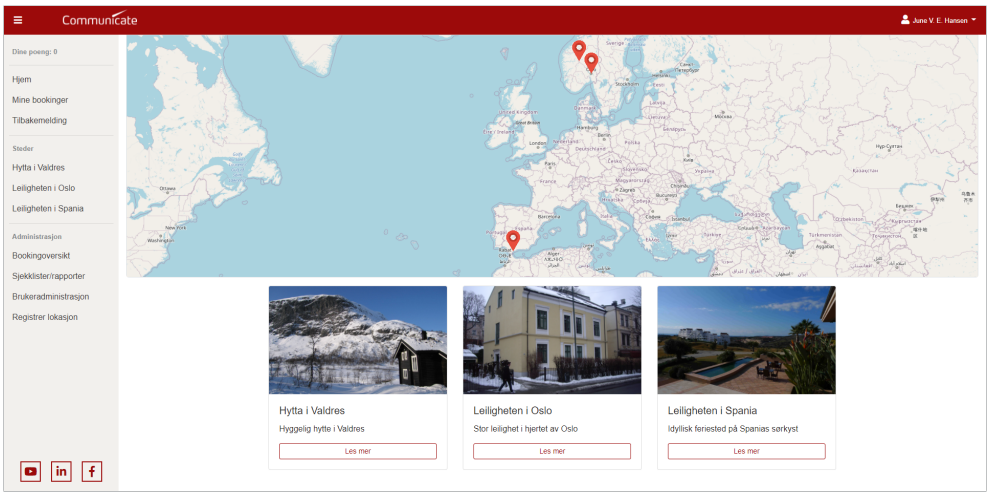
Vi lagde en tidlig versjon av forsiden som var en kombinasjon av disse to ideene. Denne kan sees i figur 5.12. Her er ideen endret noe, til å ha et større forsideelement med elementer for de ulike utleieobjektene under. Elementene for utleieobjekter er en implementasjon av cards, som er et svært vanlig og populært designmønster til bruk som



Figur 5.12: Førsteutkast forside

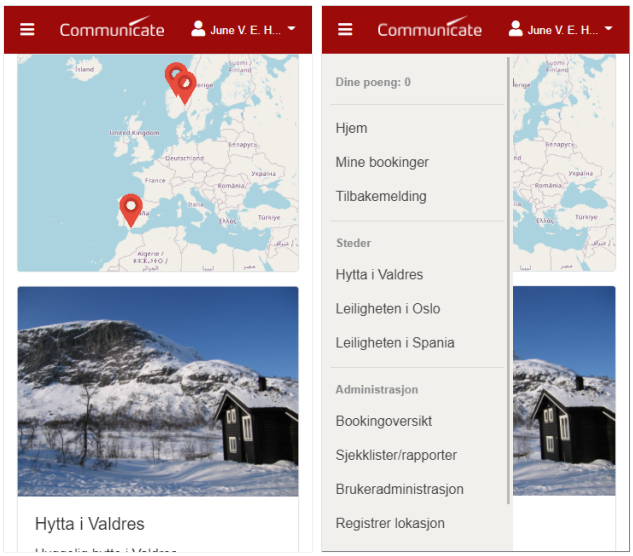
inngangspunkt til mer detaljert informasjon (Material Design 2017*a*).

Til det endelige designet (figur 5.13) ble cards-ene beholdt for tilgang til hvert utleieobjekt, men bildebanneret ble byttet ut med et interaktivt kart slik som tenkt i forsideskisse 2 (figur 5.11b). Vi byttet også ut den mørke fargen i headeren med Communicates rødfarge, siden dette er primærfargen deres (se seksjon 5.5.1 ang. fargevalg). Etter ønske fra Communicate ble det også lagt til en navigasjonsskuff, også et svært vanlig designmønster for mer komplekse navigasjonsmuligheter (Material Design 2017*b*). Dette ønsket kom som resultat av at et av deres egne interne prosjekter - som pågikk parallelt med vårt - endte opp med å benytte dette, og de ønsket et helhetlig design mellom prosjektene.



Figur 5.13: Endelig forside - visning på stor skjerm

Bruk av mobil til å besøke ulike nettsteder har blitt svært utbredt. Selv om det kun var utfylling av sjekkliste som var krav om i forbindelse med mobilvennlighet, valgte vi å gå inn for å lage et design som var responsivt nok til å gi en god brukeropplevelse også på mobil. Hvordan dette ser ut med navigasjonsskuffen åpen og lukket kan sees i figur 5.14. Hovedforskjellen mellom de to sidene ligger i hovedsak i at mo-



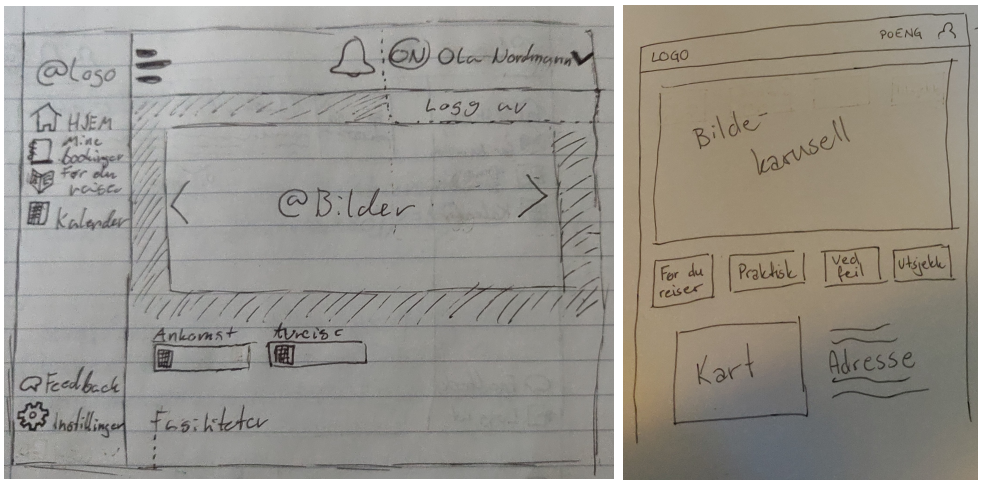
(a) Lukket navigasjonsskuff **(b)** Åpen navigasjonsskuff

Figur 5.14: Endelig forside - mobilvisning

bilsiden har navigasjonsskuffen lukket når man åpner nettsiden. PC-versjonen har derimot denne lukket som standard, med bakgrunn i at det her er mer horisontal plass.

Utleiesider

Tilsvarende ble det laget flere iterasjoner for informasjonssiden til utleieobjektene (to forslag er vist i figur 5.15).



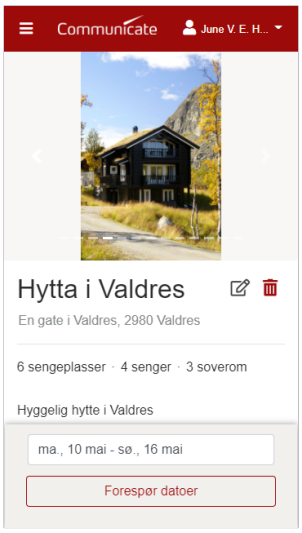
(a) Utleieobjektskisse 1 **(b)** Utleieobjektskisse 2

Figur 5.15: Eksempler på skisser av opprinnelige forslag til design av utleieobjekt

Felles for alle forslagene som ble laget er et sentralt bildeobjekt øverst på siden. I det øverste forslaget er det lagt inn en kalender-funksjon for å sende søknad om leie av utleieobjekt. Under følger det mest essensielle av informasjon om hvert enkelt utleieobjekt. Eksempler på dette er informasjon om sengeplasser, regler for opphold osv. For forklaring av resterende komponenter se første iterasjon i figur 5.11.

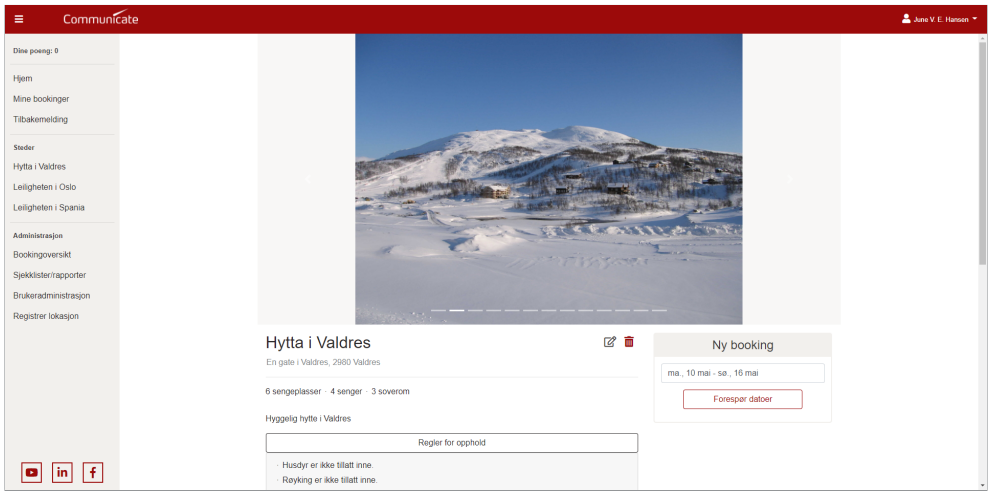
Den andre iterasjonen har fokus på ryddig fremstilling av informasjon. Her vil all informasjon tilknyttet hver enkelt kategori være samlet på egen informasjonsside. I tillegg var det tenkt et zoomet kart som viser lokal plassering som linker mot Google Maps, slik at brukeren raskt kan finne frem til rett adresse.

Figur 5.17 viser vårt endelige design for utleieobjektsiden. Bildekarusellen ble værende, inspirert av tjenester som AirBnB som har en lignende karusell på starten. Deretter har vi litt kort sentral informa-



Figur 5.16: Endelig side for utleieobjekt - mobilvisning

sjon om sengeplasser, adresse osv., samt en datovelger for valg av datoer man ønsker å booke lignende den i skisse 1 (figur 5.15a) - denne deaktiverer dager som allerede er booket.



Figur 5.17: Endelig side for utleieobjekt - visning på stor skjerm

Videre ble knappene side-om-side fra skisse 2 (figur 5.15b) byttet ut med knapper med sammentrekkbart innhold (collapsables), da dette gjorde at siden også fungerte fint på mobil. Disse knappene har stort sett samme innhold som i skissen: regler for opphold, utleieinformasjon som inneholder blant annet tildelingsperioder, inn- og utsjekkstid, leiepris osv., oversikt over utstyr som er tilgjengelig på stedet, og hvorvidt den besøkende må huske på noe før avreise - f.eks. å bestille måking til hytta eller å ha med seg noe. På bunnen av siden er kartet fra skisse 2 tatt med, samt link til Google Maps for enklere planlegging av reisen.

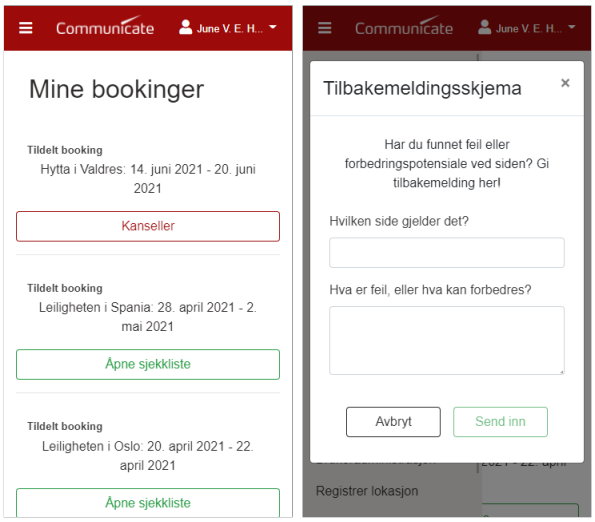
På mobil (vist i figur 5.16) er utseendet stort sett identisk, med unntak av at alt innhold kommer i en kolonne, og dato-velgeren er festet til bunnen av skjermen for lett tilgjengelighet på mobil og kortest mulig å måtte flytte fingerene.

Andre sider

I tillegg til de to overnevnte hovedsidene har alle brukere også tilgang på følgende sider:

- Oversikt over egne bookinger (menyvalg "Mine bookinger") - vist på mobil i figur 5.18a.
- Mulighet for å gi tilbakemelding (menyvalg Tilbakemelding") - vist på mobil i figur 5.18b

Under menyvalget "Mine bookinger" får brukeren opp en oversikt over sine egne bookinger (evt. en melding om at hen ikke har noen bookinger) - både bekreftede og de som fortsatt venter på tildeling. Her kan de kanselleres dersom bookingen enda ikke har startet, eller sjekklisten kan fylles ut etter at oppholdet har startet (og kan sendes inn og oppdateres senere). Sjekklisten er en oversikt over oppgaver leietakerne må gjennomføre før de forlater utleieenheten, slik som å tømme kjøleskap, sjekke at brannalarmer fungerer el.



(a) Oversikt over egne bookinger **(b)** Mulighet for å gi tilbakemelding

Figur 5.18: Diverse andre sider

Under menyvalget "Tilbakemelding" kan brukeren sende inn en tilbakemelding på siden, dette lagres i databasen. Dette var en funksjonalitet det var ønske om fra oppdragsgiver, da videre utvikling og vedlikehold sannsynligvis vil rullere mellom ulike utviklere - og det på denne måten er mulig for en utvikler som tar opp prosjektet å se hva som ønskes forbedret.

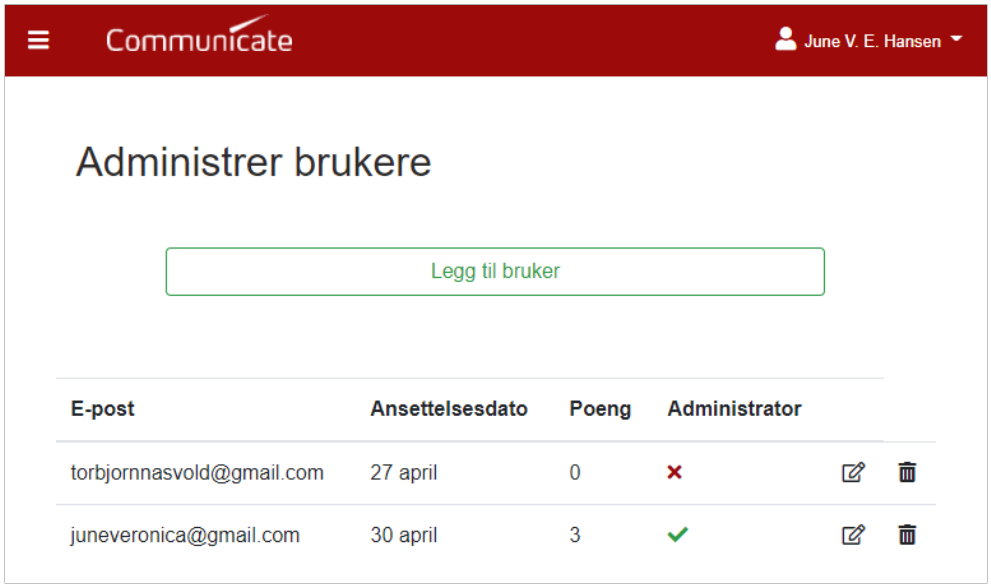
Adminer har også tilgang til tillegsfunksjonalitet utover det vanlige brukere har:

- Oversikt over alle bookinger (menyvalg "Bookingoversikt")
- Oversikt over alle innsendte sjekklister (menyvalg "Sjekklister/rapporter")
- Oversikt over alle brukere og deres poeng (menyvalg "Brukeradministrasjon")
- Mulighet for å registrere nye lokasjoner (menyvalg "Registrer lokasjon")

Bookingoversikten er mye lik en enkeltbrukers oversikt over egne bookinger inkludert muligheten til å kansellere bookinger frem i tid, men inneholder i tillegg en kalender som viser alle bekreftede bookinger for alle utleieobjekter. Sjekklister er ikke i bookingoversikten, men er plassert i en eget menyvalg for bedre oversikt.

Brukeradministrasjonssiden, vist i figur 5.19, lar en admin se alle brukere identifisert på epost, årsdag for ansettelse - slik at poeng kan økes automatisk, hvor mange poeng de har nå, og om de er adminer. Denne brukeradministrasjonen er nødvendig funksjonalitet for at feil skal kunne rettes opp, eller nye brukere legges til. Når nye brukere legges til gjøres også dette med epost som

identifikasjon.



Figur 5.19: Brukeradministrasjon

Til slutt kan adminer også legge til nye - eller endre eksisterende - utleieobjekter. Nye legges til ved å gå inn i et skjema for utfylling under Registrer lokasjon". Det er det samme skjemaet som benyttes for å endre et utleieobjekt, når det brukes for å endre fylles eksisterende informasjon ut i skjemaet. Redigering kan gjøres ved å trykke på blyant-ikonet i hvert utleieobjekt, vist i figurer 5.17 og 5.16.

5.5.3 Annen funksjonalitet

I tillegg til funksjonaliteten beskrevet som en del av kapittel 5.5.2, er også følgende funksjonalitet tilgjengelig:

Ny booking

For at brukeren skal legge til en ny booking må det velges et utleieobjekt på siden, for så å legge til ønsket dato. Om valgt dato er tilgjengelig, lagres den i databasen, ellers vil bruker få tilbakemelding om at tidsperioden allerede er booket.

Legge til og endre utleieobjekt

En administrerende bruker skal ha mulighet til å legge til et nytt utleie objekt, dette gjøres ved å fylle ut et skjema som tar imot alle verdiene og bildene som skal vises på siden til et utleieobjekt. For å endre et utleieobjekt må bruker

først gå inn på den som skal endres, og trykker deretter på et tydelig ikon som tar brukeren til samme side som brukes til å legge til nytt utleieobjekt, her skal all informasjon være ferdig utfylt og skal lett kunne endres på.

Kommentere booking

Admin skal kunne legge inn en kommentar på alle bookinger som er satt, dette er for at admin skal kunne notere seg ulike ting spesifikt for den bookingen. Vanlige brukere har ikke tilgang til disse kommentarene.

Fylle og kommentere sjekkliste

En leietaker skal ved slutten av oppholdet fylle ut en sjekklisten. Den har faste punkter tilhørende utleieobjektet, og det kan i tillegg legges inn en kommentar dersom brukeren trenger å informere admin om noe annet.

5.5.4 Retningslinjer for tilgjengelighet på nettinnhold

Retningslinjer for tilgjengelighet på nettinnhold (Web Content Accessibility Guidelines, forkortet WCAG) er retningslinjer for å gjøre nettsider mer tilgjengelig for personer med ulike funksjonshemminger, samtidig som det gjør nettet lettere å navigere for alle. WCAG har fire hovedprinsipper som må vurderes for at en nettside skal være tilgjengelig for alle - oppfattbar (Perceivable), brukbarhet (Operable), forståelig (Understandable) og robust (MDN Web Docs 2021). Under følger en oversikt over hva vi har gjort for å forsøke å oppnå de ulike prinsippene

Oppfattbar (Perceivable)

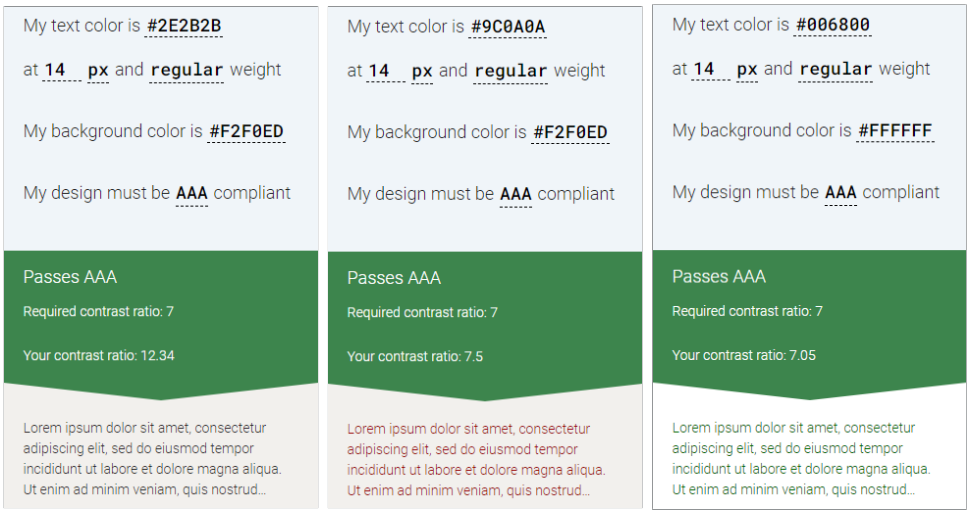
Oppfattbarhet handler om at brukerne av siden skal kunne oppfatte siden med en eller flere av sansene. Relevant for vår oppgave har WCAG her retningslinjer knyttet til bruk av riktige HTML-elementer (f.eks. bruk av `<p>` til tekstavsnitt, `<` ↪ `img>` til bilder og lignende), at siden skal kunne brukes i både portrett og landskapsmodus, at det ikke er unødvendig horisontal scrolling, at farger ikke er den eneste indikasjonen på hva som er "riktig" handling (f.eks. bekreft/avbryt) og at kontrasten er høy nok.

Vi bruker gjennomgående riktige HTML-tags, og fordi vi har laget siden med responsivt design er det ingen problem å bruke siden begge veier. Vi belager oss aldri på kun farge for handlinger, og det er kun ved mobilvisning av brukeradministrasjon det er horisontal scrolling - da for å fortsatt kunne vise tabellen med brukere på en sammenhengende måte.

Når det gjelder kontrast gjorde vi litt mer arbeid for å passe på at vi var innenfor retningslinjene. I WCAG 2.1 kreves det en minimumskontrast (AA-score) med en ratio på 4,5 : 1 mellom forgrunns- og bakgrunnselementer,og for å få en forbedret kontrast (AAA-score) krever at ratioen skal være minst 7 : 1.

Disse kravene gjelder for små elementer. Vi har valgt å gjøre kontrastsjekkene våre som om alt var små elementer, fordi det er de strengeste kravene.

Før vi gjennomførte kontrastsjekker på siden vår hadde vi både elementer med 3.05 : 1 (mellomgrå mot lys grå) og 3.54 : 1 (lys grønn mot hvit) i kontrast. Disse elementene fikk alle mørkere forgrunnsfarger, slik at de kunne oppfylle kravene. I figur 5.20 er en oversikt over de svakeste kontrastene siden vår nå har, og derfor de som får lavest score. Som figuren viser oppfyller nettsiden kravene for en AAA-score etter at vi gjorde endringene for å øke kontraster.



(a) Mellomgrå og lys grå **(b)** Rød og lys grå **(c)** Grønn og hvit

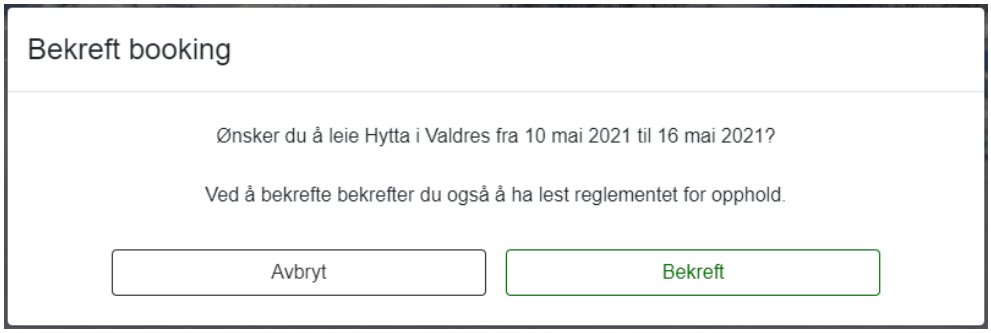
Figur 5.20: Kontrastsjekk

Brukbarhet (Operable)

Brukbarhet handler om at funksjonalitet skal være tilgjengelig for alle uavhengig av funksjonsnedsettelse. Relevant for vår oppgave har WCAG her retningslinjer knyttet til at det ikke skal være blinkende animasjoner, at det skal være lett å navigere, og at knapper skal være store nok.

Her kom mye naturlig for oss under utvikling. Vi har kun et fåtall animasjoner, og de er knyttet til elementer som glir inn og et ikon som viser at siden laster. Vi passet også på å konsekvent benytte minst standardstørrelsen på knapper (har mulighet for små, medium og stor - medium er standard) hvor vi benyttet Bootstrap-styling, og gjorde generelt en innsats på at valgmuligheter skulle være store nok og enkle å bruke.

Et fokus vi hadde som gjelder både brukbarhet og oppfattbarhet, er å gjøre brukerinput og handlinger intuitivt. Mange av brukerens interaksjoner ved nettsiden skjer gjennom bekreftelsesdialoger. Et eksempel på en slik dialog kan sees i figur 5.21, og her er designet representativt for alle slike bekreftelsesdialoger.



Figur 5.21: Bekreftelsesmodal for booking

For designet på disse bekreftelsesdialogene har vi tatt utgangspunkt i Material Design sine anbefalinger (Ravichandran 2021):

- knapper i de aller fleste tilfeller har en tekst, og belager seg ikke kun på et ikon.
- når det er to knapper i en boks burde de ligge side ved side, ikke over hverandre.
- de viktigste handlingene vektlegges enten med omriss og/eller farge.
- gråtoner benyttes for handlinger med mindre vekt.
- nok avstand mellom to knapper (minst 16dp).

Fordi nettsiden gjennomgående bruker bekreftelsesdialoger når brukeren skal fullføre en handling, er det særlig her det har vært fokus på det overnevnte. Figur 5.21 viser hvordan "Bekreft"-knappen har både grønt omriss og tekst, mens avbryt er har grått omriss og tekst. I bekreftelsesdialoger hvor man gjør en negativ eller "farlig" handling - for eksempel ved kansellering av bookinger - er "Bekreft"-knappen rød. Knappene er også av god størrelse, og har mer enn 16dp mellomrom. De benytter gjennomgående tekst og ikke ikoner, slik at ikke fargeblinde skal ha problemer selv om også farge brukes for å indikere bruken.

Forståelig (Understandable)

Forståelighet handler om at språket som benyttes skal være forståelig, at siden oppfører seg forutsigbart, og at brukeren skal hjelpes for å identifisere feil. Her har vi forsøkt å gjennomgående designe sidene forholdsvis like, hvor alle bekreftelsesdialoger har samme design, alle knapper har samme utseende, og alle sider laster på samme måte med likt ikon for å indikere at siden laster inn. For feilidentifisering har vi både beskrivende feilmeldinger dersom bookinger feiler (f.eks. informasjon om at en booking feiler fordi den overlapper med en egen eksisterende booking), og vi har fokusert på gode og beskrivende tooltips og feilmeldinger i skjemaet som fylles ut ved opprettelse eller endring av et utleieobjekt.

Robusthet (Robust)

Robusthet handler om at løsningen skal ha kompatibilitet med nåværende og senere standarder, dette kan gjøres gjennom velformet og valid HTML, definering av roller, statusmeldinger og lignende i HTML. I vårt tilfelle har vi gjennomgående passet på at HTMLen vår er velformet, samt at de få stedene vi bruker HTML-elementer som ikke matcher helt med hva de gjør (f.eks. `<a>`-tag (link) som i praksis er en knapp), har fått rolle-property som matcher det de fungerer som i praksis. I tillegg har vi properties for status på ikoner som vises når siden lastes.

5.6 Sikkerhet

Til tross for at systemet kun skal brukes internt hos oppdragsgiver ønsket vi også å få erfaring med å skape et system som er hensiktsmessig robust sikkerhetsmessig. Derfor tok vi utgangspunkt i OWASP Top 10 (OWASP 2020) for å sikre oss mot de vanligste truslene mot web-applikasjoner.

Ved undersøkelse av god praksis for implementasjon av sikkerhet i Angular, viste det seg at rammeverket allerede har mye innebygd funksjonalitet for å hindre flere av de vanligste truslene - blant annet for å stoppe forsøk på Cross-Site-Scripting, Cross-Site Request Forgery og Cross-Site Script Inclusion, samt automatisk sanitering av HTML, og styling (SCSS i vårt tilfelle), samt URLer i properties (Angular 2021*b*).

Vi har også ekstra beskyttelse mot Cross-Site Request Forgery ved å benytte Multi Step Transactions ved å gjøre alle kall mot backend via APIM (se kapittel 4.1.1), som gjør det vanskelig for en ekstern aktør å følge forespørselen.

Vi unngår også problematikk rundt implementasjon av usikker innloggingsfunksjonalitet ved å gjøre innlogging via Microsofts MSAL-bibliotek. Her henter vi tilgangsnøkler til Microsoft Identity Plattform - Microsofts .NET-rammeverk for innlogging med Microsoft-kontoer. Identity Platform benytter industristandardene Open ID Connect og OAuth 2.0 for henholdsvis autentisering og autorisering (OAuth 2.0 2021, Microsoft 2020*d*).

Problematikk rundt SQL-injections unngår vi ved å ikke bruke SQL-kall ved å ikke benytte en SQL-database, men heller Table Storage med LINQ-spørringer (Microsoft 2017*b*). For å minimere muligheten for sikkerhetsbrister i avhengigheter har vi i frontend jevnlig brukt `npm audit` kommandoen tilhørende Node Package Manager som automatisk oppdaterer avhengigheter med kjente feil (npm Docs 2021*a*), og har i backend gjennomgått alle avhengigheter og oppdatert disse til siste versjon kort tid før innlevering - mer om dette i kapittel 7.2.

5.6.1 Testing

Hovedformålet med testing er å kvalitetssikre at systemets møter de krav som er definert i prosjektets oppstart (jmfr. kravspesifisering), og samtidig fange

opp eventuelle feil.

Det finnes flere ulike metoder for å gjennomføre tester innen systemutvikling. Gruppen anså to vanlige tilnærminger for testing som aktuelle for vårt prosjekt, - testing under utvikling og testbasert utvikling. Testing under utvikling er en tilnærming der gruppen som utvikler tester systemet underveis i utviklingen. Dette innebærer som oftest å skrive funksjonalitet, for deretter å teste denne funksjonaliteten (Sommerville 2004). Testbasert utvikling er en form for inkrementell utvikling som går ut på at testene skrives sammen med funksjonaliteten som skal implementeres (Sommerville 2004). Som en konsekvens av at gruppen på generell basis hadde lite erfaring knyttet til testing ble sistnevnte metode valgt bort, da dette krevde oppsett av et testrammeverk allerede fra start.

En vanlig rekkefølge i denne formen for testing er (Sommerville 2004):

1. Test av enheter (unit testing)
2. Test av komponenter (component testing)
3. Test av systemet

I tillegg ble det bestemt at gruppen skulle følge Microsofts ”best practices” for testing. Her ble det lagt særlig fokus på struktur etter arrange - act - assert. I tillegg hadde vi fokus på at testene skulle navngis etter funksjonsnavn, hva som ble testet, og forventet resultat. Testutfallene skulle valideres gjennom utprøving av kjente feilverdier, og på denne måten sikre at både funksjonen som blir testet og testen i seg selv egner seg til sitt formål (Microsoft 2018).

5.7 Øvrige designvalg

Applikasjonen sender epostvarslinger til brukerne. Innholdet av epostene har vi valgt å formattere i HTML. Årsaken til det er at vi erfaringsmessig vet at dette gir flere muligheter til formatering, som igjen kan gi et mer leselig sluttresultat sammenlignet med ren tekst.

Kapittel 6

Implementasjon

Bruken og funksjonaliteten til systemet er i stor grad beskrevet i de tidligere kapitlene. I dette kapitlet kommer informasjon om utvalgte implementasjoner som er gjort som en del av utviklingen av applikasjonen.

6.1 Frontend

6.1.1 HTTP service

Frontend inneholder veldig lite egen informasjon, og det aller meste av innholdet hentes fra backend. På grunn av dette ønsket vi å lage en generisk service for å håndtere HTTP-kall, slik at de blir gjort likt overalt. I denne servicen gjøres feilhåndtering, alle nødvendige headere blir satt osv. Når denne servicen brukes trengs det kun spesifisere URL-en kallet skal sendes til og hvilken data som eventuelt skal sendes med. Dette kan sees i kodeliste 6.1.

```
1  export class httpService {
2      constructor (private http: HttpClient) { }
3
4      httpOptions = {
5          headers: new HttpHeaders({
6              'Ocp-apim-subscription-key': environment.apimKey,
7              'Content-Type': 'application/json'
8          }),
9          withCredentials: true
10     }
11
12     // Get all if no parameters, otherwise can be used as ie. getById
13     public get<T>(path: string, parameters?: string): Observable<T> {
14         let fullPath = path;
15         if (parameters) {
16             fullPath += parameters;
17         }
18         return this.http
19         .get<T>(`${environment.apiUrl}${fullPath}`, this.httpOptions)
20         .pipe(
21             retry(2),
22             catchError(this.handleError)
23         )
24     }
25
26     public post<T>(path: string, body: any): Observable<T> {
27         return this.http
28         .post<T>(`${environment.apiUrl}+${path}`, JSON.stringify(body), this.
               ↪ httpOptions) // + options?
29         .pipe(
```

```
30              retry(2),
31              catchError(this.handleError)
32          )
33      }
34
35      // ... Similar functions for update, and delete
36
37      private handleError(error: HttpErrorResponse) {
38          let errorMessage = '';
39          if (error.error instanceof ErrorEvent) {
40              errorMessage = error.error.message;
41          } else {
42              errorMessage = `Error code: ${error.status}\nMessage: ${error.message}`
43          }
44          console.log(errorMessage);
45          return throwError(errorMessage);
46      }
47  }
```

Kodeliste 6.1: Håndtering av HTTP-kall i http.service.ts

6.1.2 Logg inn

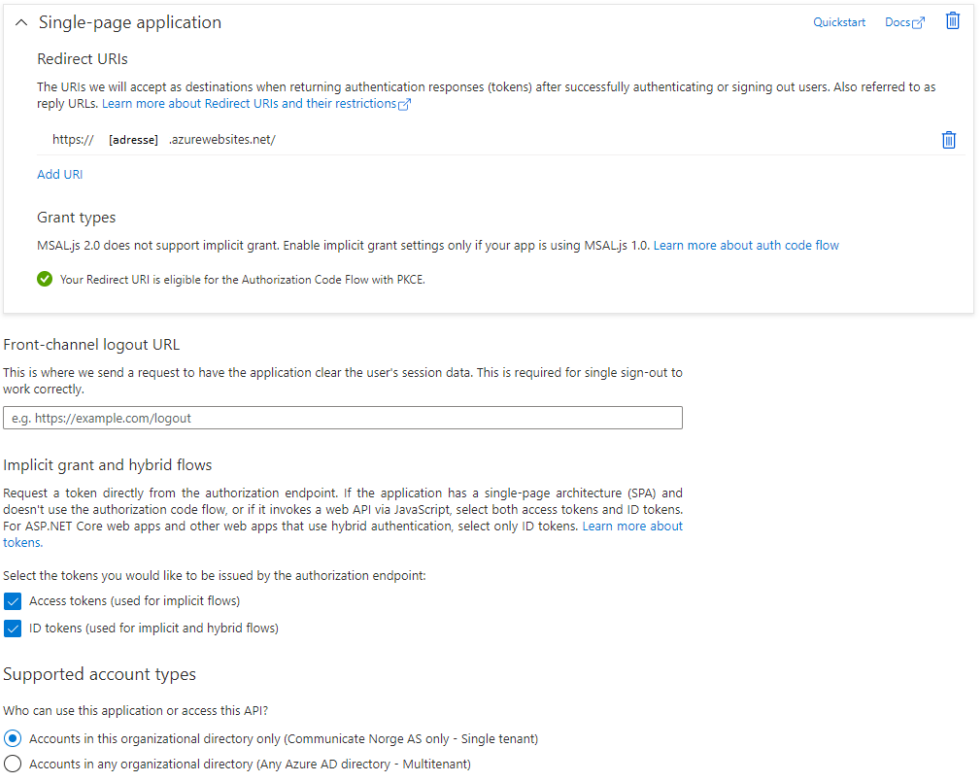
Et vanlig prinsipp for webapplikasjoner som benytter innloggingstjenester er å utsette denne prosessen så lenge som mulig, hovedformålet med dette er at brukeren skal bli kjent med applikasjonen før man trenger å registrere seg (Microsoft 2021*k*). Fordi dette er et system som kun skal brukes internt hos oppdragsgiver, og det er ønskelig at informasjon om utleieobjektene ikke ligger åpent tilgjengelig med tanke på materiell sikkerhet, har vi valgt å ikke følge dette prinsippet.

Fordi alle ansatte i Communicate allerede har en Microsoft-bruker, kom vi kjapt til enighet med oppdragsgiver om å bruke disse kontoene for innlogging også i dette systemet. Det gjør det enklere for de ansatte å bruke siden ettersom de slipper å huske påloggingsinformasjonen til enda en brukerkonto, og det gjør at vi kan benytte Microsofts bibliotek for innlogging, noe som gjør at vi unngår å lage en egen implementasjon til sikkerhetsstandarder som OAuth eller lignende.

Microsofts bibliotek for innlogging - MSAL (Microsoft Authentication Library) - benytter Azure AD som beskrevet i kapittel 4.1.1. Ved hjelp av deres mal for oppsett for Angular - MSAL Angular Sample Application (Microsoft 2020*g*) - er dette forholdsvis greit å sette opp. Noe konfigurasjon er likevel nødvendig i Web App-en for frontend i Azure, dette kan sees i figur 6.1.

6.1.3 Brukerinput

All bruker inntasting skjer gjennom Reactive Form som gir oss mulighet til å validere brukerinput mens brukeren skriver. Dataene legges inn i grupper og deaktiverer knappene for videre operasjoner helt til all input er gyldig (Angular 2021*a*). Her bruker vi egendefinerte feilmeldinger og gir beskjed til brukeren om



Figur 6.1: Konfigurasjon for login i Azure

noe har en ugyldig verdi - eller om et felt mangler. En vanlig bruker kommer nok ikke til å se dette særlig ofte, men det er en viktig del av det å gjøre systemet brukervennlig. I tillegg kan det være særlig nyttig for adminer når de skal registrere nye utleieobjekter eller registrere nye brukere. Disse stedene kan det lett oppstå inputfeil, og siden særlig å legge inn/endre et utleieobjektet er en såpass omfattende prosess, er det nødvendig med gode tilbakemeldinger.

6.1.4 Routing og komponenter

I Angular rammeverket bygges applikasjonen opp ved bruk av komponenter, disse komponentene gjør det enkelt å holde kontroll på de ulike sidene som skal bli rendret og gir en veldig solid struktur. Alle komponentene som skal brukes blir rendret inn i `<app-root></app-root>` i index.html filen. Hvilke komponenter som skal brukes blir bestemt av bestemte endepunkter, dette blir referert som routing (Acondy 2020).

Som nevnt i kapittel 5.6 bruker vi MSAL-biblioteket, som gir oss mulighet for å bruke MsalGuard (Bauknecht 2020) for å sikre at disse komponentene bare kan bli vist frem for innloggede brukere. Vi kunne også benyttet en egenlaget Guard-funksjon for å sikre at kun adminer har tilgang til admin-siden. Prosjektet slik vi leverer det bruker en vanlig if-sjekk på brukerens isAdmin-

property, men å bytte dette til en Guard-funksjon er lagt i backloggen til videre utvikling på grunn av tid.

Som nevnt i kapittel 5.4.1 bruker vi komponenter vi kaller ”views” som sider som navigeres til, mens andre komponenter kan gjenbrukes mellom de ulike visningene. Disse visningene kan routes til ved hjelp av routing-modulen. Den har et oppsett som vist i kodeliste 6.2, hvor nye endepunkter legges til ved å legge de til i listen. Innholdet i `path`-variablen legges til hovedadressen til nettsiden, slik at for eksempel `locations/:id` vil bli `www.nettide.no/locations/tittel` hvor `tittel` er utleieobjektets id - for eksempel ”Oslo”.

```
const routes: Routes = [
  {
    path: 'locations/new',
    component: RegisterNewRentalObjectComponent,
    canActivate: [MsalGuard],
  },
  {
    path: 'locations/new/:id',
    component: RegisterNewRentalObjectComponent,
    canActivate: [MsalGuard],
  },
  {
    path: 'locations/:id',
    component: RentalObjectComponent,
    canActivate: [MsalGuard]
  },
  {
    path: 'bookings/overview',
    component: AdminBookingOverviewComponent,
    canActivate: [MsalGuard]
  },
// More paths here
];

// Export routing module
@NgModule({
  imports: [
    RouterModule.forRoot(routes}),
  ],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Kodeliste 6.2: Routing definert i app-routing.module.ts

6.1.5 Caching

For å gjøre applikasjonen mer responsiv og dermed unngå unødvendige kall til backend, har vi valgt å cache data som benyttes flere ganger i applikasjonen (Katkov 2019). Det vil si at dataen som hentes blir lagret i en variabel slik at dataen ikke trenger å hentes i databasen hver gang den er forespurt. Cachingen skjer når det første HTTP-kallet er gjort og dataen er dermed tilgjengelig frem til bruker forlater siden. Det er hovedsaklig data tilknyttet utleieobjekter og pålogget bruker som caches. Løsningen kan skape samtidighetsproblematikk

dersom flere brukere er pålogget og et utleieobjekt oppdateres, ved at andre brukere da ikke ser oppdateringen før siden lastes inn på nytt. Etter diskusjon med oppdragsgiver er dette dog en grei løsning, ettersom det senker responstiden ved navigasjon en god del, og det ikke hverken forventes stor trafikk til siden, eller forventes at utleieobjektene oppdateres særlig ofte.

Et eksempel på slik caching kan sees i kodeliste 6.3, hvor en bruker som er hentet lagres i en variabel i servicen. Hver gang `getUser()` kalles vil den sjekke om variablen har innhold, hvis den har det returneres brukerinformasjonen. Hvis ikke hentes nødvendig informasjon fra MS Graph og backend før det returneres. Denne cachen blir naturlig nok tømt når en bruker logger ut.

```
1  export class UserService {
2    userInfo: SingleUser = new SingleUser();
3    private readonly graphEndpoint = 'https://graph.microsoft.com/v1.0/me';
4
5    constructor(private httpService: HttpService) {}
6
7    /**
8     * Get full user object, including info from MSGraph and database.
9     *
10    * @returns Promise<User>
11    */
12   public async getUser() {
13     if (this.userInfo === undefined || this.userInfo.points == null) {
14       await this.getUserDbInfo();
15     }
16     return this.userInfo;
17   }
18
19   // ... Other functions
```

Kodeliste 6.3: Caching av pålogget bruker

6.2 Backend

For å bygge opp APIet valgte vi å bruke en mal for ASP.NET Core applikasjoner. Dette innebærer en programfil som benytter konfigurasjoner i en oppstartsfil for å opprette et lokalt endepunkt for APIet.

6.2.1 Konfigurasjoner

Ettersom vi benytter innloggingsfunksjonalitet i klienten, var det også essensielt å beskytte APIet for kall fra andre tjenester enn vår klient, med tilhørende innlogget bruker. For å løse dette satte vi opp en CORS policy (Cross Origin Requests Policy) - en W3C standard som definerer hvilke URLer APIet tillater å motta kall fra (Rick Anderson 2021), samt hvilke typer kall som er tillatt. Se kodeliste linjer 9-16 i 6.4 for implementasjonen av CORS.

Vi ønsket også at systemet skulle bestå av uavhengige komponenter gjennom å benytte avhengighetsinjeksjon (dependency injection) - en metode for å sikre

uavhengige komponenter. Tre vanlige metoder for å løse dette i .Net Core applikasjoner er gjennom å benytte et av følgende design patterns (Maiti 2021):

- Scoped - Tjenesten blir opprettet en gang for hver forespørsel
- Transient - Tjenesten blir opprettet hver gang de er forespurt (kan skje flere ganger i samme forespørsel)
- Singleton - Tjenesten blir opprettet en gang, og er global for hele applikasjonen

For å unngå å lage unødvendige instanser av samme klasse har vi instansiert disse som singletons under oppstart. Disse singletons blir sendt til hvert sted som trenger de som en parameter i konstrukturen - en teknikk kalt konstruktør-injeksjon (constructor injection). Et eksempel på implementasjonen av dette, for klienten som jobber mot lagring av blobber, vises på linje 21 i kodeliste 6.4.

```
1  ...
2  public void ConfigureServices(IServiceCollection services)
3  {
4      services.AddControllers();
5      services.AddSwaggerGen(c =>
6      {
7          c.SwaggerDoc("v1", new OpenApiInfo { Title = "
               ↪ communicate_rental_backend_dev", Version = "v1" });
8      });
9      services.AddCors(options =>
10     {
11         options.AddPolicy("CorsPolicy",
12         builder => builder.WithOrigins(Configuration.GetValue<string>("CORSpolicy")
               ↪ )
13         .AllowAnyMethod()
14         .AllowAnyHeader()
15         .AllowCredentials());
16     });
17     services.AddAuthentication(IISDefaults.AuthenticationScheme);
18
19     var storageAccountConnectionString = Configuration.GetConnectionString("
           ↪ DefaultConnection");
20
21     services.AddSingleton<IBlobStorageClient>(BlobStorageClientFactory.Create(
           ↪ storageAccountConnectionString));
22     ...
23 }
```

Kodeliste 6.4: Konfigurasjoner definert i Startup.cs

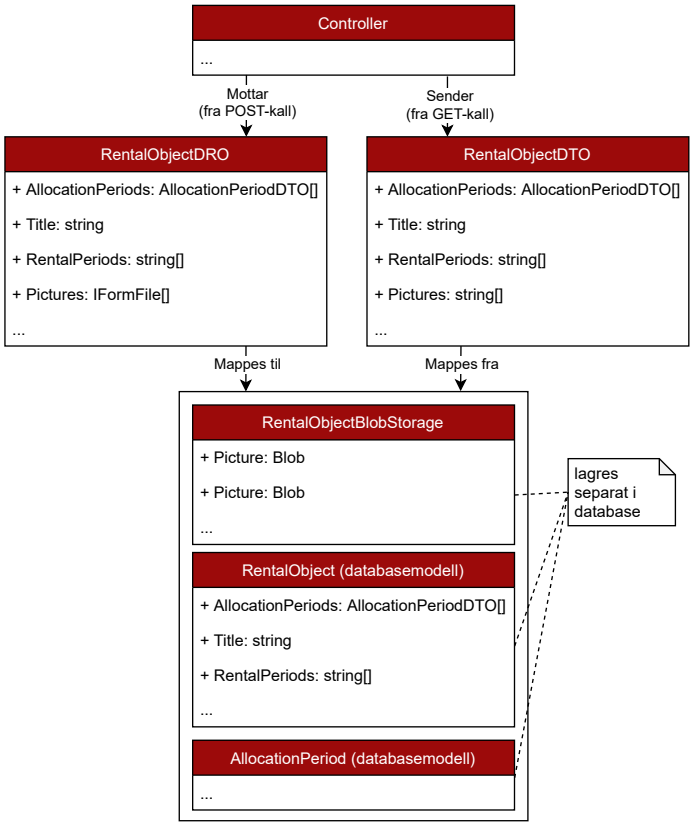
Vi har valgt å legge alle konfigurasjoner hvor verdier ansees å være sensitive i en konfigurasjonsfil som ikke sendes til DevOps repoet - den ligger kun lagret lokalt hos gruppemedlemmene (mer om hvordan disse hentes under deployment i kapittel 8). Hvordan dette er implementert vises gjennom eksempelet på linje 19 i kodeliste 6.4. Vi unngår på denne måten at opplysninger om applikasjonen, databasetilknytning og epost-kontoer ligger åpent i repoet.

6.2.2 Databehandling

Modeller

Objekter som mottas av backend er ikke nødvendigvis like de som sendes ut. For å løse denne problemstillingen implementerte vi tre ulike modeller for hvert objekt: ett "Data Recieving Object" (heretter DRO) for objekter som sendes til APIet; en databasemodell som inneholder objektets datamedlemmer i Table Storage; og ett "Data Transfer Object" (heretter DTO) som inneholder data som blir skrevet fra APIet med GET-kall (der DTO ikke matcher DRO). Et eksempel på et slikt objekt er utleieobjektene. Disse blir lagt til med tilhørende bildefiler i DRO, DTO inneholder kun URLene til aktuelle Blobs (se kapittel 5.3 for utdypende beskrivelse).

Mapping og kontrollere



Figur 6.2: Mappingeksempel

For å løse dette ble det laget egne Mappere for alle modeller som endret datamedlemmer etter hva det var behov for. Når et HTTP POST eller PUT-kall med et DRO kommer inn i kontrolleren tilhørende objektet, mappes objektet til en databasemodell. Dette kan for eksempel innebære å splitte objekter eller

legge til et datamedlem (se eksempel for et utleieobjekt i figur 6.2). Ved GET-kall vil mapperne ha som oppgave å gjøre om eller sette sammen objekter som er identiske med modellen som klienten forventer, for å sikre at informasjonen blir lest korrekt.

Hvert objekt har sin egen kontroller som har som hovedoppgave å styre dataflyten i systemet for det objektet. I tillegg gjøres logiske operasjoner i forbindelse med dette, som for eksempel kan innebære å sammenligne allerede eksisterende data, eller å hente informasjon som kreves i mappingen av objektene - for eksempel fra Microsoft Graph.

```
1  [ApiController]
2  [Route("[controller]")]
3  public class RentalObjectController : Controller
4  {
5      ...
6      [HttpPost]
7      public async Task<IActionResult> Add([FromForm]RentalObjectDRO rentalObjectDRO)
8      {
9          // If there are images, map and upload them
10         if (rentalObjectDRO.Pictures != null && rentalObjectDRO.Pictures.Any())
11         {
12             foreach (var img in rentalObjectDRO.Pictures)
13             {
14                 await imageRepository.UploadAsync(img.OpenReadStream(), Path.
                       ↪ GetFileName(img.FileName), rentalObjectDRO.EnglishTitle.
                       ↪ ToLower());
15             }
16         }
17         await imageRepository.UploadAsync(rentalObjectDRO.CardPicture.
               ↪ OpenReadStream(), Path.GetFileName(rentalObjectDRO.CardPicture.
               ↪ FileName), rentalObjectDRO.EnglishTitle.ToLower());
18
19         var cardPictureUrl = await imageRepository.GetURL(rentalObjectDRO.
               ↪ EnglishTitle.ToLower(), Path.GetFileName(rentalObjectDRO.
               ↪ CardPicture.FileName));
20
21         // If there are AllocationPeriods, map and upload them
22         if (rentalObjectDRO.AllocationPeriods != null && rentalObjectDRO.
               ↪ AllocationPeriods.Any()){
23             foreach (var period in rentalObjectDRO.AllocationPeriods)
24             {
25                 var allocationPeriod = allocationPeriodMapper.Map(period,
                       ↪ rentalObjectDRO.EnglishTitle);
26                 await allocationPeriodRepository.Add(allocationPeriod);
27                 ...
28             }
29         }
30         // Create RentalObject from RentalObjectDRO and upload
31         RentalObject rentalObject = rentalObjectMapper.Map(rentalObjectDRO,
               ↪ cardPictureUrl);
32
33         await rentalObjectRepository.Add(rentalObject);
34         ...
35     }
36     ...
37 }
```

Kodeliste 6.5: Mapping i RentalObjectController.cs

Kodeliste 6.5 viser detaljer rundt hvordan et utleieobjekt legges til i systemet. Utleieobjektet splittes opp gjennom at bildene lagres i en Blob-container. Det samme gjøres med tildelingsperiodene, disse mappes i tillegg til en databasemodell, her legges det til en id og hvilket utleieobjekt perioden tilhører, for å kunne hente dette ut senere (linje 20-25). Til slutt mappes også utleieobjektet, før det lastes opp i databasen, slik figur 6.2 viser.

Repositoryer og klienter

Repositoryene inneholder funksjoner for å hente og skrive data til databasen. Kommunikasjonen med databasen opprettes gjennom en klient av databasemodellen i hvert repository slik kodeliste 6.6 viser.

```
1 public class  RentalObjectRepository : IRentalObjectRepository
2 {
3     private readonly ITableStorageClient<RentalObject> client;
4     ...
5 }
```

Kodeliste 6.6: I repositoryet opprettes en klient for hver databasemodell (Table Entity)

Klienten som blir opprettet i kodeliste 6.6) oppretter så en tilkobling til databasen gjennom nøkkelen (”connectionString”-parameteret på linje 6) slik kodeliste 6.7 viser.

```
1  public class TableStorageClient<T> : ITableStorageClient<T> where T : TableEntity,
       ↪ new() {
2      private readonly CloudStorageAccount storageAccount;
3      private readonly CloudTableClient tableClient;
4      private readonly CloudTable table;
5
6      public TableStorageClient(string connectionString, string tableName) {
7          storageAccount = CreateStorageAccountFromConnectionString(connectionString)
               ↪ ;
8          tableClient = storageAccount.CreateCloudTableClient(new
               ↪ TableClientConfiguration());
9          table = tableClient.GetTableReference(tableName);
10         table.CreateIfNotExists();
11     }
12
13     public async Task<IEnumerable<T>> GetAll() {
14         var entities = new List<T>();
15
16         // ContinuationToken and do/while in case of more than 1000 results
17         TableContinuationToken continuationToken = null;
18         do {
19             var query = new TableQuery<T>();
20             var queryResult =  await table.ExecuteQuerySegmentedAsync<T>(query,
                   ↪ continuationToken);
21
22             continuationToken = queryResult.ContinuationToken;
```

```
23              entities.AddRange(queryResult.Results);
24
25          } while (continuationToken != null);
26
27          Console.WriteLine(entities);
28          return entities;
29      }
30      // More functions for GetById, GetByPredicate, Upsert and Delete
31  }
```

Kodeliste 6.7: Utsnitt av klienten som håndterer kommunikasjon med Table Storage

Denne generiske klienten gjør at koden i repositoryene enkelt kan kalle sin klient, som håndterer detaljene rundt uthenting fra databasen. Siden det er behov for flere repositoryer, unngår vi på denne måten kodeduplisering - gjennom at alle repositoryene benytter den samme generiske klienten til å opprette en egen instans av den samme klienten (ved hjelp av dependency injection). Hvordan den samme instansen benytter sin klient til å kalle databaseinteraksjonen er vist i kodeliste 6.8 eksemplifisert ved et utsnitt av repositoryet for utleieobjekter.

```
1   public class RentalObjectRepository : IRentalObjectRepository {
2       private readonly ITableStorageClient<RentalObject> client;
3
4       public RentalObjectRepository(ITableStorageClient<RentalObject> client) {
5           this.client = client;
6       }
7
8       public async Task<RentalObject> Get(string id) {
9           return await client.Get(id, id);
10      }
11
12      public async Task<IEnumerable<RentalObject>> GetAll()  {
13          return await client.GetAll();
14      }
15
16      public async Task AddRentalObject(RentalObject rentalObject)  {
17          await client.Upsert(rentalObject, rentalObject.EnglishTitle, rentalObject.
                ↪ EnglishTitle);
18      }
19      ...
```

Kodeliste 6.8: Repositoryet som håndterer utleieobjekter

I tillegg til funksjonene vist i eksempelet over er det mulig å hente ut dataobjekter basert på predikater. I vårt system benytter vi lambda uttrykk, for å hente ut objektet basert på en eller flere av dets datamedlemmer. Dette vises i kodeliste 6.9, hvor bookinger som er bekreftet blir hentet.

```
1   public async Task<IEnumerable<Booking>> GetConfirmedBookings()
2   {
3       return await client.GetByPredicate(x => x.IsConfirmed == true);
4   }
```

Kodeliste 6.9: Lambda uttrykk gjør det mulig å hente objekter basert på predikater

6.2.3 Logiske utregninger

Logiske operasjoner som gjør det mulig å automatisere tildelingen av utleieobjekter vil bli gjort i businesslaget. Tildelingen skal som kjent skje på ulike tidpunkter for de ulike utleieobjektene. Andre føringer for designet av businesslaget er at vi har vektlagt at det skal være mulig å legge til utleieobjekter uten å involvere utvikler. Vi løste dette ved å sette opp en Function App i Azure som trigger en controller i applikasjonslaget (se kapittel 5.4.4), funksjonene i denne controlleren kan sees i kodeliste 6.10.

```
1  [HttpPost]
2  [Route("Trigger")]
3  public void Scheduler()
4  {
5      var today = DateTime.Today;
6
7      functionAppService.UpdateUserPoints(today);
8
9      functionAppService.AllocateRentals(today);
10
11     functionAppService.UpdateAllocationPeriods(today);
12
13     functionAppService.MoveBookingsToBlobOrSendChecklistReminder(today);
14
15     functionAppService.SendListOfBookingsForMonth(today);
16
17     // If 1. jan - make sure the holidays for the next year is uploaded to db
18     //  remove holidays for last year
19     if (today.Date == new DateTime(today.Year, 1, 1).ToUniversalTime().Date)
20     {
21         functionAppService.UpdateHolidays(today);
22     }
23 }
```

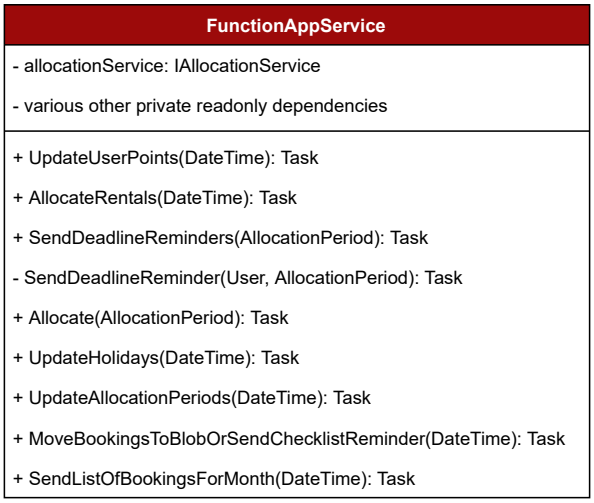
Kodeliste 6.10: Scheduler som håndterer logiske oppgaver i automatiseringsprosessen

6.2.4 FunctionAppService

FunctionAppService er en klasse som tar i mot kall fra FunctionAppControlleren diskutert i avsnittet over. Klassen inneholder ulike funksjoner for oppdatering av databasen og automatisk tildeling av bookinger. En oversikt over eksisterende fuksjoner er vist i figur 6.3.

De fleste funksjonene bruker dagens dato som sammenligningsgrunnlag for å utføre ulike operasjoner. Eksempler på slike operasjoner er å tildele bookinger av et utleieobjekt (mer om sistnevnte i kapittel 6.2.5), eller å holde databasen oppdatert ved å flytte gamle bookinger eller oppdatere helligdagaer.

Videre følger en mer detaljert beskrivelse av funksjonen for å holde orden på helligdager. Denne har vi valgt å se nærmere på som et fordi den benytter tilleggsfunksjoner til sitt formål. Utdypende beskrivelser av de resterende funksjonene fra figur 6.3, ligger i vedlegg E.



Figur 6.3: Oversikt over funksjoner i FunctionAppService

UpdateHolidays(DateTime today)

For å holde orden på helligdager bestemte vi oss tidlig for å se etter et bibliotek for å oppdatere aktuelle helligdager. Helligdagene ville vært tidkrevende å vedlikeholde manuelt, og fordi en del helligdager er bevegelige kan de ikke hardkodes.

En oversikt over helligdager er nødvendig for å vite hvor mange poeng som skal trekkes ved booking, fordi poengsystemet for utleie bestemmer at det skal trekkes to poeng for en booking som går over helligdager, kontra ett til vanlig jmf. vedlegg F. Etter undersøkelser var det to bibliotek som var aktuelle fordi de har støtte for flere ulike land, disse er PublicHoliday og Nager.Date (NagerDate 2021, Microsoft 2021*g*). Også biblioteket NordicHolidays som støtter de nordiske landene ble vurdert, men dette ble valgt bort på grunn av størrelsen. Nager.Date og PublicHolidays inneholder mye av den samme funksjonaliteten der man blant annet kan hente offentlige helligdager i et tidsrom - for et bestemt land, sjekke om en bestemt dato er helligdag, etc (NagerDate 2021, Microsoft 2021*g*). Valget falt på Nager.Date fordi det var bedre dokumentert gjennom eksempler og forklaringer, noe vi allerede hadde erfart var viktig for sømløs integrasjon (se kapittel 9.3.1).

UpdateHolidays trigges første januar hvert år. Denne funksjonen sørger for at det ligger helligdager for inneværende år, samt neste år i databasen til enhver tid. Den sletter også alle helligdager for det foregående året, slik at databasen ikke holder på store datamengder som ikke er i bruk. Funksjonen benytter Nager.Date for å hente offentlige helligdager. I tillegg inngår uke 8 og uke 40 (vinterferie og høstferie) som helligdager i databasen, da dette er definert av oppdragsgiver som ferieuker. Disse har vi beregnet gjennom en egenlaget funksjon som følger under.

```
public IEnumerable<Holiday> AddWeekAsHoliday(int weekNumber, int year)
{
    DateTime mondayInWeek1 = ISOWeek.GetYearStart(year);
    var mondayInWeek = mondayInWeek1.AddDays(7*weekNumber);

    var holidaysInWeek = new List<Holiday>();

    for (var day=mondayInWeek; day <= mondayInWeek.AddDays(6); day=day.AddDays(1))
    {
        var holiday = new Holiday {
        Id = Guid.NewGuid().ToString(),
        DateOfHoliday = day.Date.ToUniversalTime()
        };
    holidaysInWeek.Add(holiday);
    }
    return holidaysInWeek;
}
```

Kodeliste 6.11: AddWeekAsHoliday legger til dagene i en bestemt uke

Som følge av at ukene vi legger til kan inneholde helligdager som allerede eksisterer i listen over offentlige helligdager, i tillegg kjøres alle helligdagene tillagt manuelt (uke 8 og 40) gjennom en løkke. Løkken sjekker om helligdagen allerede eksisterer i listen over offentlige helligdager slik at vi ikke lagrer duplisert data i databasen. Dette gjøres i UpdateHolidays() funksjonen.

6.2.5 AllocationService

AllocationService implementerer de funksjoner som brukes ved tildelingen av bookinger. Tjenesten består av en hovedfunksjon, og flere hjelpefunksjoner med hovedformål å gjøre tildelingslogikken oversiktlig.

```
public async Task Allocate(AllocationPeriod allocationPeriod)
{
    // Trig this action on day after applicationDeadLine

    var bookingRequests = await GetBookingRequests(allocationPeriod);
    if (bookingRequests == null)
    {
        throw new ArgumentNullException("Couldnt␣find␣any␣bookings␣in␣period:␣" +
            ↪ allocationPeriod.Id);
    }
    var bookingRequestsList = bookingRequests.ToList();
    int counter = 0;
    while (bookingRequestsList.Count() != 0)
    {
        Booking allocatedBooking = await GetBookingWithHighestPriority(
            ↪ bookingRequestsList);
        allocatedBooking.IsConfirmed = true;

        allocatedBooking.IsInHoliday = await bookingRepository.IsBookingInHoliday(
            ↪ allocatedBooking.StartDate, allocatedBooking.EndDate);

        User user = await userRepository.GetById(allocatedBooking.UserId);

        if (user.Points > 0) { user.Points --; }
```

```
22
23          if (allocatedBooking.IsInHoliday && user.Points > 0) { user.Points --; }
24
25          // get email for user from microsoft graph
26          MailboxAddress email = await graphService.GetEmailAsync(user.UserId);
27          emailService.SendBookingConfirmation(email, allocatedBooking);
28
29          await userRepository.Upload(user);
30          await bookingRepository.Add(allocatedBooking);
31
32          // Remove allocatedBooking from list of requests
33          bookingRequestsList.Remove(allocatedBooking);
34          counter ++;
35
36          // Delete conflicting Bookings and upload to blob
37          foreach (var bookingRequest in bookingRequestsList)
38          {
39              await DeleteBookingsWithConflicts(allocatedBooking, bookingRequest);
40          }
41      }
42      Console.WriteLine("Period␣contains␣" + bookingRequests.Count() + "requests,␣" +
            ↪  counter +
43          "␣bookings␣allocated␣on␣date:" + DateTime.Today.ToShortDateString());
44  }
```

Kodeliste 6.12: AllocationService prioriterer bookinger

Funksjonen henter alle bookinger for tildelingsperioden. Bookingene blir prioritert i henhold til regelverket iF. All påvirket data blir deretter oppdatert i databasen.

6.2.6 GraphService

GraphService blir benyttet til å hente informasjon om brukerne, slik som eposter og navn. Klassen inneholder fire funksjoner som blir brukt i ulike deler av systemet.

- GetUserId(string Email)
- GetGraphUser(string id)
- GetEmail(string id)
- GetUserName(string id)

Alle de fire funksjonene blir brukt for ulike formål. GetUserId blir benyttet når en bruker legges til i systemet. Brukeren legges til med eposten tilknyttet selskapets Microsoft konto. GetGraphUser benyttes i mappingen av en bruker til frontend, der både epost og brukernavn er nødvendig. GetEmail blir brukt for å hente eposten tilknyttet brukeren når det skal sendes eposter i systemet. GetUserName brukes i mapping til admin tjenester, slik at en admin får opp navnet til brukeren istedenfor brukerens id.

6.2.7 EmailService

MailKit og MimeKit

Som følge av at epostvarsling er en naturlig del av vårt system, måtte vi også finne en måte å sette opp dette på. Vi fant tidlig ut at Simple Mail Transport Protocol (Smtp) var en industristandard når det kommer til å sende eposter fra APIer(Pooroni 2021). Vi begynte derfor å undersøke innebygde løsninger i .Net Core. Dokumentasjonen her anbefalte å benytte Mailkit og lignende bibliotek i nye systemer, da deres egen SmtpClient klasse ikke støtter flere av de nyere protokollene knyttet til å sende epost(Microsoft 2021*i*). Valget endte dermed på MailKit for å etablere Smtp tilknytning til gmail (som vi har benyttet som en epost for testing av tjenesten). Bakgrunnen for dette er i tillegg til Microsofts anbefaling at biblioteket støtter de aller fleste autentiseringmetoder, noe som også er essensielt da oppdragsgiver vil benytte seg av en annen type konto enn hva vi har gjort. Biblioteket er stadig under utvikling, og anses derfor å være et fremtidsorient valg for vårt system(Microsoft 2021*d*). Dette rammeverket bygger på MimeKit som i MailKit blir brukt for å bygge opp innholdet av en epost med mottaker, avsender, emne og innhold(Microsoft 2021*d*). Det vil si at MailKit sender en MimeMessage (klasse i MimeKit) gjennom å opprette en sikker kobling til SMTP-klienten. Implementasjonen av dette er vist i kodeliste 6.13.

```
1  public bool SendEmail(MimeMessage message)
2  {
3  ...
4      using (var emailClient = new MailKit.Net.Smtp.SmtpClient())
5      {
6          emailClient.Connect(emailConfig.SmtpServer, emailConfig.SmtpPort, true);
7
8          emailClient.Authenticate(emailConfig.SmtpUsername, emailConfig.SmtpPassword
               ↪ );
9
10         emailClient.Send(message);
11         emailClient.Disconnect(true);
12     }
13 ...
14 }
```

Kodeliste 6.13: Tilkobling til SMTP-klient via Mailkit

EmailService klassen inneholder flere epostvarslinger knyttet til ulike system-oppgaver. Disse bygges opp gjennom MimeMessage klassen vist i kodeavsnittet under (6.14).

```
1  public bool SendBookingConfirmation(MailboxAddress toUserEmail, Booking booking)
2  {
3      Thread.CurrentThread.CurrentCulture = new CultureInfo("no-NO");
4      var message = new MimeMessage();
5      message.To.Add(toUserEmail);
6      message.From.Add(new MailboxAddress("Communicate␣utleiesider", emailConfig.
           ↪ SmtpUsername));
7
```

```
8       message.Subject = "Din␣booking␣fra␣" + booking.StartDate.ToLocalTime().
            ↪ ToShortDateString() + "␣-␣" +
9       booking.EndDate.ToLocalTime().ToShortDateString()  + "␣er␣tildelt";
10
11      message.Body = new TextPart(MimeKit.Text.TextFormat.Html)
12      {
13          Text = string.Format (@"<p>Du␣har␣blitt␣tidelt␣din␣booking␣for␣" + booking.
                ↪ Title + "␣fra␣" +
14          booking.StartDate.Date.ToLocalTime().ToShortDateString() +
15          "␣til␣" + booking.EndDate.Date.ToLocalTime().ToShortDateString() + ".<br><
                ↪ br>" +
16          "Med␣vennlig␣hilsen<br>" +
17          "Communicate␣utleietjeneste<br></p>")
18      };
19      try
20      {
21          SendEmail(message);
22      }
23      catch (System.Exception)
24      {
25          return false;
26      }
27      return true;
28  }
```

Kodeliste 6.14: Eksempel på hvordan en MimeMessage bygges opp

Alle epost-tjenestene i systemet har en boolsk returverdi. Diskusjon rundt valg av denne formen for feilmeldinger kan leses i kapittel 9.4.2. I tillegg til eposten vist over er følgende epost-tjenester tilgjengelig:

Funksjonsnavn	**Beskrivelse**
SendBookingCancellation(...)	Sender en bekreftelse om at bookingen ble kansellert
SendDeadlineReminder(...)	Sender en påminnelse om at søknadsfristen for et utleieobjekt nærmer seg
SendChecklistReminder(...)	Sender en påminnelse om at sjekkliste etter booking ikke er levert
SendListOfBookingsToAccountant(...)	Sender en formatert liste over alle bookinger den foregående måneden til regnskap
SendEmail(MimeMessage message)	Oppretter en sikker kobling til SMTP-klienten og sender eposten

Tabell 6.1: Liste over funksjoner i EmailService

6.3 Function App

Function Appen som skal styre automatiseringen av systemet ble implementert gjennom en mal for tidsbaserte Function Apps (timer triggers) laget av Microsoft (Microsoft 2020*h*). Triggeren baserer seg på CRON-uttrykk som er en måte å uttrykke tidsbaserte hendelser. Vi har satt vår trigger til 06.00 hver dag, uttrykt på kodelinje 2 i kodelsite 6.15. Her oppretter vi en klient som

sender et kall til "FunctionApp"-endepunktet i backend.

```
[FunctionName("Scheduler")]
public static async Task Run([TimerTrigger("0␣0␣6␣*␣*␣*")]TimerInfo myTimer,
    ↪ ILogger log,
Microsoft.Azure.WebJobs.ExecutionContext context)
{
    var config = new ConfigurationBuilder()
    .SetBasePath(context.FunctionAppDirectory)
    .AddJsonFile("local.settings.json", optional: true, reloadOnChange: true)
    .AddJsonFile("settings.json", optional: true, reloadOnChange: true)
    .AddEnvironmentVariables()
    .Build();

    var ApiUrl = config.GetValue<string>("ApiUrl");

    string endpoint = ApiUrl + "FunctionApp/Trigger";

    var requestMessage = new HttpRequestMessage(HttpMethod.Post, endpoint);
    requestMessage.Headers.Add("Ocp-Apim-Subscription-Key", config.GetValue<string
        ↪ >("Apim-Key"));

    try
    {
        var response = await httpClient.SendAsync(requestMessage);
        ...
    }
    ...
}
```

Kodeliste 6.15: Function App med http-klient

6.4 Testing

Som rammeverk for testing ble Microsofts egenutviklede rammeverk MSTest satt opp for dette. For å begrense enhetstestene til kun å teste den aktuelle enheten satte vi opp Mock-er. Bruk av Mock-er gjør det mulig å definere innsendt- og returverdi manuelt, slik at injiserte avhengigheter (injected dependencies) i praksis fjernes fra testen. Da er det kun logikken i funksjonen som testes som blir igjen. Det andre alternativet som ble vurdert var å sette opp og rive ned en testdatabase. Med bakgrunn i at vi ikke ønsket å teste all underliggende logikk, ble mocking av objekter valgt.

En testklasse blir initiert for hver modell. Dette innebærer i hovedsak å sette opp de mock-er for de klassene enheten benytter seg av. Hvordan dette ble løst er eksemplifisert gjennom BookingControllerTests i kodeliste 6.16.

```
[TestInitialize]
public void TestInit()
{
    // Set up Mock dependencies
    allocationPeriodRepositoryMock = new Mock<IAllocationPeriodRepository>();
    bookingRepositoryMock = new Mock<IBookingRepository>();
    bookingMapperMock = new Mock<IBookingMapper>();
    graphServiceMock = new Mock<IGraphService>();
```

```
9       emailServiceMock = new Mock<IEmailService>();
10      allocationServiceMock = new Mock<IAllocationService>();
11      userRepositoryMock = new Mock<IUserRepository>();
12
13      controller = new BookingsController
14      (
15          allocationPeriodRepositoryMock.Object,
16          bookingRepositoryMock.Object,
17          bookingMapperMock.Object,
18          emailServiceMock.Object,
19          graphServiceMock.Object,
20          allocationServiceMock.Object,
21          userRepositoryMock.Object
22      );
23
24      fixture = new Fixture();
25  }
```

Kodeliste 6.16: Testene ble initiert gjennom å sette opp mock-er for benyttede klasser

Testene er satt opp slik at de som hovedregel tester at alle forventede utfall trigger (som beskrevet i 5.6.1), for eksempel teste at ulik data resulterer i de forventede statuskodene som returverdi - eller at de feiler når det er forventet. Dette gjøres ved å definere at ulike variabler sendes inn i testfunksjonen. Et eksempel på dette kan sees i kodeliste 6.17, som er en test som validerer funksjonaliteten rundt om en booking skal tildeles direkte, eller lagres som en søkand. Her sendes ulik data inn for hver gang testen kjøres. Dette er definert i [DataRow]-ene som tas i mot i funksjonens parametre.

```
1   [TestMethod]
2   [DataRow(1, 3, 202)]
3   [DataRow(2, 1, 202)]
4   [DataRow(3, -1, 201)]
5   [DataRow(2, 10, 202)]
6   [DataRow(2, 0, 201)]
7   public async Task Add_ShouldAssign_ShouldReturnExpected(int endDay, int periodStart
        ↪ , int expected)
8   {
9       // Arrange
10      var input = fixture.Create<BookingDRO>();
11      input.StartDate = DateTime.Today.ToUniversalTime();
12      input.EndDate = input.StartDate.AddDays(endDay);
13
14      // Setup booking data
15      var mappedBooking = fixture.Create<Booking>();
16      bookingMapperMock
17      .Setup(mock => mock.Map(It.IsAny<BookingDRO>())).Returns(mappedBooking);
18      mappedBooking.StartDate = DateTime.Today.ToUniversalTime();
19      mappedBooking.EndDate = mappedBooking.StartDate.AddDays(endDay);
20      mappedBooking.IsConfirmed = false;
21      mappedBooking.IsInHoliday = false;
22
23      // Setup mock data
24      var currentPeriodMock = fixture.Create<List<AllocationPeriod>>();
25      allocationPeriodRepositoryMock
26      .Setup(mock => mock.GetCurrent(It.IsAny<DateTime>(), It.IsAny<string>())).
            ↪ ReturnsAsync(currentPeriodMock);
```

```
27
28      ...
29
30      // Act
31      var result = await controller.Add(input) as ObjectResult;
32
33      // Assert
34      result.StatusCode.Should().Be(expected);
35  }
```

Kodeliste 6.17: Både verdier for suksess og feil ble testet for å validere enhetene

Kapittel 7

Kvalitetssikring

Gruppen forsøkte gjennomgående å kvalitetssikre systemet, slik at det har funksjonaliteten som er beskrevet i kravspesifikasjonen (se kapittel 3). For dette har vi benyttet en rekke metoder som er beskrevet i de videre delkapittelene. I tillegg har vi hatt svært god nytte av ukentlige møter med veilederne våre, hvor vi har hatt gode diskusjoner med både intern veileder og kontaktperson i Communicate.

7.1 Kodekvalitet

For å skape et robust system var kodekvalitet et fokusområde. Vi valgte derfor å sette opp en regel (policy) i Azure DevOps som innebar at et annet medlem av utviklingsteamet måtte gjøre kodesjekk (code review) når ny funksjonalitet eller forbedringer skulle implementeres i systemet. Bruken vår av pipelines (se kapittel 8 for detaljer om disse) innebærer også en sjekk at bygger og godkjennes. Sammen gjorde disse reglene det mulig for oss å fange opp kode som ikke kompilerte, samt de fleste bugs og andre feil, før koden ble produksjonssatt.

I tillegg ble det gjennomført en større kodesjekk med oppdragsgiver i starten av siste utviklingsperiode. Dette innebar forslag til forbedringer innenfor både effektivisering av kode og dokumentasjon.

7.1.1 Kodestandard

For å oppnå høy kodekvalitet ble det før oppstart av utviklingen definert felles grupperegler for kodestandard. Disse reglene ble revidert etterhvert som gruppen ble kjent med verktøyene som ble benyttet slik at reglene var mer hensiktsmessige for hvordan vi faktisk jobbet. De reviderte kodestandardene kan leses i vedlegg D. Disse reglene gjorde det lettere for gruppen å skrive kode som var lettleselig for alle, fulgte normer i dokumentasjon, og dermed lettere å gjennomgående gjøre lik.

For å oppnå lik formatering mellom gruppemedlemmene benyttet vi også Prettier i frontend (da det ikke er støttet for C#) for både TypeScript, HTML og SCSS. Prettier omtaler seg selv som ”en kodeformaterer med meninger”, og er et verktøy som automatisk formaterer koden. Denne benyttet som standard de samme reglene vi allerede hadde fastsatt i kodestandarden vår for frontend. Samtidig innførte den ekstra regler knyttet til for eksempel hvor lange kodelinjer kan være, hvilken type anførseltegn som brukes til stringer osv.

7.1.2 Linting

Linting er et konsept som handler om å fjerne feil og svakheter i koden ved hjelp av lintere. Lintere er verktøy som automatisk sjekker kildekoden for programmatiske og stilistiske feil ved hjelp av statisk kodeanalyse. Linterne som gruppen benyttet seg av kjører kontinuerlig så lenge vi har det relevante verktøyet åpent, slik at linterfeil har blitt rettet fortløpende.

I .NET er eksterne linterverktøy mindre utbredt, men det er mer bruk av Microsofts egen linter Omnisharp. Denne kan installeres til de fleste IDE-er, men kommer inkludert med både Visual Studio og Visual Studio Code, som gruppen brukte. Omnisharp viser advarsler og programmeringsfeil, og gjør det - som lintere flest - enklere å finne feil underveis (Chiaretta 2017).

I frontend er eksterne linterverktøy mer vanlig. Vi valgte å benytte oss av ESLint som i utgangspunktet ble laget for JavaScript, men som har en egen versjon for TypeScript. ESLint er modulært og gir konfigureringsmuligheter på hvilke ting som skal sjekkes, sammen med mulighet for å plugge inn andres (standardiserte) konfigurasjoner (OpenJS Foundation 2017). Fordi dette var et område ingen av gruppemedlemmene hadde noe særlig erfaring med valgte vi å gå for de standard inkluderte konfigurasjonene i ESLint, samt tilleggskonfigurasjonene laget for Angular.

7.2 Avhengighetshåndtering

Backend benytter få eksterne avhengigheter - de fleste er ferdigbygde biblioteker fra Microsoft. Likevel passet vi her på å benytte de nyeste versjonene av disse bibliotekene ved oppstart av prosjektet, og oppdaterte bibliotekene til nyere versjoner der det eksisterte mot slutten av prosjektet.

Frontend benytter Angular som er et omfattende rammeverk, samt ulike pakker for ferdiglagde komponenter som lastes ned gjennom pakkebehandleren Node Package Manager. Dette skaper en rekke avhengigheter (dependencies) som er utviklet av andre, hvor det kan oppstå svakheter.

Dette er naturlig nok et kjent problem, derfor har npm et system for innmelding av svakheter og videreformidling av disse. Når kommandoen `npm audit` kjøres, gjøres en sjekk på om det har dukket opp svakheter til noen av avhengighetene til programmet, og disse vises frem. Ved å kjøre `npm audit fix` vil pakkene med svakheter oppdateres til nyere versjoner som ikke lenger har disse feilene (eventuelt `npm audit fix --force` for å også oppdatere pakker som kan ha endringer som er ødeleggende).

Registeret med svakheter som dukker opp i `npm audit` oppdateres fortløpende (npm Docs 2021*b*), og selv om vi jevnlig hadde kjørt disse oppdateringene under utviklingen dukket det to uker før innlevering av oppgaven opp enda flere svakheter - kjøringen av `npm audit` i starten av mai kan sees i kodeliste 7.1 (forkortet med [...] enkelte steder).

```
junev@DESKTOP-07DULSM MINGW64 ~/communicate_rental_frontend-dev (main)
$ npm audit
# npm audit report

[...]
6 vulnerabilities (1 moderate, 5 high)

To address issues that do not require attention, run:
  npm audit fix
[...]

junev@DESKTOP-07DULSM MINGW64 ~/communicate_rental_frontend-dev (main)
$ npm audit fix

removed 2 packages, changed 4 packages, and audited 1689 packages in 3s

# npm audit report
[...]
2 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

junev@DESKTOP-07DULSM MINGW64 ~/communicate_rental_frontend-dev (main)
$ npm audit fix --force
npm WARN using --force Recommended protections disabled.
npm WARN audit Updating karma to 6.3.2,which is a SemVer major change.

added 5 packages, removed 34 packages, changed 12 packages, and audited 1660
    ↪ packages in 11s

found 0 vulnerabilities
```

Kodeliste 7.1: "npm Audit"-kommando kjørt tidlig mai

7.3 Dokumentasjon

Fordi applikasjonen vår skal videreutvikles av andre, er dokumentasjon en viktig del av kvalitetssikringen. God dokumentasjon vil gjøre det enklere å plukke opp prosjektet vårt for en som ikke er kjent med det, og gjør det raskere å komme i gang med videreutvikling.

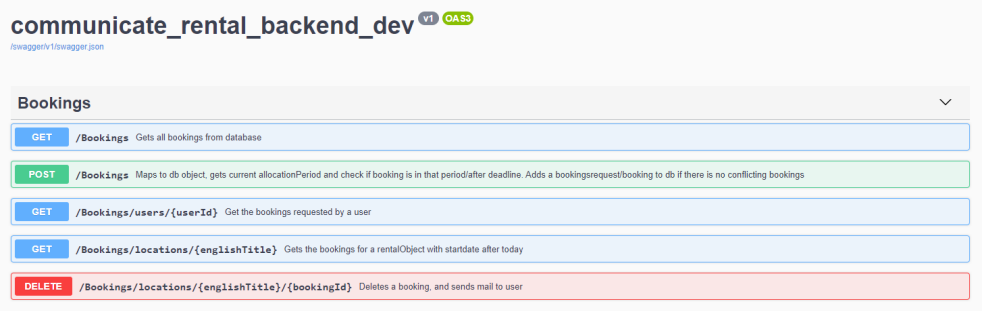
7.3.1 Repo Wiki

Som de aller fleste plattformer for versjonskontroll har prosjekter satt opp i Azure DevOps mulighet til å benytte en informasjonsside (Wiki) for prosjektet. Denne siden kan være så detaljert eller overordnet som ønsket, men formålet er å dokumentere prosjektet slik at andre utviklere skal kunne bruke mindre tid på forståelse av arkitekturen. Dette kapittelet inneholder detaljer rundt vår dokumentasjon av applikasjonen. Med bakgrunn i valgene av dokumentasjon for øvrig (se videre i kapitlet), ble informasjonssiden brukt til mer overordnet designmessige valg. Mer konkret inneholder Wiki-en detaljer rundt hvordan de

ulike delene henger sammen. I tillegg inneholder den flere av figurene brukt i denne oppgaven for beskrivelse.

7.3.2 API - Swagger

Swagger er et verktøy laget for å forenkle utvikling ved bruk av APIer. Som en del av dette kan Swagger benyttes for å autogenerere dokumentasjon for APIer. Dette legges til som en konfigurasjon ved oppstart av systemet, og genererer en XML-fil for endepunktene med tilhørende kommentarer. I tillegg inneholder Swagger funksjonalitet for å opprette et grensesnitt for tilknytningspunktene til applikasjonen som utvikles (Swagger 2021). Dette er nyttig funksjonalitet som forklarer hva de ulike endepunktene gjør, samtidig som den gir eksempler på hvilken type data hvert endepunkt forventer, og hvilke returverdier som blir returnert. I tillegg kan kall gjøres direkte mot backend via Swagger, slik at man kan gjøre testkall. Dette gjør det enklere å bruke APIet, og gjør det enklere å utvikle frontend uten å måtte grave i kildekoden til backend.



Figur 7.1: "Swagger UI" - brukergrensesnitt for bruk og dokumentasjon av API-endepunkter

Figur 7.1 viser de endepunktene i kontrolleren som styrer funksjonalitet i APIet tilknyttet bookinger. Hvert endepunkt kan ekspanderes ved å klikke på det, slik at eksempler på data som forventes og returdata vises, og systemresponsen kan testes.

7.3.3 Kildekodedokumentasjon

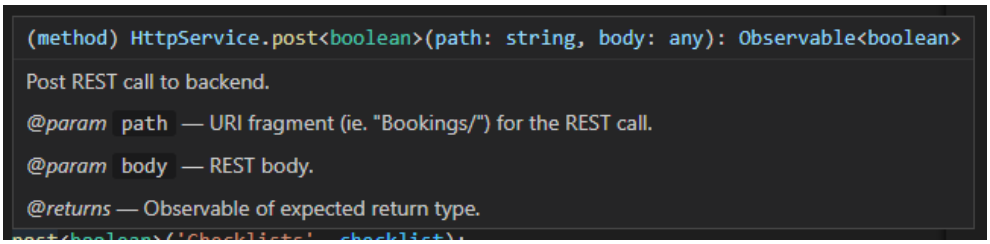
Gjennomgående benyttet gruppen normal kommentering innad i funksjoner for å klargjøre funksjonalitet, forklare hvorfor ting er gjort som de er, forklare hva koden gjør i de tilfellene det ikke er lettleselig og lignende. I tillegg har gruppen benyttet XML-dokumentasjon i backend og JSDoc-dokumentasjon i frontend.

Både XML-dokumentasjon og JSDocs-dokumentasjon er metoder for å legge til kommentarer i selve kodebasen med syntaks som gjør at dokumentasjonen er synlig når man holder musepekeren over funksjonen når den er brukt andre

steder, og er særlig nyttig for funksjoner i klasser som kan injectes. Et eksempel på bruk av JSDocs kan sees i kodeliste 7.2, og hvordan det ser ut når man holder over funksjonen der det er brukt kan sees i figur 7.2.

```
1  /**
2  * Post REST call to backend.
3  *
4  * @param path - URI fragment (ie. "Bookings/") for the REST call.
5  * @param body - REST body.
6  * @returns Observable of type T.
7  */
8  public post<T>(path: string, body: any): Observable<T> {
9      // The function does things which are not relevant for the example
10 }
```

Kodeliste 7.2: Eksempel på JSDoc-dokumentasjon av httpService post



Figur 7.2: Hvordan JSDocs vises i koden

XML-dokumentasjon i C# benyttes på samme måte som JSDocs-eksemplene, men har en litt annen syntaks. Et eksempel på dette kan sees i kodeliste 7.3. Denne dokumentasjonen vises på samme måte som eksemplet i figur 7.2, og er også det som gjør at dokumentasjonen dukker opp i Swagger som vist i figur 7.1.

```
1  /// <summary>
2  /// Gets graphUser by id, and calls graph to get the name of a user
3  /// </summary>
4  /// <param name="userId">Microsoft graph ID</param>
5  /// <returns>String that contains name of the user</returns>
6  public async Task<string> GetUserName(string userId)
7  {
8      // The function does things which are not relevant for the example
9  }
```

Kodeliste 7.3: Konfigurasjoner definert i Startup.cs

Igjen er dette dokumentasjon som er med på å gjøre det enklere å plukke opp for en utvikler som ikke er kjent med applikasjonen.

7.4 Testing

7.4.1 Programvaretesting

I utgangspunktet siktet vi mot høy dekningsgrad av tester av hele systemet, men det ble tidlig klart at dekningsgrad utover hele systemet ville være vanskelig å oppnå. Derfor fokuserte vi på enhetstesting av logikken i backend - den viktigste funksjonaliteten for å sikre at systemet fungerer. I tillegg ble det gjort noe testing i frontend, samt mye manuell testing i forbindelse med å forsikre seg om at UI og datahenting fungerte som forventet.

Startfasen av utviklingen bestod i hovedsak av komponent- og systemtesting, for å oppnå ende-til-ende kommunikasjon. Etterhvert som fokuset rettet seg mot automatisering av systemet ble verktøy for enhetstesting satt opp i backend.

Frontend

I frontend gjorde vi et forsøk på å implementere automatisert testing, men klarte aldri å prioritere å lese oss opp på hvordan vi skulle få til gode tester med tanke på avhengighetsinjeksjon og lignende - derfor ble hovedvekten av testingen manuell. Siden mesteparten av det logiske ble gjort av backend, fokuserte vi på å teste at UIet oppførte seg som forventet - særlig i forbindelse med livssyklusene til de ulike komponentene, at korrekt data ble sendt til backend, og at dataen hentet fra backend ble vist frem korrekt.

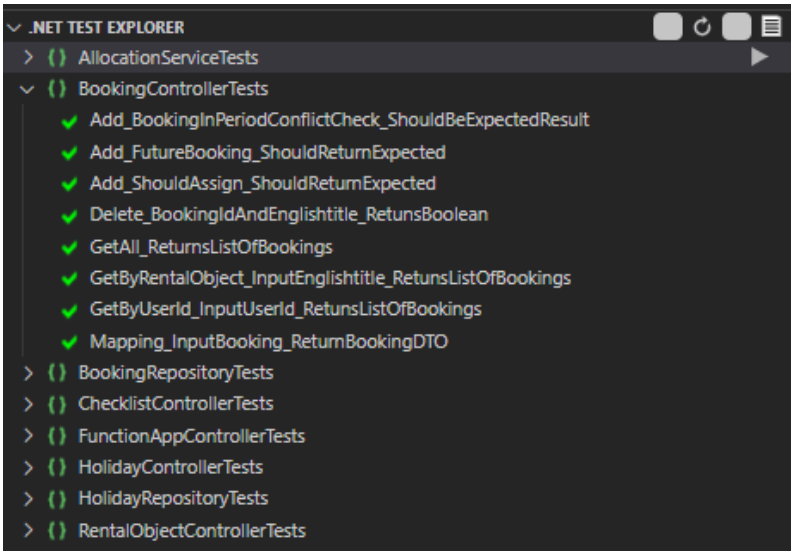
Feil som ble oppdaget gjennom denne typen testing var blant annet at bildekarusellen og kartene i utleieobjektene ikke ble korrekt oppdatert når vi navigerte mellom utleieobjektene ved bruk av sidemenyen - hver navigasjon førte til ytteligere bilder/kart etter hverandre. Dette ble rettet ved å endre hvordan komponentene lastet inn data. Det ble også oppdaget at bookingoversikten vist i kalender som er synlig for adminer viste alle datoer en dag for kort. Årsaken her ble oppdaget etter grundigere undersøkelse av den ferdiglagde kalenderkomponentens dokumentasjon, og deretter rettet.

Backend

I startfasen ble Swagger benyttet hyppig for manuell testing av endepunkter i APIet. Etterhvert som mer funksjonalitet forelå, ble en kombinasjon av Swagger og enhetstesting satt opp for APIet.

Det ble etterhvert utarbeidet en testplan med hovedformål å dekke det vi anså som systemkritiske operasjoner. Årsaken til dette var at vi prioriterte å levere et komplett system, fremfor å skrive unit-tester for funksjonalitet vi var sikre på fungerte gjennom komponent- og systemtestingen som allerede var gjennomført. I testplanen for APIet ble tjenester med funksjonalitet som er avgjørende for korrekt datalagring prioritert. Dette innebar i første omgang POST-kall og logiske operasjoner tilhørende følgende tjenester:

1. AllocationService
2. BookingsController
3. FunctionAppController
4. ChecklistController
5. Øvrige kontrollere



Figur 7.3: Test explorer som viser oversikt over tester

AllocationService ble prioritert som følge av at automatiseringen av systemet foregår i denne tjenesten, med mye logikk bakenforliggende. Kontrolleren for bookinger ble ansett som den mest systemkritiske kontrolleren gjennom at den inneholder innformasjon for systemets hovedoppgave. FunctionAppControlleren er kontrolleren som trigger ovenfornevnte AllocationSevice, og er på den måten avgjørende for systemet. ChecklistControlleren har funksjonalitet som vil være viktig i brukertestingen gjennom at den returnerer en sjekkliste som enten er tom eller fylt ut, og var funksjonalitet vi ønsket å teste før brukertestingen ble igangsatt. For å holde oversikt over implementerte tester, ble støtteverktøyet .Net Test Explorer tatt i bruk. Anvendelsen av dette verktøyet er vist i figur 7.3 gjennom en totaloversikt over gjennomførte tester med tilhørende fargekode for teststatus - grønt for vellykkede tester og rødt for feilende tester.

7.4.2 Brukertesting

Organisering og gjennomføring

Brukertesting ble gjennomført over en uke i begynnelsen av siste utviklingsfase/sprint av prosjektet. Her ble det utarbeidet en informasjonstekst som inneholdt tilgjengelig funksjonalitet, hva vi særlig ønsket tilbakemelding på, kjente

feil, og hvordan vi ønsket å motta tilbakemeldinger. Den utsendte teksten, samt oppsummert og rå data kan leses i vedlegg H

Formålet med brukertestingen var å avdekke svakheter og mangler, - fra et brukervennlighetspektiv, for både vanlige brukere og adminer. Måten vi gjennomførte testing på var at kontaktpersonen vår i selskapet fikk tak i et utvalg av personer som var interesserte i å teste den nye løsningen, både 4 personer som kunne teste som vanlige brukere samt 3 som adminer. Disse fikk tilsendt en epost med nettadressen de kunne besøke for testing, informasjon om hvordan og når vi ønsket tilbakemelding, oppgaver vi ønsket at de gjennomførte, samt en oversikt over kjente feil/mangler (for å unngå overveldene tilbakemelding på disse, men heller avdekke ukjente svakheter).

Vi valgte å gjennomføre tester på denne måten, da Korona-situasjonen gjorde det uaktuelt å gjennomføre fysisk testing, og dette var metoden som gjorde det mest fleksibelt for testerne våre - og derfor måten vi anså kom til å gjøre at vi fikk mest og grundigst tilbakemeldinger.

Oppgavene vi ga testerne var å navigere seg rundt på siden, å legge til en ny booking både i nær og fjern fremtid, å kansellere en booking, å gjennomgå en sjekkliste, og å sende inn en tilbakemelding via innebygd funksjonalitet. Admintesterne ba vi også om å legge til nye brukere i systemet, å legge til en ny lokasjon, og å se at bookingoversikten og de innsendte sjekklistene oppførte seg som forventet.

Resultater

De fleste tilbakemeldingene som kom var småfeil som var nevnt av en eller to av testerne, men det var to smertepunkter som dukket opp fra flere testere - at bookinger kunne legges inn overlappende, og at brukerne må kunne se hva som er fylt inn i sjekklisten og at den må kunne oppdateres eller deaktiveres etter at den er innsendt.

Tabell 7.1 viser en oversikt over ønskene testerne kom med, hvor alvorlig vi anså at mangel på oppfyllelse av ønsket var, hvor høyt vi prioriterte å innfri ønsket, og status ved avslutning av utvikling. Nummer markert med stjerne (*) har ekstra kommentar under tabellen. Vi vurderte alvorlighetsgrad med særlig tanke på hvor systemkritisk ønsket var, men også i hvor stor grad det påvirket brukervennligheten. Prioriteringer ble gjort fortløpende basert på alvorlighetsgrad, tidsperspektiv og grad av viktighet i form av brukeropplevelse. I mange av tilfellene var ønskene såpass små at de lett kunne gjennomføres, slik at prioritet i all hovedsak hadde lite å si - det ble gjennomført likevel.

Nr	**Ønske**	**Alvorlig-hetsgrad**	**Prioritet**	**Status**
1	Rette opp i at et felt ikke oppdaterte seg korrekt i RentalObject-skjemaet.	Tolererbart	Middels	Utbedret

2*	Gjøre sjekklisten mer brukervennlig.	Alvorlig	Høy	Utbedret
3	Retting av språkfeil på en knapp.	Tolererbart	Middels	Utbedret
4*	Mulighet for å sjekke navigasjon i kartene.	Tolererbart	Lav	Utbedret
5	Ønske om automatisk utsendelse av informasjon før avreise.	Tolererbart	Lav	I backlog
6*	Mer informasjon om sengeplasser, bad, stuer osv.	Uproblematisk	Lav	Utbedret
7	Fritekstfelt i sjekklisten.	Tolererbart	Middels	Utbedret
8	Gjøre alle cards like store.	Tolererbart	Middels	Utbedret
9	Epostbekreftelse ved kansellering.	Alvorlig	Høy	Utbedret
10	Tooltip om hvor sjekklisten sendes.	Tolererbart	Lav	Utbedret
11	Ikke tillate overlappende bookinger.	Alvorlig	Kritisk	Utbedret
12	Fikse at bildekarusell og kart ikke oppdaterer seg ved navigasjon mellom utleieobjekter.	Alvorlig	Høy	Utbedret
13	Ønske om rik tekst i tekstfeltene for registrer lokasjon.	Tolererbar	Lav	I backlog
14	Ønske om 24-timersklokke istedenfor AM/PM i registrering av lokasjon.	Tolererbar	Middels	Utbedret
15*	Ønske om god håndtering av klokkeslett.	Alvorlig	Medium	Utbedret/ i backlog
16	Fjerne piltaster for pris i registrer lokasjon.	Uproblematisk	Lav	Utbedret
17	Restriksjon på størrelser og format på bilder på utleieobjekt.	Uproblematisk	Lav	I backlog
18	Tilbakemelding ved registrering av utleieobjekt.	Alvorlig	Høy	Utbedret
19	Avbryt-knapp på booking.	-	-	Eksisterer allerede
20	Totaloversikt over bookinger for adminer.	Alvorlig	Høy	Utbedret
21	Adminkommentar på bookinger.	Tolererbar	Middels	Utbedret
22*	Egen pil for valg av årstall på brukeradministrasjon.	Tolererbar	Middels	Utbedret
23	Annen sortering av innsendte sjekklister.	Tolererbar	Middels	I backlog

24*	Fjerning av adresse som ikke er i bruk.	-	-	Finnes ikke

Tabell 7.1: Kartlegging av tilbakemeldinger under brukertesting

Kommentarer:

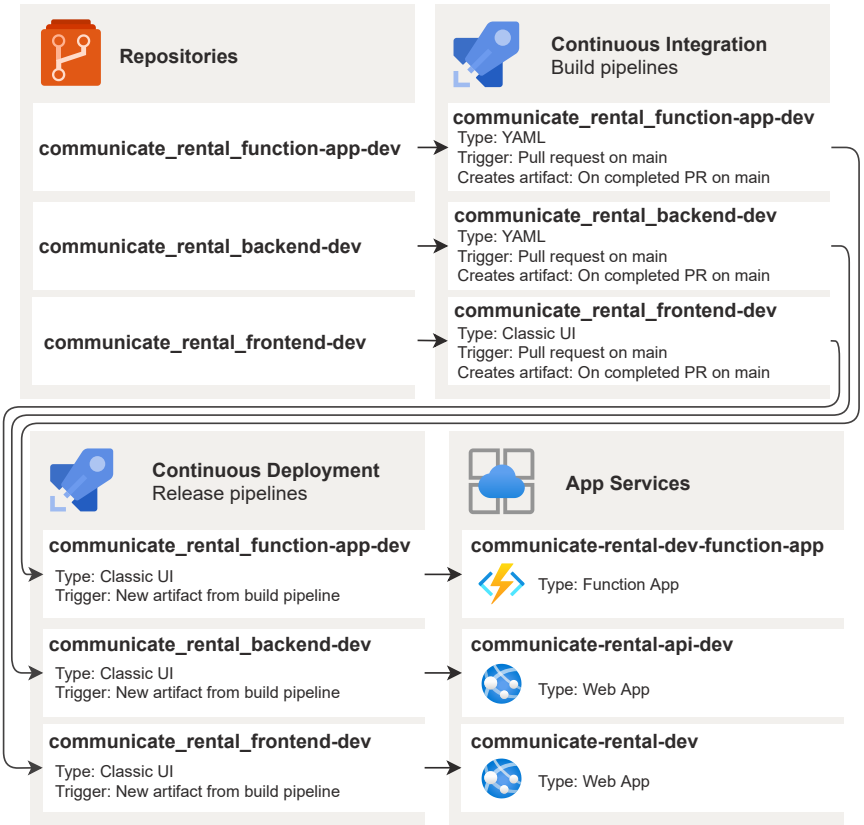
- 2: Det kom mange forslag og ønsker knyttet til sjekklisten, som bunnet i ønsket om mer brukervennlighet. Her ble det oppdatert slik at brukeren kunne se hva den allerede hadde fylt ut etter innsendelse, samt mulighet for å oppdatere den og sende inn på nytt.
- 4: Å gi navigasjonsmulighet i selve kartet ville vært veldig mye jobb. Vi gjorde et kompromiss for forbedring ved å legge til en knapp på kartet som ledet til ekstern karttjeneste med lokasjon allerede markert for å forenkle dette.
- 6: Dette var allerede forbedret fra tidligere løsning, så her mistenker vi brukeren har oversett informasjonen. Det kan jo likevel være en indikasjon på at det burde tydeliggjøres.
- 15: Her gjorde vi om slik at bookingene ble lagt til med spesifiserte start- og sluttidspunkt. Det kan likevel være feil knyttet til for eksempel bytte mellom sommer- og normaltid som ikke er avdekket, derfor er det også lagt i backloggen for vidre undersøkelse.
- 22: Her gjorde vi en forbedring ved å fjerne behovet for årstall ved å ikke lagre det, og legge til en tooltip om dette.
- 24: Her har vi ingen anelse om hvilken adresse det er snakk om. Vi mistenker kanskje at brukeren her har gått inn på en av linkene til de sosiale mediene og sett en adresse som ikke er i bruk der.

Kapittel 8

Produksjonssetting

8.1 Bakgrunn

Oppdragsgiver spesifiserte i oppgaveteksten at vi skulle benytte continuous integration og continuous deployment (heretter CI/CD) via build- og release-pipelines i Azure DevOps. Dette ble satt opp tidlig i utviklingsprosessen, slik at applikasjonen kontinuerlig har vært produksjonssatt og oppdatert under utvikling. Vi satt opp pipeline-ene slik at hovedgrenen i hvert av repoene produksjonssettes til to Azure Web Apps og en Azure Function App. En oversikt over hvordan dette ble satt opp kan sees i figur 8.1. Valget av Web Apps og Function Apps er mer grundig beskrevet i kapittel 4.1.1.



Figur 8.1: Oversikt over CI/CD i prosjektet

Azure DevOps har to muligheter for oppsett av pipelines - den nyere metoden som benytter YAML-filer, og Azures orginale verktøy hvor pipelines settes opp via et UI. En av de sentrale fordelene ved bruk av YAML er at det er kodebasert, slik at det må gjennom code review som nevnt i kapittel 7. I tillegg gjør bruk av YAML-filer det enklere å sammenligne en oppdatert pipeline med en tidligere versjon, og pipelinen vil følge versjonen til koden dersom det reverseres til en tidligere versjon (f.eks. for å se hvorfor oppdatering av pipelinen gikk galt). Azures originale UI-verktøy har noe mer funksjonalitet enn YAML fordi det er mer modent, men er også planlagt at skal utfases i fremtiden (Leung 2019).

8.2 Build pipelines

Build-pipelinen har som hovedoppgave å bygge prosjektet og publisere artifakter (en samling av filer produsert gjennom bygging), men kan også brukes til blant annet substitusjon av variabler, å verifisere at tester kjører grønt, og å verifisere at et prosjekt bygger (uten publisering av artifakt).

På bakgrunn av fordelene ved YAML, samt at det klassiske UIet er planlagt deprikert, ønsket vi å benytte YAML hvor mulig. YAML-pipeline benyttes derfor for backend og Function Appen (hvorfor dette ikke gjøres i frontend blir diskutert senere). Kodeliste 8.1 viser YAML-filen brukt til backend, men Function Appen har en identisk build-pipeline.

```
1  trigger:
2  - main
3
4  pool:
5    vmImage: 'windows-latest'
6
7  variables:
8    buildConfiguration: 'Release'
9    Parameters.projects: '**/*.csproj'
10
11 steps:
12 - task: DotNetCoreCLI@2
13   displayName: Build
14   inputs:
15     projects: '$(Parameters.projects)'
16     arguments: '--configuration $(BuildConfiguration)'
17
18 - task: DotNetCoreCLI@2
19   displayName: Test
20   inputs:
21     command: test
22     projects: '$(Parameters.TestProjects)'
23     arguments: '--configuration $(BuildConfiguration)'
24
25 - task: DotNetCoreCLI@2
26   condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
27   displayName: Publish
28   inputs:
29     command: publish
```

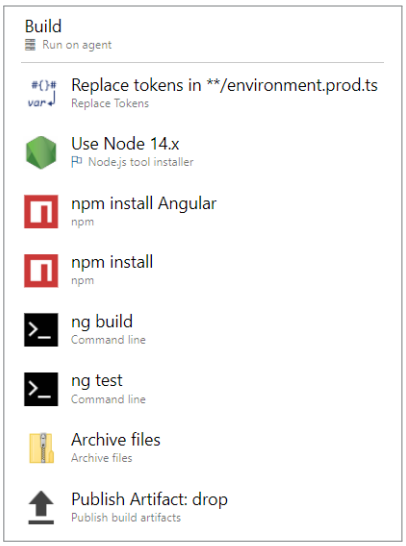
```
30      publishWebProjects: True
31      arguments: '--configuration $(BuildConfiguration) --output $(build.
            ↪ artifactstagingdirectory)'
32      zipAfterPublish: True
33
34  - task: PublishPipelineArtifact@1
35    condition: and(succeeded(), ne(variables['Build.Reason'], 'PullRequest'))
36    displayName: 'Publish Artifact'
37    inputs:
38      PathtoPublish: '$(build.artifactstagingdirectory)'
```

Kodeliste 8.1: YAML-fil for build pipeline til backend

Kodeliste 8.1 viser hvordan build-pipelinen for YAML-løsningen er satt opp slik at build- og testoppgavene (tasks) blir kjørt hver gang pipelinen trigger, mens lagring av artifakter og publisering av disse kun gjennomføres når en merge til main godkjennes (se kodelinjer 26 og 35). De foregående stegene må også være vellykkede. Fordelen med dette oppsettet er at build og test kjøres på pull requests, slik at det automatisk kan sjekkes at systemet bygges og testene passerer, før pull requests kan merges inn i hoved-branchen. Dette er også spesifisert i stegene i frontend.

I alle tre repoene hadde vi behov for å legge til variabler i konfigurasjonsfilene for at programmet skal ha tilgang til nødvendige nøkler. Av sikkerhetsmessige årsaker er det lite ønskelig at disse variablene skal ligge i klartekst i repoene. For å unngå dette er det satt opp konfigurasjonsfiler med tomme verdier som substitueres i build/release-pipelinene. I .NET-prosjektene fant vi ut at dette enklest gjøres gjennom innebygd funksjonalitet i release pipelinen (se delkapittel 8.3). I frontend prosjektet må det derimot brukes en egen task (”Replace tokens” i figur 8.2) til dette formålet.

”Replace tokens”-tasken fungerte ikke i YAML, så vi valgte å bruke det klassiske UI-et i frontend for å få det til å fungere. I denne pipelinen gjøres substitusjonen før build-steget, fordi det er enklere å gjøre substitusjonen når vi vet hvor variablene ligger i TypeScript-koden, enn det er å måtte finne de etter at det har blitt kompilert til JavaScript.



Figur 8.2: Build pipeline frontend

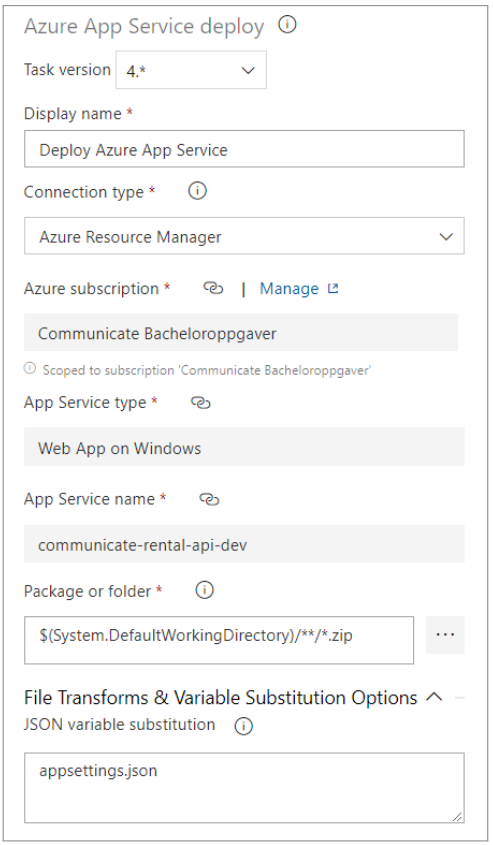
Figur 8.2 viser hvordan pipelinen blir satt opp for frontend. De videre stegene etter ”Replace tokens” i denne pipelinen skiller seg også fra .NET-prosjektene ved at både Node og Angular må installeres før build og test kan kjøres. Build og test-taskene fungerer likt som backend, hvor alt før produksjonen av artifakt (”Archive files”) kjøres hver gang pipelinen trigges, mens det kun lages et artifakt som lagres for publisering når merge til main gjennomføres.

8.3 Release pipelines

En release-pipeline har som hovedoppgave å hente artifaktet som ble laget av den tilhørende build-pipelinen og deretter publisere det til spesifisert sted. Figur 8.3 viser et eksempel med pipelinen for backend.

Pipelinene som sørger for publisering er veldig like. Sett bort fra navngiving og hvor variabelsubstutisjon gjøres er det ingen vesentlige forskjeller, utover at Function Appen benytter en pipeline ment for nettopp Function Apps i sin pipeline.

Vi trodde i utgangspunktet at release-pipelinene måtte settes opp med den klassiske UI-en fordi valget som het "Releases" under "Pipelines" kun inneholdt det klassiske UI-et. Senere i utviklingen oppdaget vi at vi kunne ha benyttet YAML ved å sette opp release som et nytt steg i samme skjermbilde som vi satt opp build-pipelinene våre. På dette tidspunktet hadde vi allerede et fungerende oppsett med klassisk UI, så vi anså det som dårlig prioritering av tid å konvertere det til YAML på det tidspunktet.



Figur 8.3: Release pipeline backend (ubrukte valg er klippet bort)

Kapittel 9

Diskusjon

9.1 Krav og mål

Prosjektets resultatmål var å utvikle en webløsning for utleie av oppdragsgivers feriesteder, med fokus på å automatisere så mye som mulig av det administrative arbeidet ved utleien, samt å forbedre brukeropplevelsen til nettsiden. Løsningen skulle som minimum fungere på PC/Tablet, og skulle dokumenteres godt underveis slik at andre kunne fortsette videreutviklingsarbeidet etter endt bachelorperiode.

Gruppen har oppfylt resultatmålene, og de andre kravene satt til oppgaven - både use cases, funksjonelle krav, krav til sikkerhet og øvrige krav. I tillegg har gruppen strukket seg lenger på flere områder, ved blant annet å gjøre nettsiden gjennomgående mobilvennlig og responsiv utover minstekravet. Vi har også forsøkt å lage en robust applikasjon som tåler at både database, frontend, backend og Function Appen kan byttes ut med minimal kodeendring i de andre delene av applikasjonen.

De fleste effektmålene er naturlig nok vanskelig å dømme før applikasjonen er tatt i bruk. Likevel er vi trygge på at også disse oppfylles når systemet tas i bruk, nettopp fordi vi har et fungerende system som i mye større grad er automatisert enn det eksisterende, og vi fikk gode tilbakemeldinger på brukeropplevelsen under brukertestingen.

Alle gruppemedlemmene har hatt en betydelig kompetanseheving gjennom prosjektet, og har blitt godt kjent med både .NET, Azure og Angular. Vi har lært å innhente relevant informasjon for å ta valg som resulterer i en god løsning, og vi har blitt enda bedre på å finne løsninger på problemer - enten ved å finne informasjon som direkte kan løse problemet, eller ved å finne måter å omgå problemet på. Vi har også fått gode ferdigheter innen dokumentasjon av prosjektet, og vi har tilegnet oss erfaringer med kanban, og med elementer fra Scrum. På denne måten anser vi at også læringsmålene er oppfylt.

9.2 Prosess og organisering

Gruppen mener at gjennomføringen av utviklingsmodellen Kanban fungerte godt. Kanbantavlen var et nyttig verktøy som gjorde det enkelt for gruppemedlemmene å alltid plukke de viktigste oppgavene. I tillegg gjorde den det enklere å omprioritere oppgaver og viktigheten deres gjennom evalueringsmøter, men også løpende etterhvert som vi oppdaget oppgaver som måtte gjøres.

En ting vi her nok kunne vært bedre på hadde vært å være mer konsekvente på omtrent hvor omfattende en oppgave var. I begynnelsen av prosjektet hadde vi ekstremt store og generiske oppgaver som kunne ta mange dager å gjennomføre, mens vi mot slutten la inn oppgaver som var veldig detaljspesifikke. Dette var en naturlig progresjon av bruken av tavlen ettersom vi forsto mer hva som måtte på plass og hva en oppgave faktisk innebar.

Det ble gjennomført daglige statusmøter internt i gruppen hvor alle typer problemstillinger ble tatt opp. I begynnelsen av en utviklingsperiode var dette gjerne knyttet til planlegging av perioden, mens videre møter ofte var idédugnader for hvordan å løse problemstillinger eller debug-økter. Ukentlige møter med kontaktperson hos oppdragsgiver fungerte også godt, hvor vi både kunne få oppklaring av hvordan det var ønsket at funksjonalitet skulle fungere, og få teknisk hjelp dersom vi sto fast uten å finne løsninger selv. Også (stort sett) ukentlige møter med internveileder hos NTNU fungerte godt. Disse møtene brukte vi hovedsaklig til hjelp med rapporten, men også til diskusjon av hva vi burde vektlegge i de ulike delene av prosjektet - noe vi hadde stor nytte av.

Dersom bacheloroppgaven hadde vært en yrkessituasjon kunne tidsaspektet ved enkelte av debug-øktene vært noe problematisk. Som nevnt i kapittel 2.2.2 ble det anslått at det gjennomsnittlig ble brukt ca. 12 arbeidstimer totalt per uke. Vi ser for oss at i en yrkessituasjon må oppgaver i større grad løses selvstendig, men tror vi likevel brukte færre timer totalt ved å unngå at ett gruppemedlem satt seg helt fast. Gjennomføringen vår av debug-økter var frivillig for gruppemedlemmene, men alle valgte som regel å være med på bakgrunn av læringsaspektet ved denne aktiviteten.

9.2.1 Utviklingsperioder

Generelt vurderer vi metoden for oppsett av utviklingsperioder som en god løsning. Som nevnt ble prosjektperioden delt opp i treukersperioder, hvor hver periode hadde ulikt fokus. Dette fungerte veldig bra, og gjorde at vi fikk på plass en minimumsløsning tidlig i prosjektet. Begrensningen på tre uker viste seg å være et godt valg - det var en passe mengde tid til å få på plass det meste av funksjonaliteten vi ønsket å få på plass i perioden, samtidig som det ikke var så langt at vi ble sittende å tvinne tommeltotter eller å kun forbedre tidligere arbeid uten å komme videre.

Likevel dukket det naturlig nok opp problemer som forskjøv på disse utviklingsperiodene, særlig i begynnelsen. I den første perioden møtte vi på klassiske problemer knyttet til manglende kunnskap og erfaring, slik at ting tok lenger tid enn det behøvde. Særlig knyttet disse problemene seg til følgende:

- Å få til oppkoblingen mot databasen fra backend viste seg å være mer utfordrende enn forventet, dette var særlig knyttet til at vi benyttet en nylig oppdatert versjon av .NET - dette er videre diskutert i delkapittel 9.3.1.
- Å sette opp CI/CD viste seg å ikke være helt trivielt uten den grunnleg-

gende forståelsen for hvordan systemet fungerte. Vi hadde særlig trøbbel med pipelinen til frontend, også dette mistenker vi var knyttet til at vi brukte nyere versjoner av pipelines enn eksemplene vi fant på oppsett.
- Vi satt orginalt opp frontend-prosjektet som et vanlig Angular-prosjekt, men ble anbefalt av kontaktpersonen vår i Communicate å endre det til et ASP.NET-prosjekt med et Angular-oppsett innebygd. Årsaken til dette var at hun tidligere hadde opplevd problematikk med CORS policies med rene Angular-prosjekter i tilknytning til APIM - derfor konverterte vi prosjektet til ASP.NET. Litt senere i utviklingen viste det seg at biblioteket vi trengte å bruke for innlogging med Microsoft-kontoer (tilknyttet Azure AD) ikke trigget korrekt ved bruk av et ASP.NET-prosjekt, slik at vi måtte konvertere tilbake igjen til et rent Angular-prosjekt - hvor vi senere måtte løse ovenfornevnte CORS policy-problematikk.

Disse problemene dukket alle opp i løpet av første utviklingsperiode, og var hovedårsaken til feilestimeringen. Vi balanserte dette ved at to av gruppens medlemmer fortsatte utvikling med fokus på neste utviklingsperiode, mens sistemann jobbet med de utfordringene som hadde dukket opp. Gjennom prosesslæring, gode prioriteringer og evalueringer klarte gruppen å komme seg ajour i løpet av utviklingsperiode to uten videre justeringer.

9.2.2 Ansvarsfordeling

Arbeidsfordelingen har fungert godt gjennom hele prosjektet. Hvert enkelt medlem har oppfylt sine arbeidsoppgaver, og tatt ansvar for egen læring. Gruppen har i stor grad prøvd å fordele oppgaver på en slik måte at arbeidsmengden har blitt jevn, og utover dette har det vært opp til hvert enkelt medlem å arbeide etter eget ønske.

9.3 Teknologier

9.3.1 Hovedrammeverk

Den største problemstillingen som dukket opp tidlig var dokumentasjonen til de ulike rammeverkene vi benyttet, og hva som var nyttig for oss nå. Det viste seg både at Angular og .NET-sfæren er modne rammeverk som har en lang rekke tidligere versjoner, og det samme problemet kan løses på forskjellige måter basert på hvilken versjon man benytter - uten at søk viser det særlig tydelig.

I .NET 5 viste det seg at mye var endret fra tidligere versjoner, og fordi versjonen bare hadde vært tilgjengelig i to måneder ved oppstart av prosjektet - og ikke har planlagt LTS - viste det seg at versjonen var vesentlig dårligere dokumentert enn tidligere versjoner. Gruppen slet samtidig med at dokumentasjon laget for andre versjoner ofte ikke fungerte der det ikke fantes dokumentasjon for gjeldende versjon. Microsoft er heller ikke særlig god på å dokumentere

hvilken versjon dokumentasjonen tilhører, noe som også kan pekes på som en årsak til nevnte problemer. I Angular kom vi i begynnelsen ofte over flere løsninger, hvor noen fungerte og noen ikke. Her var det noe bedre markering av versjoner. Ovennevnte problemer førte til en intern evaluering som resulterte i at oppmerksomheten rundt versjoner måtte bevisstgjøres. Det viste seg at dette tidlig ga større fremdrift i utviklingen, og gjeldende versjon ble derfor videreført.

Til tross for at vi klarte å håndtere problematikken hadde det sannsynligvis vært et bedre valg for gruppen om hadde gått for en eldre versjon av .NET - for eksempel .NET Core 3.1. Dette var dog svært vanskelig å vurdere med kunnskapsgrunnlaget gruppen hadde rundt denne problemstillingen ved oppstart av prosjektet. Eventuelt kunne det vært brukt tid på å konvertere underveis, noe som ikke ble prioritert.

9.3.2 Database

Generelt sett har valget av databasemodell ikke hatt noen store begrensninger for systemet. Hovedproblemstillingen vi møtte på var at Table Storage ligner på en standard SQL-database gjennom at flere verdier ikke kan/bør lagres i samme felt, slik at lagring av arrayer tilhørende et objekt var en liten utfordring. Dette løste vi ved at vi de stedene det ikke var behov for å søke på innholdet i en array, ble innholdet json-serialisert og lagret det som en string. De stedene vi hadde behov for å søke på innholdet ble arrayen lagret til en egen tabell, slik som for eksempel AllocationPeriods.

Likevel utgjorde ikke det noen større problematikk. Det kunne vært enklere å håndtere ved bruk av Azure Cosmos DB som lagrer objekter som json, men ikke nok til at det ville vært verdt den økte kostnaden ved drift.

9.3.3 Bootstrap

Valget vi tok om bruk av Bootstrap er også verdt å diskutere. Bootstrap er på ingen måte et lite rammeverk, og gjør en nettside mye tyngre å laste. Det viste seg at det ikke var et veldig stort antall funksjonaliteter vi trengte fra Bootstrap, så her kunne et bedre valg vært å bruke et rammeverk som kompilerer til å bli mindre - slik som for eksempel TailwindCSS (Tailwind CSS 2021). Motargumentet her er at vi ikke kjente til f.eks. TailwindCSS ved oppstart av prosjektet, men at større tidsbruk på kartlegging kunne i den sammenheng gitt oss et bedre overblikk. Samtidig prioriterte gruppen å komme raskt i gang med utviklingen, og benyttet derfor anbefalinger som ble nevnt i planleggingsfasen. På bakgrunn av dette er vi likevel fornøyd med å ha brukt Bootstrap, fordi det ville vært vanskelig å oppnå et like godt resultat både visuelt og i forhold til brukervennlighet uten å benytte et slikt rammeverk.

9.4 Design

Den viktigste designavgjørelsen vår var å lage applikasjonen vår som tre separate prosjekter - frontend, backend og Function App. Dette var et valg som ga oss utfordringer knyttet til rekkefølgen på oppdateringer i frontend og backend, særlig i starten av prosjektet. Vi opplevde ved enkelte anledninger at det var skrevet kode i enten frontend eller backend som var avhengig av noe som enda ikke var publisert i det andre hovedprosjektet (for eksempel oppdaterte datamodeller), og derfor ga feil på nettsiden. Dette var problematikk som naturlig forbedret seg etterhvert som de viktigste endepunktene var på plass, samtidig som vi også ble mer klar over problematikken og dermed også bedre på kommunikasjonen når nye versjoner ble publisert. Vi vurderte ikke denne problemstillingen til å være særlig problematisk for dette prosjektet siden størrelsen på prosjektgruppen gjorde kommunikasjonen enkel. Derfor ble det heller ikke gjort andre tiltak mot dette. Samtidig er gruppen klar over at dette kunne vært løst gjennom versjonshåndtering i Azure, hvor man definerer hvilken versjon oppdateringer hører sammen med - noe som ville vært særlig aktuelt med en større utviklingsgruppe.

9.4.1 Separation of Concerns

"Separation of Concerns" er et prinsipp som det ble lagt stor vekt på i designløsningen for både API og klient, spesielt med fokus på datatilgang - se kapittel 5.4.2. Dette prinsippet ble også fulgt gjennom designfasen av logiske operasjoner, noe som har gitt en fungerende løsning som vi i ettertid har sett har forbedringspotensial. Gjennom å benytte en Function App i Azure åpner dette for muligheten til å skru av ikke-fungerende funksjoner. Det vil si at vi i vår løsning som inneholder en funksjon, samtidig skrur av all logikk for systemet dersom det viser seg at en operasjon er ikke-fungerende. Dette kunne vært løst gjennom å flytte all funksjonalitet ut i separate funksjoner i Function Appen, - med hver sin klient som gjør dataspørringer til APIet for å gjennomføre kalkulasjoner. Denne formen for arkitektur ville lagt opp til en kombinasjon mellom microservices i Function Appen i tillegg til den nåværende løsningen. Når det er sagt, innehar produktet en løsning der dette ikke skal være noe problem å gjennomføre på et senere tidspunkt da all funksjonalitet er lagt uavhengig av hverandre i APIet. Årsaken til at dette ikke ble prioritert var tidsperspektivet, i form av at forslaget kom i den avsluttende fasen av utviklingsarbeidet.

I tillegg har gruppen i stor grad prøvd å gjøre valideringer av data i både klient og API, slik at ved en eventuell utskiftning vil den gjenværende delen fortsatt regnes som godt sikret.

9.4.2 Feilmeldinger

Feilmeldinger er et tema gruppen også hadde diskusjoner rundt. Her tok gruppen en avgjørelse på å som hovedregel unngå å kaste exceptions, fordi det er

veldig lett å i praksis bruke disse som "else"-utfall av kode. Å definere returverdier ga oss også større kontroll med tanke på hvilke feilkoder og feilmeldinger som ble sendt fra APIet ved feil.

9.5 Kvalitetssikring

Diskusjonen rundt kvalitetssikring har i stor grad handlet om nyttighetsgrad i vårt system. Daglige statusmøter hvor mindre problemstillinger hele veien har blitt diskutert har vært med på å holde arbeidet rettet mot et overordnet mål.

Samtidig har det blitt gjennomført evalueringer av arbeidsprosess, oppståtte problemer og implementert funksjonalitet. Dette har til sammen ført til at gruppen har lært masse av hverandre, samtidig som systemet har blitt forbedret. Nyttighetsgraden vurderes til stor gjennom kvalitetssikring, læring og forbedring på tvers av gruppen.

Gruppens manglende erfaring innenfor automatiserte tester i samspill med problemer tilknyttet tidsestimering av første utviklingsperiode gjorde at automatiserte tester kom inn sent i prosjektet. Allikevel klarte gruppen å disponere ressursene effektivt mot slutten av prosjektet slik at det meste av logikk er dekket gjennom automatisert enhetstesting.

Gruppen har stor tiltro til at den ferdige løsningen oppfyller kravet om brukervennlighet, gjennom kvalitetssikring i form av brukertesting. Vi fikk gode tilbakemeldinger på brukeropplevelsen, samtidig som vi ble gjort oppmerksom på forbedringspotensialer og feil ved systemet som stort sett er utbedret. Totaltsett var dette en læringsrik prosess for gruppen der vi ble utfordret på hvordan vi skulle gjennomføre denne prosessen for å få mest mulig ut av tilbakemeldingene.

9.6 Videre arbeid

Fordi arbeidsgiver har uttrykt et klart ønske om at systemet skal ferdigstilles slik at det kan tas i bruk, har også en del av gruppens arbeid vært å gjøre klart for dette. Det betyr at systemet gjennomgående er godt dokumentert både på wiki-en i DevOps, gjennom API-dokumentasjon i Swagger, systemdokumentasjon gjennom JDocs og XML-dokumentasjon. Selv om vi leverer et fungerende produkt er det likevel arbeid vi ser for oss kan være nyttig å gjøre for videreutvikling. Dette er lagt i backloggen i Kanban-tavlen vi har benyttet, slik at dette skal være enkelt å fortsette på for andre. Oppgaver som ligger der er blant annet:

- Produksjonssette applikasjonen i et reelt produksjonsmiljø, hvor det benyttes App Services som ikke er på gratisnivå slik at nettsiden laster i et fornuftig tempo, hvor reell epostadresse benyttes for utsendelse av bookingbekreftelse osv.
- Bytte ut nøkkelsubstitusjon i pipelines (se kapittel 8) med å benytte

Azure Key Vault for å gjøre nøkler mer fleksible å bytte dersom det skulle skje et sikkerhetsbrudd.

- Bruke mulighetene Azure AD tilbyr bedre for sikring av systemet, for eksempel ved:
  - Å legge til autorisering av kall mot APIM basert på Azure AD (se kapittel 4, slik at for eksempel kun adminer kan gjøre kall knyttet til disse.
  - Å legge til en egendefinert guard i frontend, basert på brukergrupper i AD, slik at vanlige brukere ikke kan navigere til disse sidene dersom de har URLene.
- Forbedre testingen i frontend, med mer automatiserte enhetstester, samt UI-testing der det kunne vært aktuelt.
- Forbedre validering av hva som er obligatorisk data i de ulike datamodellene i backend.
- Legge til ekstra funksjonalitet som f.eks. å la adminer legge inn bookinger på andres vegne.
- Se på hvorvidt det kunne vært nyttig å legge til ARIA-tags (Accessible Rich Internet Applications - brukt for skjermlesere osv.) i HTMLen.
- Vurdere et redesign av utleieobjektsiden, slik at det blir mindre mobilfokusert.
- Gjøre optimaliseringer i frontend med mer caching av data og lat lasting (lazy loading) av f.eks. adminsider en vanlig bruker ikke trenger å laste.
- La brukere og adminer velge hvilke typer hendelser de ønsker epostvarslinger om.
- Når lønnsavdelingen er klar for det - integrere løsningen mot Vipps slik at bookinger kan betales direkte.

Alle disse tingene er funksjonalitet som ble nedprioritert under utvikling fordi de ikke direkte bidro til et fungerende produkt, men er forbedringspotensiale. Dette var som tidligere nevnt en del av avtalen med oppdragsgiver hvor vi utviklet så mye vi fikk tid til, og resten ble lagt godt beskrevet i backloggen vår til videreutvikling av selskapet på et senere tidspunkt.

Kapittel 10

Konklusjon

Målet med oppgaven var å utvikle et system for internutleie av Communicates feriesteder. Vi har utviklet en web-applikasjon hvor Angular benyttes til frontend, og .NET benyttes til backend. En rekke relevante Azure-tjenester er benyttet for lagring, produksjonssetting, og sikring av tjenesten, samt som utviklingsmiljø. Communicates ønske var en tjeneste som automatiserte mye av det administrative arbeidet rundt internutleien, samtidig som systemet for fordeling basert på internt regelverk skulle beholdes. Det var også sentralt at tjenesten skulle ha en god brukeropplevelse, da den eksisterende løsningen var svært utdatert. I tillegg er læringsmålene som en del av bacheloroppgavefaget naturlig nok et sentralt aspekt ved oppgaven. Gruppen har hovedsaklig benyttet Kanban som utviklingsmetode, med elementer fra både Scrum og Extreme Programming.

Applikasjonen automatiserer de manuelle arbeidsoppgavene knyttet til nåværende løsning. Som et direkte resultat vil mye arbeidstid kunne bespares når systemet tas i bruk. Applikasjonen følger også kjente designprinsipper for god brukeropplevelse. Ved brukertesting fikk vi gode tilbakemeldinger knyttet til brukeropplevelsen, samt en del ønsker og forbedringspotensiale. Vi har innfridd de aller fleste av disse tilbakemeldingene, og har kun utelatt de som ville tatt uforholdsmessig mye tid med tanke på nyttigheten sett i lys av gjenværende tid av prosjektet.

Gruppen har på eget initiativ også lagt vekt på å utvikle et robust system på ulike måter. Blant annet har vi gjennomgående forsøkt å forholde oss til "Separation of Conerns"-prinsippet ved å gjøre de ulike delene av systemet uavhengige av hverandre. Dette er gjort både ved å sikre at det er lett å bytte alle deler av systemet uten store kodeendringer, og ved å separere funksjonalitet i ulike klasser med egne fokusområder. Applikasjonen er også beskyttet mot de vanligste sikkerhetstruslene, og lagrer så lite persondata som mulig.

Alle gruppens medlemmer har tilegnet seg ny kunnskap innen valgt utviklingsmodell og innenfor nye teknologier. Gjennom god planlegging, strukturering og kvalitetssikring har vi utviklet en applikasjon som i stor grad er klar til bruk. Utfordringer underveis har blitt løst eller prioritert på en måte som har gjort at vi kom i mål. Ingen av gruppens medlemmer hadde noen form for erfaring med rammeverkene vi har benyttet ved oppstart av prosjektet, mens vi ved avslutning har god kjennskap til både Angular, .NET, Azure som de viktigste teknologiene.

Andre faktorer gruppen vil trekke frem som avgjørende for sluttresultatet

er ansvar for egen læringsprosess, som igjen har ført til at hver enkelt har tatt ansvar for sluttproduktet.

Tjenesten vi leverer er en helhetlig applikasjon som oppnår oppgavens mål, og som har fått gode og positive tilbakemeldinger fra både oppdragsgiver og brukertestere.

Bibliografi

Acondy, K. (2020), 'Angular Routing Explained'. Last accessed 11.05.2021.
URL: *https://www.newline.co/@krishna/angular-routing-explained--3541df8d*

AltexSoft (2020), 'The Good and the Bad of Angular Development'. Last accessed 19.04.2021.
URL: *https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/*

Angular (2021*a*), 'Reactive forms'. Last accessed 12.05.2021.
URL: *https://angular.io/guide/reactive-forms*

Angular (2021*b*), 'Security'. Last accessed 07.05.2021.
URL: *https://angular.io/guide/security*

Bauknecht, P. (2020), 'Authenticating Angular apps with Azure Active Directory using MSAL Angular 1.0'. Last accessed 11.05.2021.
URL: *https://medium.com/medialesson/authenticating-angular-apps-with-azure-active-directory-using-msal-angular-1-0-d7b2f4914be9*

Bootstrap (2021), 'Bootstrap - Introduction'. Last accessed 21.04.2021.
URL: *https://getbootstrap.com/docs/4.6/getting-started/introduction/*

Caballero, C. (2019), 'Understanding MVC Services for Front End: Angular'. Last accessed 03.03.2021.
URL: *https://betterprogramming.pub/https-medium-com-ccaballero-understanding-mvc-services-for-front-end-angular-a6196492ee74*

Chiaretta, S. (2017), 'Omnisharp: the tooling behind the C# integration in VS Code'. Last accessed 11.05.2021.
URL: *https://codeclimber.net.nz/archive/2017/12/18/omnisharp-the-tooling-behind-the-c-integration-in-vs-code-day-18-24-days-of-front-end-development-with-aspnet-core-angular-and-bootstrap/*

Datatilsynet (2019*a*), 'Om personopplysningsloven med forordning og når den gjelder'. Last accessed 07.04.2021.
URL: *https://www.datatilsynet.no/regelverk-og-verktoy/lover-og-regler/om-personopplysningsloven-og-nar-den-gjelder/*

Datatilsynet (2019*b*), 'Personvernprinsippene'. Last accessed 07.04.2021.
URL: *https://www.datatilsynet.no/rettigheter-og-plikter/personvernprinsippene/*

Hornbech, J. (2021), 'Support Tip: Get started with Microsoft Graph'. Last accessed 30.04.2021.
URL: *https://techcommunity.microsoft.com/t5/intune-customer-success/support-tip-getting-started-with-microsoft-graph-api/ba-p/364257*

Joshi, A. (2016), 'Comparison between Angular and Polymer'. Last accessed 21.04.2021.
URL: *https://www.tothenew.com/blog/comparison-between-angular-and-polymer/*

Kanbanize (2020), 'What Are Kanban Swimlanes and How to Use Them in Practice?'. Last accessed 29.01.2021.
URL: *https://kanbanize.com/kanban-resources/kanban-software/kanban-swimlanes*

Katkov, Y. (2019), 'How To Create a Caching Service for Angular'. Last accessed 12.05.2021.
URL: *https://betterprogramming.pub/how-to-create-a-caching-service-for-angular-bfad6cbe82b0*

Kudchikar, S. (2019), 'Repository Pattern C#'. Last accessed 26.02.2021.
URL: *https://codewithshadman.com/repository-pattern-csharp/*

Kumar, V. (2020), '5 Key Features of AngularJS that make it the best for Web Development'. Last accessed 03.03.2021.
URL: *https://www.matridtech.net/5-key-features-of-angularjs-that-make-it-the-best-for-web-development/*

Lander, R. (2019), 'Introducing .NET 5'. Last accessed 19.04.2021.
URL: *https://devblogs.microsoft.com/dotnet/introducing-net-5/*

Leung, W. (2019), 'Azure DevOps Pipeline — Choosing Between YAML and Classic UI'. Last accessed 09.05.2021.
URL: *https://medium.com/@wywywywy/azure-devops-pipeline-choosing-between-yaml-and-classic-ui-b5612c3e211a*

Maddox, I. (2021), '13 best practices for user account, authentication, and password management, 2021 edition'. Last accessed 14.05.2021.
URL: *https://cloud.google.com/blog/products/identity-security/account-authentication-and-password-management-best-practices*

Maiti, A. (2021), 'Understanding AddTransient Vs AddScoped Vs AddSingleton In ASP.NET Core'. Last accessed 04.05.2021.
URL: *https://www.c-sharpcorner.com/article/understanding-addtransient-vs-addscoped-vs-addsingleton-in-asp-net-core/*

Malan, R. & Bredemeyer, D. (2001), 'Functional Requirements and Use Cases'.
URL: *http://www.bredemeyer.com/pdf_files/functreq.pdf*

Material Design (2017*a*), 'Components–Cards'. Last accessed 17.02.2021.
URL: *https://material.io/archive/guidelines/components/cards.html*

Material Design (2017*b*), 'Patterns - Navigation drawer'. Last accessed 04.05.2021.
URL: *https://material.io/archive/guidelines/patterns/navigation-drawer.html*

MDN Web Docs (2021), 'Understanding the Web Content Accessibility Guidelines'. Last accessed 05.05.2021.
URL: *https://developer.mozilla.org/en-US/docs/Web/Accessibility/Understanding_WCAG*

Microsoft (2015), 'C# Coding Conventions (C# Programming Guide)'. Last accessed 29.01.2021.
URL: *https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions*

Microsoft (2017*a*), 'About API Management'. Last accessed 19.04.2021.
URL: *https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts*

Microsoft (2017*b*), 'Language-Integrated Query (LINQ) (C#)'. Last accessed 07.05.2021.
URL: *https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/*

Microsoft (2018), 'Unit testing best practices with .NET Core and .NET Standard'. Last accessed 07.05.2021.
URL: *https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-best-practices*

Microsoft (2019), 'Get started guide for Azure developers'. Last accessed 19.04.2021.
URL: *https://docs.microsoft.com/en-us/azure/guides/developer/azure-developer-guide*

Microsoft (2020*a*), 'Common web apllication architectures'. Last accessed 25.01.2021.
URL: *https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures*

Microsoft (2020*b*), 'Introduction to Azure Blob storage'. Last accessed 22.02.2021.
URL: *https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction*

Microsoft (2020*c*), 'Introduction to Azure Functions'. Last accessed 14.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview*

Microsoft (2020*d*), 'Introduction to Identity on ASP.NET Core'. Last accessed 07.05.2021.
URL: *https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio*

Microsoft (2020*e*), 'Microsoft identity platform and OpenID Connect protocol'. Last accessed 08.02.2021.
URL: *https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-protocols-oidc*

Microsoft (2020*f*), 'Microsoft Identity Platform ID-tokens'. Last accessed 08.02.2021.
URL: *https://docs.microsoft.com/en-us/azure/active-directory/develop/id-tokens*

Microsoft (2020*g*), 'MSAL Angular Sample Application'. Last accessed 18.05.2021.
URL: *https://docs.microsoft.com/en-us/samples/azure-samples/active-directory-javascript-singlepageapp-angular/active-directory-javascript-singlepageapp-angular/*

Microsoft (2020*h*), 'Timer trigger for Azure Functions'. Last accessed 07.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer?tabs=csharp*

Microsoft (2020*i*), 'Tutorial: Sign in users and call the Microsoft Graph API from a JavaScript single-page app (SPA) using auth code flow'. Last accessed 26.02.2021.
URL: *https://docs.microsoft.com/en-gb/azure/active-directory/develop/tutorial-v2-javascript-auth-code*

Microsoft (2020*j*), 'What is Azure Active Directory?'. Last accessed 19.04.2021.
URL: *https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis*

Microsoft (2021*a*), 'Get access without a user'. Last accessed 30.04.2021.
URL: *https://docs.microsoft.com/en-us/graph/auth-v2-service*

Microsoft (2021*b*), 'Introduction to Azure Cosmos DB'. Last accessed 03.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/cosmos-db/introduction*

Microsoft (2021*c*), 'Introduction to the core Azure Storage services'. Last accessed 20.04.2021.
URL: *https://docs.microsoft.com/en-us/azure/storage/common/storage-introduction*

Microsoft (2021*d*), 'Mailkit'. Last accessed 30.04.2021.
URL: *https://github.com/jstedfast/MailKit*

Microsoft (2021*e*), 'Overview over Microsoft Graph'. Last accessed 30.04.2021.
URL: *https://docs.microsoft.com/en-us/graph/overview*

Microsoft (2021*f*), 'Plan and manage costs in Azure SQL database'. Last accessed 03.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/azure-sql/database/cost-management*

Microsoft (2021*g*), 'Public Holiday'. Last accessed 03.05.2021.
URL: *https://github.com/martinjw/Holiday*

Microsoft (2021*h*), 'Q63.When Azure offers SQL Database why should I use Windows Azure Table Storage in Azure application?'. Last accessed 03.05.2021.
URL: *https://vkinfotek.com/azureqa/why-should-i-use-windows-azure-table-storage.html*

Microsoft (2021*i*), 'SmtpClient Class'. Last accessed 30.04.2021.
URL: *https://docs.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient?view=net-5.0*

Microsoft (2021*j*), 'Storage account overview'. Last accessed 26.02.2021.
URL: *https://docs.microsoft.com/en-us/azure/storage/common/storage-account-overview*

Microsoft (2021*k*), 'Tutorial: Add sign-in to Microsoft to an ASP.NET web app'. Last accessed 03.02.2021.
URL: *https://docs.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-asp-webapp*

Microsoft (2021*l*), 'Understanding the differences between NoSQL and relational databases'. Last accessed 03.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/cosmos-db/relational-nosql*

Microsoft (2021*m*), 'What is Azure DevOps?'. Last accessed 19.04.2021.
URL: *https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops*

Microsoft (2021*n*), 'What is Azure Logic Apps?'. Last accessed 14.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview*

Microsoft (2021*o*), 'What is the Azure SQL database'. Last accessed 03.05.2021.
URL: *https://docs.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview*

NagerDate (2021), 'Nager.Date - Official Website'. Last accessed 30.04.2021.
URL: *https://github.com/nager/Nager.Date*

npm Docs (2021*a*), 'Auditing package dependencies for security vulnerabilities'. Last accessed 07.05.2021.
URL: *https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities*

npm Docs (2021*b*), 'npm-audit: Run a security audit'. Last accessed 11.05.2021.
URL: *https://docs.npmjs.com/cli/v7/commands/npm-audit*

NTNU (2021), 'BIDAT39 - Bacheloroppgave - Dataingeniør'. Last accessed 19.01.2021.
URL: *https://www.ntnu.no/studier/emner/BIDAT39*

OAuth 2.0 (2021), 'OAuth 2.0'. Last accessed 07.05.2021.
URL: *https://oauth.net/2/*

OpenJS Foundation (2017), 'ESLint - About'. Last accessed 11.05.2021.
URL: *https://eslint.org/docs/about/*

OWASP (2020), 'OWASP Top Ten'. Last accessed 07.05.2021.
URL: *https://owasp.org/www-project-top-ten/*

OWASP (2021), 'Authentication Cheat Sheet'. Last accessed 14.05.2021.
URL: *https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html*

Pooroni, J. (2021), 'How to use a Web API to Send Email'. Last accessed 30.04.2021.
URL: *https://www.socketlabs.com/blog/web-api-send-email/*

Radigan, D. (2014), 'What is kanban?'. Last accessed 22.01.2021.
URL: *https://www.scrumguides.org/scrum-guide.html*

Rajendran, P. (2020), 'Azure API App vs Web App'. Last accessed 19.04.2021.
URL: *https://www.serverless360.com/blog/azure-api-app-vs-web-app*

Ravichandran, R. (2021), 'Improving comprehension through intuitive actions'. Last accessed 05.05.2021.
URL: *https://medium.com/google-design/improving-comprehension-through-intuitive-actions-f7e6336e12e6*

Richards, M. (2015), *Software Architecture Patterns*, O'Reilly Media.

Rick Anderson, K. L. (2021), 'Enable Cross-Origin Requests (CORS) in ASP .NET'. Last accessed 07.05.2021.
URL: *https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-5.0*

Schwaber, K. & Sutherland, J. (2020), 'The Scrum Guide'. Last accessed 22.01.2021.
URL: *https://www.scrumguides.org/scrum-guide.html*

Shet, P. (2017), 'Writing Complex Queries Using LINQ And Lambda'. Last accessed 07.05.2021.
URL: *https://www.c-sharpcorner.com/article/writing-complex-queries-using-linq-and-lambda/*

Smith, I. (2020), 'Repository Pattern'. Last accessed 01.03.2021.
URL: *https://deviq.com/design-patterns/repository-pattern*

Sommerville, I. (2004), *Software Engineering*, 10 edn, Pearson Education.

Swagger (2021), 'About Swagger'. Last accessed 04.05.2021.
URL: *https://swagger.io/about/*

Tailwind CSS (2021), 'Tailwind CSS'. Last accessed 15.05.2021.
URL: *https://tailwindcss.com/*

W3Schools (2019), 'Sass Introduction'. Last accessed 21.04.2021.
URL: *https://www.w3schools.com/sass/sass_intro.asp*

Warren, M. (2018), 'A History of .NET Runtimes'. Last accessed 19.04.2021.
URL: *https://mattwarren.org/2018/10/02/A-History-of-.NET-Runtimes/*

Wells, D. (1999), 'When should Extreme Programming be Used?'. Last accessed 22.01.2021.
URL: *http://www.extremeprogramming.org/when.html*

Vedlegg A

Oppgavebeskrivelse

Oppdragsgiver: Communicate Norge AS
Vi har jobbet med integrasjon – selve bindevevet innen digitalisering – siden 1996 og er Norges største ekspert innen området. Forskjellen fra den gang er at teknologien nå gir uendelig flere muligheter. Og hastigheten på utviklingen er enorm. I dag leverer vi tjenester for å gi kundene forretningsnytte i en ny digital verden. Digitale plattformer, strategisk rådgivning, virksomhets- og løsningsarkitektur, integrasjonsutvikling og skytjenester er stikkord.

Adresse: Communicate Norge AS, Torget 5, 1767 Halden.

Kontaktperson/Veileder på oppgaven: Kaya Masterød Karlsen, Telefon: [...], Mail: [...]

Oppgave: Communicate utleie sider (et fullstack prosjekt med både backend og front-end)

Oppgaven går ut på å utvikle en tjeneste og eksponere den gjennom en nettside. På den måten er det ganske kort vei til et resultat som kan vises frem samtidig som det kan legges til nesten ubegrenset med nye features som kan implementeres dersom de får tid. Alt skal selvfølgelig gjøres i Azure og med .NET, samtidig som de skal benytte seg av Azure DevOps til planlegging, kildekontroll og CI/CD (Continious Integration/Continious Development). Fokuset er å gjøre oppgaven så lik som en vanlig arbeidsdag for utviklere i Communicate.

Oppgavens mål: Utvikle et system som er demonstrerbart for oss som jobber i Communicate hvor vi kan booke hytter og leiligheter som vi kan benytte i fritiden vår.
Teknologi: .NET, C# og skytjenesten AZURE , VisualStudio/Code, Azure DevOps, samt andre relevante Microsoft løsninger og tjenester.

Vedlegg B

Prosjektavtale

NTNU

Norges teknisk-naturvitenskapelige universitet

Vår dato Vår referanse

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

COMMUNICATE NORGE AS

(oppdragsgiver), og

June V. E. Hansen

Sindre Opskar Hjelle

Leif Torbjørn Næsvold (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11/01-21 til 20/05-21 .

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bestått bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Rune Hjelsvold

Oppdragsgivers kontaktperson (navn): KAYA M. KARLSEN

Student(er) (signatur): June Hansen dato 22/01-21

Sindre O. Hjelle dato 23/01-21

dato 23/01-21

dato

Oppdragsgiver (signatur): Kaya M. Karlsen dato 20/01-21

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): dato

Vedlegg C

Prosjektplan

C.1 Mål og Rammer

C.1.1 Bakgrunn

Communicate Norge er et selskap som jobber med integrasjon mellom ulike plattformer. Selskapet ble startet opp i 1996, og har kontorer i Oslo, Halden og Stavanger. Selskapet har i dag 95 ansatte, som har mulighet til å disponere tre ulike feriesteder gjennom selskapet. Alle feriestedene er tilgjengelige hele året.

Dagens løsning for utleie av Communicate Norge sine feriesteder er i stor grad utdatert både administrativt og visuelt. For å kunne benytte et av feriestedene i et gitt tidsrom må man i dag sende en mail til administrasjonen som behandler disse forespørslene manuelt. Communicate ønsker seg et system som er så automatisert som mulig - alt fra prosessen å booke en hytte/leilighet for en ansatt, til tildeling av bruk også i ferier hvor flere har ønske om å benytte de. Brukervennlighet, samt automatisering av arbeidsoppgaver som i dag gjøres manuelt, står i sentrum. Designet til siden er veldig simpelt satt opp med mange sider å navigere med mye repetisjon av tekst på flere sider.

C.1.2 Prosjektmål

Målet med prosjektet kan i hovedsak deles inn i tre ulike kategorier; *resultatmål*, *effektmål* og *læringsmål*. Resultatmål og effektmål er selvsagte måltyper her, men læringsmål er i tillegg tatt med fordi Communicate har uttrykt et ønske om at oppgaven skal benyttes til læring for prosjektgruppen. I tillegg er det nevnt i oppgavebeskrivelsen at utviklingsprosessen, fra prosjektplanlegging til ferdig resultat, skal legges opp så likt en vanlig arbeidsuke for et profesjonelt utviklingsprosjekt som mulig.

Resultatmål

Målet med prosjektet er å lage en webløsning for utleie av selskapets feriesteder. Løsningen skal som et minimum fungere på PC/Tablet, og helst utvides slik at den også fungerer fra mobil-enheter. Videre funksjonalitet implementeres så langt det lar seg gjøre etter en liste oppsatt prioritert etter funksjonalitet.

- Å fremstille informasjon på en ryddig og oversiktlig måte.
- Å automatisere prosessen med utleie slik at administrative kostnader knyttet til tilbudet minimeres.

- At løsningen skal utvikles og dokumenteres på en slik måte at andre utviklere senere kan gå inn å vedlikeholde eller videreutvikle den.

Effektmål

Oppgaven er gitt på bakgrunn av at et av gruppemedlemmene har fått jobb hos oppdragsgiver, hovedformålet med oppgaven er derfor at vedkommende skal bli mest mulig kjent med verktøyene som oppdragsgiver normalt bruker.Selve prosjektet skal erstatte en eksisterende løsning som er lite oversiktlig, svært utdatert, og som krever uforholdsmessig store ressurser administrativt. Den nye løsningen har derfor som mål:

- Gi fremtidig ansatt erfaring fra verktøyene som oppdragsgiver benytter
- Å redusere driftskostnader knyttet til dagens løsning.
- Å redusere administrative kostnader i form av at ansatte kan bruke ressursene sine på verdiskapning for bedriften eksternt.

Læringsmål

Bacheloroppgaven har overordnede læringsmål som følge av emnebeskrivelsen til bacheloroppgavefaget (NTNU 2021), men Communicate har også uttrykt at de ønsker at oppgaven skal benyttes for at bachelorgruppen skal lære, blant annet å bli kjent med .NET-rammeverket og Azure. I tillegg har gruppen selv noen mål om ønsket læring. Derfor er følgende læringsmål også inkludert:

Fra NTNU

Etter gjennomført bacheloroppgave har studenten tilegnet seg:

- ny kunnskap innen en selvvalgt del av sitt fagområde.
- forståelse for metodisk arbeid, evne til refleksjon og evne til systematisk/vitenskapelig vurdering.
- kompetanse til å planlegge og utføre en selvstendig oppgave, formulere problemstillinger og analysere disse med utgangspunkt i både teoretisk og empirisk materiale og å gjennomføre en oppgave på en metodisk tilfredsstillende måte.
- ferdigheter i å utarbeide konkrete problemstillinger av samfunnsmessige interesser innen fagområdet, under veiledning.
- ferdigheter i å identifisere og vurdere litteratur som er relevant for problemstillingen, under veiledning.
- ferdigheter i å gå i dybden på avgrensede problemstillinger og utarbeide konkrete løsningsalternativer på problemet.
- ferdigheter i å dokumentere og formidle resultatene fra prosjektarbeidet på en systematisk/vitenskapelig måte.

(NTNU 2021)

Fra Communicate

- Bli godt kjent med .NET, Azure og valgfri (Angular) frontend-plattform.
- Tilegne seg kunnskap underveis god nok til å kunne velge en god løsninger for sluttproduktet.
- Bli kjent med problemer som kan dukke opp i en utviklingsprosess, og finne løsninger som kan hjelpe oss å bli mer egnet for slike prosjekt arbeid i senere tid.

Fra bachelorgruppen

- Tilegne seg praktisk erfaring innen bruk av en valgt utviklingsmodell.
- Lære seg å avgrense valgt løsning til mindre oppgaver og utarbeide ulike konkrete løsningsalternativer og løsninger.
- Få bedre kjennskap til utviklingen av en progressiv web-applikasjon, samt skape et godt brukergrensesnitt.
- Få bedre kjennskap til kvalitetsikring og testing.

De viktigste av disse læringsmålene med tanke på formålet til oppgaven er selvsagt målene fra NTNU, da disse utgjør grunnlaget for karakteren til oppgaven. Disse er likevel forholdsvis generelle, og gir rom for konkretisering. For eksempel er å bli kjent med .NET, Azure og Angular en spesifisering av å skaffe ny kunnskap innen en sevlvalgt del av sitt fagområde. Det samme gjelder å lære seg å avgrense valgt løsning til mindre oppgaver, som overlapper med NTNUs mål om å gå i dybden på avgrensede problemstillinger og utarbeide konkrete løsningsalternativer. På denne måten er målene utover NTNUs læringsmål i stor grad utdypninger av disse.

C.1.3 Rammer

Det er to ulike organisasjoner som er interessenter for oppgaven, og som legger førende rammer for prosjektet. NTNU har lagt en tidsramme for når dokumentasjonen skal være ferdigstilt. I tillegg til at oppdragsgiver, Communicate Norge, har lagt føringer for mange av de tekniske rammene gitt ved prosjektet.

Gitt av NTNU

- Oppgaven skal utarbeides mellom prosjektoppstart 11.01.2021 og 20.05.2021, med enkelte innleveringer underveis.
- Prosjektoppgaven forhåndsgodkjennes av ansvarlige for bacheloroppgavefaget, og det gis rettledning av intern veileder gjennom forløpet av prosjektarbeidet.
- Det skal utarbeides en rapport som omhandler utviklingen, med særlig fokus på valg gjort underveis.

Gitt av Communicate

- Løsningen skal benytte .NET, samt et valgfritt frontend-rammeverk (Angular).
- Tjenesten skal være så kostnadseffektiv som mulig uten at det går utover webløsningens tilgjengelighet.
- Tjenesten skal så langt som hensiktsmessig mulig automatisere administrasjonsoppgaver knyttet til utleien av feriestedene.
- Tjenesten skal benytte CI/CD for deployment til et developer-miljø, og skal kjøre i Azure.

Bachelorgruppen fikk selv velge frontend-rammeverk, men ble anbefalt Angular (11.0.7) med TypeScript (4.0.5) på bakgrunn av muligheten for å støtte seg på selskapets kompetanse ved behov. Ingen av gruppemedlemmene har erfaring fra dette rammeverket, og videre diskusjonen rundt dette temaet i hovedrapporten. Det er verdt å merke seg at tekniske bestemmelser skal skje før kravspesifiseringsprosessen begynner.

C.2 Omfang

C.2.1 Fagfelt/Problemområdet

Løsningen som skal utvikles er i utgangspunktet en klassisk webløsning, hvor det skal kunne gjennomføres utleie av feriesteder. Applikasjonen er likevel noe annerledes i forhold til en "vanlig"utleieside, da det er snakk om internutleie innad i en bedrift - ikke en løsning som skal være allment tilgjengelig. Det betyr for eksempel at kun ansatte skal ha tilgang til å booke feriesteder, som naturlig kan løses ved innlogging begrenset til microsoft-kontoer tilhørende Communicate. Det er heller ikke helt first come, first serve", da tildeling av hytter skjer på fastsatte datoer etter et poengsystem som rettferdiggjør tildelingsprosessen - særlig i ferier eller ved dobbelbooking. De ansatte betaler heller ikke normalt for et opphold, men blir trukket en leieutgift fra neste lønning. Alt dette gjør at prosjektet må spesialutvikles, fremfor å bruke eksisterende løsninger.

For utviklingen kommer en rekke teknologier og konsepter til å benyttes. Disse vil blant annet være:

- Bruk av rammeverket Angular, med HTML, CSS og Typescript for webapplikasjonen.
- .NET med REST API endepunkter benyttes til backend.
- Hentsiktsmessige databaser benyttes - sannsynlig både No-SQL og SQL.
- Gjennomgående unittesting, samt brukertester for tilbakemeldinger underveis.

C.2.2 Avgrensning

Oppdragsgiver har allerede en eksisterende løsning som i stor grad benytter e-post-tjenester for booking. Dette innebærer at en person hele tiden er ansvarlig for manuelt å oppdatere kalenderen for når feriesteder er ledig, samt tildele disse i fellesferier. Oppdragsgiver har gitt oss valget om å jobbe utifra eksisterende webløsning, eller å begynne helt på nytt. Vi vurderer dagens løsning til å være lite oversiktlig. Kombinert med at den tilbyr et fåtall av tjenestene det endelige systemet skal inneha, har vi besluttet at vi ikke ønsker å benytte den eksisterende systemløsningen som et utgangspunkt i vår utvikling.

Vi viderefører derimot bedriftens arbeidsflyt, i form av at eksisterende poengsystem for rettferdiggjøring av utleie benyttes, det legges opp til samme type frister for søknad om utleie (se vedlegg F), informasjonen fra den eksisterende siden kommer i stor grad til å overføres direkte osv. Årsaken til at vi velger denne løsningen er at vi tror at de administrative kostnadene knyttet til for store endringer for allerede etablerte frister fort kan bli store. Spesielt med tanke på at gruppen ikke ser noen grunn til at en videreføring av arbeidsflyten skal påvirke resultatmålene satt for systemet.

C.2.3 Prosjektbeskrivelse

Oppdragsgiver har gitt føringer for hvilke funksjonaliteter de ønsker, der spesielt utleiereglement, aktiviteter, veibeskrivelse, og annen informasjon som finnes på den gamle løsningen skal videreføres. Samtidig er det blitt lagt vekt på at vi står ganske fritt når det kommer til den grafiske tilnærmingen til webapplikasjonen. Under følger en liste over funksjonaliteter som på dette stadiet blir sett på som et minimum som skal implementeres:

- Brukeren skal enkelt kunne få oversikt over hvilke utleieobjekter som finnes, utleieinformasjon og -regler knyttet til disse.
- Brukeren skal kunne legge inn, endre og slette sine egne bookinger på et utleieobjekt.
- Brukeren får tilgang til systemet gjennom sin Microsoft-bruker som allerede er knyttet opp mot Communicate.
- En bruker kan tildeles administrator-rettigheter, som gjør det mulig å gå inn og gjøre endringer på alle bookinger

I tillegg er følgende funksjonalitet høyt prioritert:

- Systemet skal holde oversikt på ferier/helligdager, og tildeling skjer automatisk - basert på interne bestemmelser i forhold til et poengsystem - i disse periodene.
- Det ønskes en administratorside hvor det kan gjøres endringer av innholdet på siden uten å involvere en utvikler, slik at tildelingsperioder og frister kan endres, man kan se rapporter, fjerne bookinger, få oversikt over poeng osv.

Videre krav til løsningen vil defineres i kravspesifikasjonsprosessen.

C.3 Organisering

C.3.1 Ansvarsfordeling

- June
  - Prosjektleder - valgt av gruppen, og var et ganske naturlig valg med tanke på hennes tilknytning til oppgaven og Communicates kontaktperson, samt utdannelse innen administrasjon og ledelse.
  - Fleksibel mellom arbeid frontend og backend, pga. ønske om å lære mest mulig. Gir også ekstra sikkerhet dersom en av de to andre skulle falle ut av prosjektet underveis.
- Sindre
  - Utvikler frontend - Hovedansvar for frontend med bakgrunn i tidligere erfaring med webutvikling.
  - Dokumentansvarlig - Selvpåtatt ansvar for at dokumenter leveres når de skal, inneholder det de skal osv.
- Torbjørn
  - Utvikler backend - Hovedansvar for backend med bakgrunn i tidligere utviklingserfaring.
  - Referent - Tok naturlig rollen fra første møte, og ble derfor senere valgt til referent.

Dette vil være er en fleksibel rollefordeling som kan endres dersom enkelte ønsker å jobbe med andre ting en periode, eller et gruppemedlem av ulike årsaker faller bort. Hvert gruppemedlem har også mulighet til å jobbe med ulike deler av prosjektet etter ønske, slik at det er mulighet for læring og selvutvikling.

Ved behov vil ansvar for mer spesifikke ting som f.eks. å undersøke en spesiell teknologi, sikkerhet eller lignende bli definert. Dette gjøres fleksibelt underveis, og ikke før kravspesifikasjoner og andre steg i planleggingsfasen er ferdigstilt.

C.3.2 Rutiner og regler i gruppen

Grupperegler ligger som helhet vedlagt, noen viktige er:

- 30 timer arbeid per person i uka.
- Korte fellesmøter man-fre kl. 11:00.
- Møter og fellesarbeid vil som hovedregel foregå digitalt, med mulighet for å jobbe på campus om muligheten oppstår.
- Timer loggføres i Google Regneark.
- Møteplikt, dersom ikke annet er avtalt på forhånd.
- Konstruktiv kritikk ønskes velkommen.

- Avgjørelser som påvirker sluttproduktet fattes i flertall.

C.4 Planlegging, oppfølging og rapportering

C.4.1 Prosessrammeverk

Valg av rammeverk

Det er noen hovedelementer som klart legger føringer for utviklingsprosessen uavhengig av utviklingsmodell:

- Kort utviklingsperiode på bakgrunn av prosjektperioden til bacheloroppgaven.
- Brukervennlighet og automatisering er de viktigste egenskapene ved sluttproduktet.
- Situasjonen med Koronaviruset (Covid-19) gjør at utviklingsteamet er lokalisert på ulike steder, og har begrensede og usikre muligheter for å møtes.
- Produkteier anbefaler på det sterkeste at vi samler så mye som mulig i prosjektportalen i Azure - både repositories, bruk av Boards, Pipelines ol.
- Ukentlige statusmøter med produkteier.
- Ukentlige diskusjonsmøter med intern veileder.
- Daglige lunsjmøter innad i utviklingsteamet.

I tillegg vil vi som en del av kravspesifikasjonsprosessen sette opp en prioriteringsliste over funksjonalitet. Etter ønske fra produkteier vil dette bli plassert i en Kanban-tavle i Azure Boards slik at produkteier også har tilgang til hvordan gruppen ligger an i prosessen. Ettersom vi kan få ønsker om ny funksjonalitet eller forslag til endringer underveis, vil en smidig metodikk være enklest å forholde seg til.

Gruppen vurderte de største smidige utviklingsmodellene. Extreme Programming (XP) virket lite aktuelt da det er mye fokus på parprogrammering, og samlokalisering som vanskeliggjøres pga. Covid-19. XP har også hovedfokus på utvikling hvor spesifikasjonene endrer seg hyppig - noe som ikke er tilfellet her, selv om vi forventer noen endringer underveis (Wells 1999).

Både Scrum, iterativ utvikling og Kanban blir sett på som svært gode alternativer for prosjektet. Vi ser for oss en utviklingprosess der kontinuerlig forbedring står sentralt for hele tiden å innhente tilbakemelding fra brukerne. Gruppen ønsker også å være sikre på at vi får en kontinuerlig arbeidsflyt, selv om vi er fullstendig klar over at tidsestimater med såpass lite erfaring som vi har også kommer med risiko (mer om det i kapittelet om risikohåndtering). Dette er sentrale deler av begge modellene (Sommerville 2004).

Gruppen ønsker korte daglige møter der tidsestimat for utviklingsperioden kan vurderes. Spørsmål knyttet til utviklingen, og andre avgjørelser som påvirker prosjektet kan tas opp. Dette tilsvarer de daglige statusmøtene som er

vanlig i scrum (Sommerville 2004). I tillegg ønsker vi å starte hver utviklingsperiode med planlegging, der tidsestimering vil stå sentralt. I etterkant ønsker vi å evaluere utviklingsperioden. Dette for å sikre at gruppen er samstemt gjennom hele prosessen, samt holde kontroll på hvordan gruppen ligger an underveis i både utviklingsperiodene og prosjektet (Schwaber & Sutherland 2020).

Produkteier oppfordret gruppen til å holde oversikt over oppgaver i Azure Boards som er en Kanban-tavle. Dermed blir Kanban en naturlig del av utviklingsprosessen. Kanban gir også et fokus på å hele tiden gjennomføre påbegynte oppgaver, samt fokus på å alltid ta øverst prioritert oppgave når man starter på noe nytt. På denne måten vil alltid den neste viktigste oppgaven bli tatt hånd om og gjennomført, og gruppen kan unngå tidstyvene som ofte oppstår i forbindelse med multitasking (Radigan 2014).

Kontaktpersonen i selskapet er ikke den samme personen som har hovedansvar for dagens løsning. Vi synes derfor ikke det er naturlig å kalle denne produkteier.

En iterativ utviklingsprosess kunne også vært en god modell for vårt prosjekt. Manglende erfaring både innen tidsestimater, og utviklingsrammeverk gjør derimot at vi ønsker korte daglige statusmøter for å minimere tidsbruk på administrative problemer knyttet til rammeverket. Det er samtidig vanskelig å si at modellen vi benytter vil ha svært ulike elementer sammenlignet med iterativ utvikling.

Modellens anvendelse i prosjektet

På bakgrunn av overnevnte diskusjon har vi valgt elementer som vi mener er fordelaktige for hvordan vi på best mulig måte kan styre prosjektperioden. Dette innebærer at gruppen vil benytte en Kanban-tavle med oversikt over hvilke oppgaver som skal gjøres og hvor i prosessen hver oppgave er. Dette har vi valgt med bakgrunn i at Azure har dette integrert. Og anbefalingen fra oppdragsgiver. Vi har delt opp prosjektet i ulike svømmebaner - oppdelingsrader i Kanban-boardet som visuelt kategoriserer ulike oppgaver (Kanbanize 2020) - som tilsvarer de planlagte utviklingsperiodene:

- Minimumsløsning
- Automatisering
- Administrasjonsside
- Forbedring

Her kan det være verdt å merke seg at vi hele tiden tilstreber tilbakemelding fra brukeren. Forbedringsprosessen vil derfor være en kontinuerlig prosess som går simultant med de andre utviklingsperiodene (for mer detaljert forklaring se Gantt-diagram). Vi har i tillegg satt opp følgende regler for bruk av kanbantavlen som skal være med å sikre god arbeidsflyt, og ferdigstillelse av oppgaver gjennom prosjektet:

- Det kan maksimalt ligge tre ferdigstilte oppgaver ventende på review til

enhver tid.
- Hvert gruppemedlem kan kun ha en oppgave i testfase til enhver tid.
- Hvert gruppemedlem kan ha opptil to aktive oppgaver til samme tid - inkludert oppgaver relatert til rapporten.

Oppgavene vil settes opp i prioritert rekkefølge i prosjektets backlog. Gruppen vil benytte møtefrekvens tilsvarende Scrum med både planleggingsmøter, daglige statusmøter og evaluering i etterkant av hver utviklingsperiode. Periodene skal alltid starte med en planleggingsseanse der tidsestimering og arbeidsfordeling vil være de sentrale holdepunktene. Underveis i hver av periodene vil vi gjennomføre korte daglige statusmøter mandag-fredag, der beslutninger av ulikt omfang kan sluttes. Hver periode avsluttes med en evaluering av arbeidet som er gjort. I tillegg til dette vil det avholdes ukentlige møter med intern veileder, samt ukentlige møter med kontaktperson fra Communicate - denne frekvensen justeres utover i prosjektperioden etter behov. Oppdateringer av backlog skjer i påfølgende statusmøte jamfør møte med oppdragsgiver.

C.5 Organisering av kvalitetssikring

C.5.1 Dokumentasjon, standardbruk, kildekode

Gruppen vil gjennomgående streve etter gode rutiner for både dokumentasjon, standardbruk og kildekode underveis. Det innebærer blant annet:

- All kildekode skal holdes sentralt i Azure DevOps Repos.
- All kode skal skrives som følge av kodestandard vedtatt av gruppen, se vedlegg D.
- Koden skal kommenteres godt underveis, og ved behov for mer gjennomgående dokumentasjon benyttes repository-ets wiki slik at det er lett å finne informasjonen.
- Koden skal også ha gjennomgående god testing, hvor gruppen sikter mot en høy testprosent på all kode, og all kritisk kode skal være fullstendig testet. Dette for å forenkle arbeidet dersom det gjøres større endringer, og for å forenkle prosessen med vedlikehold og videreutvikling senere.
- All kode skal gjennomgås av minst en annen på gruppen før den integreres.

C.5.2 Lagring

Alt arbeid lagres som hovedregel i skytjenester, og lokale kopier tas underveis for å sikre at data ikke forsvinner dersom en av tjenestene skulle gå ned.

- Rapport skrives i LaTeX-verktøyet Overleaf. Her legges også møtereferater, kontrakter, notater underveis og lignende.
- All prosjektkode legges løpende inn i respektive repositories i Azure DevOps Repos.
- Google Regneark benyttes til timeføring.

C.5.3 Utviklingsrutiner

- Azure Boards vil benyttes som Kanban-tavle for hele tiden å holde en prioritert liste over oppgaver som skal gjennomføres. På denne måten har hele gruppen, samt produkteier, alltid oversikt over hvordan arbeidet ligger an.
- Oppgaver fra Kanban-tavlen tas alltid etter høyest prioritet innenfor respektivt ansvarsområde.
- Hvert medlem skal kun ha ansvar for en oppgave av gangen, og den må legges klar (inkludert testing) for gjennomgang før neste oppgave kan påstartes.
- Prioriteten av oppgavene i Kanban-tavlen oppdateres etter hvert møte med oppdragsiver, med særlig fokus på hvordan den skal prioriteres ved starten av hver prosjektperiode.

C.5.4 Verktøy

Gruppen vil benytte en rekke verktøy under utviklingsprosessen. I tabell C.1 er en oversikt over disse.

Navn	**Type**	**Bruksområde**
Overleaf	Redigeringsverktøy og kompilator for LaTeX-dokumenter	Prosjektrapport
Visual Studio Code	Utviklingsmiljø front- og back-end	Utvikling
Azure DevOps	Primær utviklingsplatform med repositories, Kanban board og pipelines for CI/CD	Utvikling og prosjektstyring
Discord	Kommunikasjonsplattform	Kommunikasjon
Google Sheets	Timeføring	Prosjektstyring
Draw.io	Diverse diagrammer underveis	Diagramkonstruksjon
MatchWare MindView	Gantt-skjema	Prosjektstyring

Tabell C.1: Verktøy

C.5.5 Risikoanalyse

Gruppen har identifisert en rekke risikoer som kan påvirke prosjektet underveis i prosjektperioden. Dette er risikoer som har ulik grad av sannsynlighet, og ulike konsekvenser. Disse risikoene er:

- Prosjektet er ikke ferdig til deadline.

Gruppen tenker det er er forholdsvis usannsynlig at prosjektet ikke er ferdig til deadline, fordi kravene til hva som skal implementeres ikke er helt fastsatte. Det er lagt opp til at gruppen gjør så mye det er tid til med fokus på

kjernefunksjalitet, og leverer det som er klart. Det betyr at det hovedsaklig er rapporten som eventuelt er problematisk. Rapporten er hovedproduktet til prosjektet, og dersom denne ikke er klar ved prosjektslutt vil hovedmålet ved prosjektet ha feilet.

- En eller flere av medlemmene på gruppen blir langvarig syke eller på annen måte utilgjengelig(e).

Gruppen tenker at dette også er forholdsvis usannsynlig. Prosjektperioden pågår samtidig som Covid-19-pandemien, som gjør at gruppemedlemmene møter andre mennesker lite og tar få risikoer, slik at vanlige sykdommer og skader er mindre sannsynlig enn vanlig. Likevel er Covid-19 absolutt en risiko, og kan gjøre at gruppemedlemmene blir satt ut av å kunne arbeide produktivt dersom et medlem blir smittet. I tillegg er en av gruppemedlemmene idrettsutdøver, og det forhøyer naturlig sannsynligheten for skade mer enn for resterende medlemmer.

- Viktig arbeid, f.eks. kildekode eller rapport, går tapt.

Det er få som ikke har opplevd å miste filer til en disk som har blitt korrupt, en laptop som er mistet eller stjelt, arbeid som er glemt lagret eller lignende. Derfor er det en viss risiko for at dette skjer for gruppen, men det legges opp til håndtering for å mitigere risikoen.

- Urealistiske mål og arbeidsplan.

Det legges opp til mye funksjonalitet i applikasjonen. Å få til all funksjonaliteten kan derfor bli vanskelig å oppnå i løpet av utviklingsperioden. Dette er avklart og forholdsvis forventet av oppdragsgiver, som i utgangspunktet har som hovedmål å få en bedre brukeropplevelse og mer automatisering av arbeid som med dagens løsning gjøres manuelt. Derfor er det lite problematisk at ikke all funksjonalitet blir ferdig, så lenge et minimumsprodukt, samt automatisering, leveres.

- Tjenester kritisk for utviklingen går ned.

Det hender ofte at plattformer som benyttes går ned - slik som Azure DevOps eller Overleaf. Fordi dette er såpass store og velbrukte plattformer, virker det usannsynlig at slik nedetid vil være lang. Derfor kan tiden brukes til utvikling lokalt. I tillegg kan tid som ville vært brukt til utvikling heller bli brukt til rapportskriving, og tid som ville vært brukt til rapportskriving være tid som blir brukt på utvikling. Dette knytter seg også til risikoen om tapt arbeid, og håndtering av dette.

- Andre selskaper utvikler lignende løsninger.

Det er meget stor sannsynlighet for at lignende løsninger for utleie både allerede eksisterer, og er under utvikling. Likevel er det tilpasningene som gjøres i forhold til hvilke utleieobjekter som finnes og presentasjon av disse, tilpasninger til selskapets arbeidsflyt og automatisering av arbeidsoppgaver som i dag

gjøres manuelt, som gir prosjektet verdi for oppdragsgiver.

- Communicate gjør store endringer i kravspesifikasjonen underveis.

Dette ansees også som forholdsvis usannsynlig. Det har skjedd endringer i kravspesifikasjonen i forhold til første kommunikasjon allerede i planleggingsprosessen, men god kommunikasjon mellom gruppen og kontaktperson i Communicate har gjort at dette er oppklart tidlig. Derfor forventer gruppen at det kun kommer mindre endringer underveis, som fint kan håndteres med en smidig prosess.

- Kravene til teknologi som er stilt er for høye slik at gruppen ikke klarer å tilfredsstille dem.

Dette ansees også som forholdsvis usannsynlig at gruppen ikke klarer å benytte de fastsatte teknologiene, da det er teknologier og plattformer som er veldig utbredt i bransjen, og derfor er meget godt dokumentert. Skulle det likevel bli et problem med at gruppen ikke klarer å finne ut av et problem, er det god mulighet for å få hjelp av oppdragsgiver. På denne måten burde dette ikke bli noe problem. Det kan likevel hende at gruppen kaster bort tid, og det vil kunne påvirke risikoen om å ikke bli ferdig med prosjektet til deadline.

Risikotabell

Disse risikoene, sannsynligheten for at risikoen inntreffer og konsekvensen dersom den gjør det, er satt opp systematisk i tabell C.2. Skalaene for sannsynlighet og alvorlighet av konsekvens er oversettelser av Sommerville (2004). Sannsynlighet har gradene usannsynlig, lav, moderat, høy og veldig høy. Alvorlighet av konsekvens har gradene uproblematisk, tolererbar, alvorlig og katastrofal.

Nr.	**Risiko**	**Sannsynlighet**	**Konsekvens**
1	Prosjektet er ikke ferdig til deadline.	Lav	Katastrofal
2	En/flere av utviklerne blir langvarig syke eller på annen måte utilgjengelig(e).	Moderat	Alvorlig
3	Viktig arbeid, f.eks. kildekode eller rapport, går tapt.	Moderat	Alvorlig
4	Urealistiske mål og arbeidsplan.	Høy	Tolererbar
5	Tjenester kritisk for utviklingen går ned.	Moderat	Tolererbar
6	Andre selskaper utvikler lignende løsninger.	Veldig høy	Uproblematisk
7	Store endringer underveis i kravspesifikasjonen fra Communicate.	Usannsynlig	Alvorlig
8	Kravene til teknologi som er stilt er for høye slik at gruppen ikke klarer å tilfredsstille dem	Lav	Alvorlig

Tabell C.2: Risikoanalyse

Risikohåndtering

For risikoene som har et gjennomsnitt av over medium risk og konsekvens, har gruppen satt opp følgende metoder for risikohåndtering:

I tabell C.3 er en oversikt over hvordan gruppen vil håndtere de identifiserte risikoen:

Nr.	Tiltak
1	Arbeidet legges opp til at minste brukbare produkt tidlig er klart, og deretter de mest sentrale funksjonalitetene. Arbeid som ikke kan gjennomføres, og hvor avslutning legges diskuteres med oppdragsgiver.
2	Prosjektgruppen vil sette opp et møte, om et gruppemedlem ikke har mulighet, vil møtet enten bli satt til et annet tidspunkt eller kjøres uten dette medlemmet. Her vil det bli tatt opp kontraktavtale og konsekvenser dette vil gi for medlemmet det angår. Dersom en av utviklerene med hovedansvar for et område (Sindre eller Torbjørn) forsvinner, tar June over ansvar for det området.
3	For å sikre seg mot at kritiske dokumenter går tapt, enten kildekode eller rapport, lagres dataen både lokalt og i nettbaserte tjenester. Kildekode lagres løpende i repositories i Azure DevOps samt lokalt, og prosjektrapporten skrives i Overleaf og tas jevnlige backups av. Det legges inn mekanismer for disaster recovery der mulig, særlig for konfigurasjon av den allerede integrerte koden, samt nedlastning av all kildekode til LaTeX-dokumentet slik at lokale kompilatorer eventuelt kan benyttes.
4	Arbeidet settes opp på en slik måte at det fort utvikles et minste brukbare produkt, og deretter jobbes det etter prioritert liste over ønsket funksjonalitet. Dermed kommer gruppen så langt den gjør, men det er likevel et produkt som kan benyttes.
5	Utviklingen fortsetter lokalt hos den enkelte til tjenestene er oppe igjen, samt at det kan brukes tid på rapportskriving eller utvikling motsatt av tjenesten som er nede. Det legges ikke opp til kritisk arbeid helt opp til frister.
8	Gruppen gjør sitt beste for å finne svar på problemet, og støtter seg på kompetansen hos oppdragsgiver hvis nødvendig. Dersom dette heller ikke leder frem går gruppen sammen for å diskutere om teknologien er viktig for å levere et ferdig produkt, hvis ikke kan den bli lagt til sides. Ellers vil gruppen gå sammen for å finne en eventuell erstatning, eller løsning for å få den til.

Tabell C.3: Risikohåndtering

C.6 Plan for gjennomføring

C.6.1 Tidsplan

Etter planleggingsperioden legger gruppen opp til utviklingsperioder på tre uker, hvor hver periode har fokus på et område av løsningen. Arbeidet legges opp slik at løsningen vil være nyttig fra start, men bli mer komplett etterhvert som prosjektperioden pågår. Følgende datoer har blitt satt opp som en type milepæler for når ulike deler av løsningen skal være ferdig, lignende sprinter i

scrum.

- 1. Februar: Prosjektplan og prosjektavtale er ferdigstilt og levert. Valg av rammeverk skal være spesifisert.
- 6. Februar: Hovedvekt av kravspesifikasjoner er definert og utviklingsmiljø er satt opp.
- 8. Februar: Seneste oppstart utvikling.
- 26. Februar: Hovedfuksjonalitet (minste brukbare produkt) ferdig.
- 19. Mars: Automatiseringsfunksjonalitet ferdig.
- 9. April: Administrasjonsside ferdig.
- 30. April: Forbedringsperiode ferdig - utvikling ferdigstilles.
- 20.mai: Rapport leveres.

Det som ikke fremkommer i denne oversikten av milepæler, ei heller av Gantt-diagrammet, er at det vil skje iterasjoner internt i en utviklingsperiode. Dette vil komme naturlig av at gruppen har ukentlige møter med kontaktperson i Communicate, og at det her vil vises frem hva som er gjort siden forrige møte, og hva som evt. ønskes endret osv. Forbedringsperioden som avslutter utviklingen er ment som en periode for å fange opp bugs, kunne teste ut ulik funksjonalitet som ikke i utgangspunktet er en del av kravspesifikasjonen, samt gi noe fleksibilitet i tidsplanen.

Gantt-diagram

Med bakgrunn i tidsplanen gruppen har satt opp, er det laget et Gantt-diagram (figur C.1) som viser hovedaktivitetene i prosjektperioden. Målet med diagrammet er at gruppen skal få et oversiktlig innblikk i tidsplan for hovedaktivitetene som skal gjennomføres i prosjektperioden.

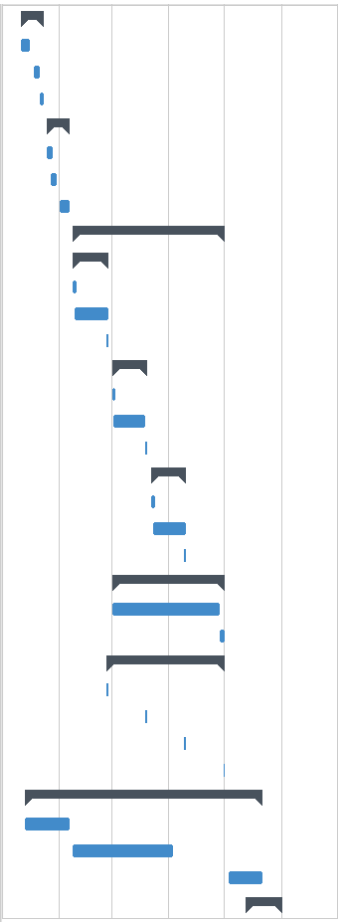
Den innledende delen av prosjektet vil bestå av to ulike deler. Prosjektplan, gruppeorganisering og informasjon om rammeverk vil prege de første ukene av prosjektet. Siste del av forprosjektet vil bestå av kravspesifikasjoner som samtidig skal dokumenteres i rapporten og på Azure plattformen til gruppen.

Umiddelbart etterpå vil utviklingsperioden settes i gang. Lengde på utviklingsbolkene vil være rimelig faste (varierer med 1-2 dager). I tillegg vil vi etter at hver funksjon er implementert, gå i gang med evaluering av perioden og testing. Her vil flere aktiviteter foregå parallelt gjennom planlegging, utvikling, testing og evaluering.

Etter at minimumsløsningen er implementert vil vi gjøre en vurdering etter tilbakemelding fra oppdragsgiver, som innebærer omprioritering for neste periode. Det vil si at vi vil ta stilling til om det er viktigere å fortsette med ny funksjonalitet veid opp mot forbedring av den allerede implementerte løsningen. Her vil det være stor sannsynlighet for at flere ulike oppgaver vil løses parallelt.

Utviklingsprosessen avløses av en periode hvor ferdigstillelse av sluttproduktet skal være hovedfokus. Når dette er avsluttet vil strukturering av innhentede resultater prioriteres. Før rapporten skal ferdigstilles og prosjektet avsluttes

1		**Prosjektplan**	**10 days**	**11.01.2021**	**22.01.2021**
2		Gruppeorganisering	5 days	11.01.2021	15.01.2021
3		Utviklingsmodell	3 days	18.01.2021	20.01.2021
4		Tidsplanlegging	2 days	21.01.2021	22.01.2021
5		**Kravspesifisering**	**10 days**	**25.01.2021**	**05.02.2021**
6		Funksjoner	3 days	25.01.2021	27.01.2021
7		Arkitektur	3 days	27.01.2021	29.01.2021
8		Komponentanalyse	5 days	01.02.2021	05.02.2021
9		**Utvikling**	**60 days**	**08.02.2021**	**30.04.2021**
10		**Min-løsning**	**15 days**	**08.02.2021**	**26.02.2021**
11		Planlegging	2 days	08.02.2021	09.02.2021
12		Utvikling	14 days	09.02.2021	26.02.2021
13		Evaluering	1 day	26.02.2021	26.02.2021
14		**Automatisering**	**15 days**	**01.03.2021**	**19.03.2021**
15		Planlegging	2 days	01.03.2021	02.03.2021
16		Utvikling	13 days	02.03.2021	18.03.2021
17		Evaluering	1 day	19.03.2021	19.03.2021
18		**Administrasjonsside**	**15 days**	**22.03.2021**	**09.04.2021**
19		Planlegging	2 days	22.03.2021	23.03.2021
20		Utvikling	14 days	23.03.2021	09.04.2021
21		Evaluering	1 day	09.04.2021	09.04.2021
22		**Forbedring**	**45 days**	**01.03.2021**	**30.04.2021**
23		Utvikling	42 days	01.03.2021	27.04.2021
24		Ferdigstillelse	3 days	28.04.2021	30.04.2021
25		**Test og deploy**	**46 days**	**26.02.2021**	**30.04.2021**
26		Min-løsning	1 day	26.02.2021	26.02.2021
27		Administrasjonsside	1 day	19.03.2021	19.03.2021
28		Automatisering	1 day	09.04.2021	09.04.2021
29		System	1 day	30.04.2021	30.04.2021
30		**Rapportskriving**	**92 days**	**13.01.2021**	**20.05.2021**
31		Kravspesifikasjoner	18 days	13.01.2021	05.02.2021
32		Dokumentasjon av pro...	40 days	08.02.2021	02.04.2021
33		Klargjøring	14 days	03.05.2021	20.05.2021
34		**Presentasjon**	**14 days**	**12.05.2021**	**31.05.2021**



Figur C.1: Gantt-diagram som viser overordnede aktiviteter i prosjektperioden

med en presentasjon.

Vedlegg D

Kodestandard

All kode skal godkjennes av en annen før implementasjon gjennom godkjennelse av pull request.
I backend benyttes Microsoft's kodestandarder for C# (Microsoft 2015), med følgende utdypninger:

- **Variabler**: Skal ha navn som tilsvarer dens innhold eller bruksområde.
- **Kommentering**: Så lenge funksjonsnavn/variabelnavn er gode, så vil en kommentar om hva funksjonen gjør være tilstrekkelig. Bedre med kort og godt kommentert enn lange og uleselige kommentarer.
- **Whitespaces**: Blanke linjer: Ingen unødige, men benyttes likevel for å dele opp funksjonalitet for å gjøre det mer leselig, slik at det ikke blir en "wall of text".
- **Whitespaces**: Alle operatører (+, -, =, ...) skal ha whitespace både foran og bak.

Annet

- Paranteser: Åpnende parantes på samme linje som funksjonsnavn.
- Testing: Viktig/sentral funksjonalitet skal alltid ha testing. Ellers sikter vi mot høy coverage.
- Leseliget: Sikt på å gjøre koden så lettlest og forståelig som mulig.

I frontend benyttes som hovedregel samme standarder, med følgende unntak:

- **Krøllparanteser**: Settes på samme linje som funksjonsnavn.
- **Variabler**: For variabelnavn benyttes camelCase.
- **Variabeltyping**: Bruk spesifisert variabeltype heller en ”any” som hovedregel, ”any” kan brukes for svært lokale variabler eller veldig tydelig type.

Vedlegg E

Detaljerte beskrivelser av funksjoner i FunctionAppService

AllocateRentals(DateTime today)

AllocateRentals er en funksjon som henter alle tildelingsperioder i databasen. For hver tildelingsperiode vil funksjonen sjekke om det er dagen etter en søknadsfrist eller 14 dager før. Dersom det er dagen etter en søknadsfrist vil tildelingsfunksjonen trigges. Dersom det er 14 dager før en søknadsfrist vil en epost med en påminnelse om søknadsfristen sendes til alle brukerne. Dersom det er hvilken som helst annen dag vil funksjonen avsluttes.

UpdateUserPoints(DateTime today)

UpdateUserPoints henter brukere (om noen) ansatt på dagens dato. Den lager en ny dato - dagens dato i år 2000 (siden alle brukere er satt til dette ansettelsesåret jmfr 5.2). Dersom en eller flere brukere blir returnert i dette kallet vil brukeren(e) bli tillagt ett poeng.

UpdateAllocationPeriods(DateTime today)

UpdateAllocationPeriods er en funksjon som trigges hver gang FunctionAppen trigges. Dens oppgave er å hente alle tildelingsperioder som har sluttdato før dagens dato. Det vil si at dersom det eksisterer en tildelingsperiode i databasen som stemmer overens med dette predikatet, vil året for denne oppdateres til neste år. På denne måten vil det hele tiden ligge tildelingsperioder for hvert utleieobjekt mellom ett og to år frem i tid (5.4.2).

SendBookingsToBlobOrSendChecklistReminder (DateTime today)

Henter bookinger med sluttdato etter i dag, sender epost om at sjekkliste enda ikke er levert dersom dette er tilfelle. Dersom sjekkliste er levert og

booking oppfyller kravene for flytting (se kapittel 5.4.3) lastes den opp i blob-containeren.

SendListOfBookingsForMonth(DateTime today)

Denne funksjonen trigges første dag hver måned, henter alle bookinger for den foregående måneden. Listen over bookinger blir formatert og sendt til gjennomgang hos ansvarlig i selskapet.

Vedlegg F

Praktiske notater fra administrator

De følgende notatene er svar fra kontaktperson i Communicate ifht. spørsmål gruppa har hatt relatert til praktiske spørsmål om hvordan enkelte teknikaliteter som påvirker utviklingen gjøres i praksis, særlig i forbindelse med poengsystemet og i forbindelse med trekk av lønn.

Notater til utleiesiden etter samtale med Sissel

fredag 22. januar 2021 10:18

Alle henvendelser til booking av en utleieenhet skjer via booking@communicate.no.
Når det gjøres en booking via siden i dag, genereres en mail som sendes dit. Man kan også sende en mail direkte til denne adressen.
Booking@communicate.no har ingen innebygget automatikk, og alle henvendelser håndteres manuelt.

Hver søknadsperiode har en frist. Eks. Perioden 1.mai til 30. oktober har frist 15. mars. I forkant av denne fristen sendes det ut en mail med påminnelse om at fristen nærmer seg (Gjøres manuelt i dag).
Alle søknader for perioden som sendes inn før fristen, samles opp (i dag i et word-dokument). Når fristen er nådd, ser man på alle søknadene som har kommet inn. Er det konkurranse om en utleieenhet i samme periode, avgjøres dette via poengsystemet.
Om man søker i en periode etter at fristen for perioden har utgått, behandles søknaden fortløpende. Eks. 1. Juni går jeg inn på utleiesidene for å se om uken 7. - 13. Juni er booket av noen. Er den ikke det, kan jeg sende inn en søknad om å booke den uken, selv om fristen for perioden var 15. mars. Med mindre noen søker på den samme perioden samtidig, vil jeg mest sannsynlig få booket denne uken.
Når en søknad er behandlet og bookingen bekreftet, legges den manuelt inn i en google calendar som et event, hvor navnet på eventet er navnet på den ansatte som leier i perioden.

En skal kunne avbestille bookingen sin. Skjer dette 1 uke eller lenger før leieperioden starter, er det ingen kostnad. Skjer avbestillingen under en uke før leieperioden starter, skal fremdeles brukeren betale for leien, med mindre en annen bruker booker perioden.
Kostnaden av leie blir trukket fra brukeren sin neste lønning.

Poengsystemet er i dag kun tilgjengelig for Adminer i selskapet å se (Får tak i dataen når det blir relevant). Tellingen og justeringen av poeng gjøres i dag manuelt gjennom et excel ark.
En bruker får 1 poeng for hvert år de jobber i communicate. Når de leier en utleieenhet, trekkes de 1 poeng. Om de leier i en oppsatt ferie (eks. Vinterferie, jul, høstferie osv), trekkes de 2 poeng. Er det konkurranse om en enhet, er det brukeren med flest poeng som får utleieenheten. Har brukerne like mange poeng, gjøres en tilfeldig trekking.

Alle utleieenhetene har et tilhørende rapporteringsskjema som fylles ut når man forlater enheten. Dette skjemaet skal i dag leveres sammen med nøklene. Dette er en del av løsningen jeg tror kan digitaliseres, spesielt om man kan fylle ut skjemaet på en mobil mens man går gjennom enheten og sjekker at alt er gjort. Denne kunne så bli sendt direkte til et endpoint som behandler den. Kanskje kan det også sendes en mailvarsling om siste leiedag nærmer seg, og man ikke har fylt ut skjemaet.

Jeg tror løsningen kan dra god nytte av å ha en admin side. Her kan man for eksempel justere tildelingsperioder og frister, se rapporter, fjerne bookinger osv.
Det hadde også vært fint om man her kunne endret regler, endret rapporteringsskjema osv. Dette gjør det lett å modifisere innholdet på siden uten at man trenger å innvolvere en utvikler.

Dere kan helt fint flytte rundt på informasjon på siden, så lenge all informasjonen er med, og er knyttet til relevant enhet. Jeg ser ikke noe problem i det å ha en global informasjonsside også, som inneholder informasjon på tvers av alle enheter.

Notater til utleiesiden etter samtale med Sissel 2

onsdag 27. januar 2021 16:12

Når trekkes poeng fra en bruker?

Poengene trekkes når en bruker får tildelt den perioden de har søkt om. Poeng trekkes KUN når det gjøres en søknad før fristen har gått ut. Det vil si at "førstemann til mølla" tildelinger ikke trekker poeng.
Om det blir gjort endringer eller avbestillinger i søknaden ETTER at fristen har gått ut, blir poengene fremdeles trukket, og brukerne får dem ikke tilbake.
Om en bruker søker om endring av en periode (for eksempel fra en ferieuke = 2 poeng til en vanlig uke = 1 poeng) er det den første trekningen som gjelder uansett. Ja, dette betyr at brukere skal kunne søke om å endre bookingene sine likevel.

Man kan søke om å booke en utleieenhet selv om man ikke har opptjent poeng. Om dette gjøres i hovedtrekningen (før fristen går ut) gjelder følgende:
Om 2 personer som har 0 poeng har søkt på den samme utleieenheten den samme uken, er det brukeren som startet i selskapet først som vinner bookingen.
Startet begge brukerne på samme dato (ikke uvanlig på grunn av MSU), gjøres det en tilfeldig loddtrekning.

Kommunikasjon med lønnsavdelingen

Leien blir trukket av lønna til hver enkelt ansatt.
Det sendes en oversikt over hvem som skal trekkes hver måned, og denne oversikten sendes noen dager før lønnsutbetalingen. Vi har utbetaling den 15. hver måned, som betyr at denne oversikten normalt sett sendes ut mellom 10. og 12. hver måned. Denne datoen må kunne justeres fra måned til måned (av en admin), i og med at det av og til er behov for informasjonen tidligere enn dette (feks ferier osv).
Oversikten sendes som en mail til lonn@communicate.no (Ikke bruk denne mailen i development, bruk en personlig mail som test).
Oversikten inneholder alle bookinger som gjelder fra 15. måneden før, til 15. gjeldende måned (Dvs. En mail som sendes ut 12. april inneholder bookinger fra 15. mars til 15. april).
Mailen må inneholde navn på person som skal trekkes i lønn, hvilken utleieenhet de skal leie, hvilken uke de leier og beløpet de skal trekkes.
En admin kan spørre en bruker om småting, som for eksempel å ta med nye batterier, ta med nye lyspærer, lese av vannmåler osv. Noe av dette kan medføre at de skal trekkes mindre, eller ikke i det hele tatt. Derfor må det være mulig for en admin å modifisere mailen som sendes til lønnsavdelingen før den sendes.
Det er også fint om en admin kan gjøre notater på dette per booking, slik at de kan gå tilbake og se hvem som tok med lyspærer for en uke siden feks. (Fritekst notatfelt?!)

Om det skulle skje at en bruker avbestiller etter at trekkingen i lønn har blitt sendt til lønnsavdelingen må det noteres et sted at brukeren har en gratis leie til gode (Igjen, samme fritekstfelt??). Slik det er i dag, om en bruker avbestiller slik, og en annen bruker tar over bookingen, blir de bedt om å gjøre opp økonomisk seg imellom. Vet ikke om dere finner en god løsning på dette?

Hva ønsker admin varsel om?

I og med at admin skal kunne forespørre ting fra leietakere, gi beskjeder, gi ut nøkler osv, er det greit om de blir varslet om alle bookinger, avbestillinger og endringer. (Jeg tenker systemet vi snakket om, hvor admin kan velge hva de blir varslet om er good her).

Merk: I påfølgende tekst har adresser og stedsnavn blitt byttet ut med mer generiske navn for noe anonymisering av stedene for publisering av oppgaven. Det er markert [slik] hvor det er gjort endringer.

Vi leier ut [hytta i Valdres og leiligheten i Spania] kun ukesvis mandag-mandag, og [leilighten i Oslo] hele helgen. Noen ganger kan det være at en ansatt bare skal sove over i [leiligheten i Oslo] den ene natta, og da kan en annen kollega overta den andre natten. Da er det normalt at de deler på leien; på den måten at de gjør opp seg imellom (Communicate trekker hel leie fra den som booket). For [hytta i Valdres] er det som oftest helgen som er aktuell å bruke, så her tror jeg ikke folk pleier å dele. Men om en person har lyst å være der en dag ekstra eller to, så må han sjekke når nestemann kommer – og kommer han ikke før mot helgen uansett så kan han jo være der noen dager ekstra såfremt det er ok for både nestemann og vaskemannen (som normalt vasker på mandager). Men Communicate trekker kun søkeren for leie.

Pendlere kan benytte [leiligheten i Oslo] fast i ukedagene, og dette er avtalt på forhånd med lederen. Andre ansatte kan også booke seg inn ifm f.eks kurs eller overtid/sene kvelder midt i uka. Både pendlere og andre ansatte må booke seg inn som vanlig før oppholdet, og da legges det ut på kalenderen slik at alle kan se. Hverken pendler- eller ansatt-opphold midt i uka blir trukket for leietrekk. Vi tar kun betalt for helgene/fridagene, hvor man har leiligheta for seg selv. Om faste pendlere betaler noen slags fast husleie utenom vet jeg ikke hva som er avtalt, men uansett ikke via systemet.

Angående varsling av brukere når det kommer til innsending av sjekklister: Påminnelsen sendes til personen som har leid utleieobjektet, gjerne 2-3 dager etter endt leieperiode. Denne varselen trenger ikke å sendes til admin, så lenge admin har en side hvor de kan se oversikt over alle innsendte sjekklister.

Vedlegg G

Timeføring

Januar

Day of Month	Day of Week	June	What	Torbjørn	What	Sindre	What	Total
11	Monday	1:30	Gruppemøte + ekst. veileder	1:30	Gruppemøte	1:30	Gruppemøte	4:30
12	Tuesday	0:30	Møte int. veileder	0:30	Møte med intern veileder	0:30	Møte med veil.	1:30
13	Wednesday	2:30	Research .NET og Angular	0:45	C#, Azure Board/DevOps	0:00		3:15
14	Thursday	1:45	Lynkurs (infomøte)	2:45	Lynkurs, gamle oppgaver, prosjektplan	0:45	.net, typescript læring	5:15
15	Friday	0:00		3:00	C#, .Net, Prosjektplan, nåværede løsning	3:00	Angular, Prosjekt oppsett	6:00
16	Saturday	0:00		0:00		0:00		0:00
17	Sunday	0:00		2:30	Azure/typescript læring	0:00		2:30
18	Monday	3:00	Møte, research Angular	6:00	Møte og .NET læringsvideoer. Bli kjent i azure	3:00	prosjektplan skriving og planlegging	12:00
19	Tuesday	4:30	Møte, prosjektplan	5:00	Møte, prosjektplan og eksempelprsosjekt	4:00	Møte, eksempelprosjekt, research	13:30
20	Wednesday	4:00	Research utv.modell, prosjektplan	3:00	Møte og Gantt diagram	2:30	Møte, research	9:30
21	Thursday	6:00	Møter, prosjektplan	7:00	Møter, prosjektplan	5:00	Møte, prosjektplan	18:00
22	Friday	7:00	Prosjektplan	2:00	Arkitektur research	0:00		9:00
23	Saturday	0:00		2:30	Prosjektplan, Gantt, Tidligere oppgaver	0:00		2:30
24	Sunday	0:00		0:00		0:00		0:00
25	Monday	3:00	Prosjektplan, research	4:00	Microsoft developer docs	2:00	Use-Case diagram og prosjektplan	9:00
26	Tuesday	8:30	Prosjektplan, kravspesifisering, møter	4:30	Møter, prosjektplan, arkitektur	5:00	Møter, Domene modell, research	18:00
27	Wednesday	4:00	Møte, kravspesifisering, iterasjon av tekst	4:00	Møte, research	5:00	Designskisser, møte, kravspesifisering	13:00
28	Thursday	3:00	Møte, kravspesifisering	4:00	Veiledning, prosjektplan	2:50	Møte, prosjektplan	9:50
29	Friday	8:00	Møte, prosjektplan, div. vedlegg, research	3:00	Møte, prosjektplan, C#	3:00	Møte, prosjektplan	14:00
30	Saturday	9:00	Research (GameJam)	3:30	C#	0:00		12:30
31	Sunday	9:30	Research (GameJam)	2:00	C#/Rapport	2:00	designskisser, angular	13:30
		75:45		61:30		40:05		177:20

Februar

Day of Month	Day of Week	June	What	Torbjørn	What	Sindre	What	Total
1	Monday	1:00	Møte	3:00	Møte, C#, rapport	7:00	Møte, Angular, ts, bootstrap	11:00
2	Tuesday	0:00		3:00	research autentisering/autorisering	4:00	Angular, bootstrap	7:00
3	Wednesday	2:00	Møte, research	3:00	Møte, bli kjent med angular	4:00	Møte, angular research	9:00
4	Thursday	10:00	Frontend - utleieobjektvisning	3:00	Research azure ad	5:00	Frontend-Header	18:00
5	Friday	8:00	Frontend - utleieobjektvisning	4:00	Møte, OpenId, OAuth	2:00	Frontend-Header	14:00
6	Saturday	0:00		3:00	Identity, sekvens for Logg inn	0:00		3:00
7	Sunday	0:00		2:30	Rapport			2:30
8	Monday	6:00	Konvertere frontendprosjekt, build pipeline	4:00	Rapport , research	7:00	Frontend-header og framside	17:00
9	Tuesday	10:00	Møte, lage/fikse figurer, opprette resources, research, frontend - utleieobjektvisning	3:00	møte, arkitektur	3:00	frontend-framside	16:00
10	Wednesday	5:00	Møte, frontend - utleieobjektvisning	5:00	møte, planlegge funsksjoner backend, ad	4:00	frontend-hjemmeside, møte	14:00
11	Thursday	8:30	Møte, div. ressursadmin, frontendmerging	7:00	Ad, logg inn	7:00	møte, frontendmerging	22:30
12	Friday	7:30	Møte, frontendmerging, utleieobjektvisning	4:30	Logg inn feilsøking	5:00	DB research	17:00
13	Saturday	0:00		3:00	Logg inn feilsøking	0:00		3:00
14	Sunday	0:00		3:00	Feilsøk logg inn	0:00		3:00
15	Monday	8:00	Møte, frontend - utleieobjektvisning, feilsøk	4:30	Møte, feilsøk logg inn	7:00	Møte, DB-setup/research	19:30
16	Tuesday	5:00	Møte, frontend - utleieobjektvisning, feilsøk	4:00	Møte, domenemodell	5:00	Møte, DB-setup	14:00
17	Wednesday	6:00	Rapport, frontend - kart, feilsøk	4:30	rapport	4:00	Møte, DB-setup	14:30
18	Thursday	9:30	Møter, konvertere frontendprosjekt, build pipeline	5:30	rapport, møte, db	5:00	Møte, db-setup, EF Core research	20:00
19	Friday	3:30	Møte, frontend	4:00	db setup	4:00	db setup	11:30
20	Saturday	0:00		0:00		0:00		0:00
21	Sunday	0:00		0:00		0:00		0:00
22	Monday	9:45	Møte, db-setup, frontend, build pipeline	7:30	møte, db	7:00	Møte, db setup/research	24:15
23	Tuesday	7:30	Møte, frontend, nytt forsøk på deployment	5:00	møte, backend/db	4:30	Møte, db/api setup/research	17:00
24	Wednesday	7:30	Møte, deploymentfeilsøk, API-endepunkter	6:30	backend	5:00	Api setup/research	19:00
25	Thursday	2:30	Møte, build pipeline feilsøk	7:00	backend	4:00	api setup	13:30
26	Friday	7:00	Møte, rapportskriving	4:00	backend	3:50	api setup	14:50
27	Saturday	0:00		0:00		0:00		0:00
28	Sunday	7:00	Frontend & backend cleanup for fullstack calls	0:00		0:00		7:00
		131:15		103:30		97:20		332:05

Mars

Day of Month	Day of Week	June	What	Torbjørn	What	Sindre	What	Total
1	Monday	9:15	Rapport, møte, utleieobjekt fullstack	6:30	Møte, models	7:00	møte, form til nytt utleieObjekt	22:45
2	Tuesday	7:00	Møte, rapport, bugfixes, booking	8:45	Checklists, møter	5:00	form til nytt utleieObjekt	20:45
3	Wednesday	7:30	Møte, CORS-feilsøk	6:00	allokering	7:00	Research, form til nytt utleieObjekt	20:30
4	Thursday	3:30	Møter	7:00	møte, allokering	5:00	form til nytt utleieObjekt	15:30
5	Friday	3:00	Møte, booking	4:00	Emailvarsling, møte	4:30	form til nytt utleieObjekt	11:30
6	Saturday	0:00		0:00		0:00		0:00
7	Sunday	0:00		0:00		0:00		0:00
8	Monday	5:30	Møte, booking	6:00	Emailvarsling	6:00	form til nytt utleieObject	17:30
9	Tuesday	7:00	Møter, booking	5:15	Emailvarsling, msgraph	5:45	form til nytt utleieObject, rentalPage	18:00
10	Wednesday	4:00	Møte, booking	4:30	Msgraph, møte	4:30	Bootstrap research	13:00
11	Thursday	5:30	Møte med msgraph feilsøk	4:45	Msgraph, møte	8:00	rentalPage	18:15
12	Friday	7:30	Møte, booking, msgraph	6:45	Msgraph, functions	5:00	rentalPage	19:15
13	Saturday	0:00		3:30	Msgraph, diverse opprydding av endringer	0:00		3:30
14	Sunday	0:00		0:00		0:00		0:00
15	Monday	8:30	Booking, møter, checklist	6:00	msGraph, function apps	6:00	rentalPage, AngularTesting	20:30
16	Tuesday	7:00	Møter, checklist	8:00	function apps	7:30	angular testing	22:30
17	Wednesday	8:00	Møter, checklist, testing	8:15	function apps, deployment	8:00	angular testing	24:15
18	Thursday	8:30	Møter, function app debug, div. research	6:45	function apps, mapping	7:15	Møte, angular testing	22:30
19	Friday	6:00	Møter, checklist	6:15	code review	6:00	angular testing	18:15
20	Saturday	0:00		0:00		0:00		0:00
21	Sunday	0:00		1:00	research testing	0:00		1:00
22	Monday	9:00	Møte, checklist, booking kansellasjon	7:15	booking, rentalObject init, testing	8:00	Registrer rentalObject	24:15
23	Tuesday	10:30	Møter, frontend adjustments, testing	6:30	code review improvement	7:30	Registrer rentalObject	24:30
24	Wednesday	7:00	Møte, checklist	5:00	code review improvement	5:45	registrer rentalObject validation	17:45
25	Thursday	3:00	Møte, merge inn unittester i backend	3:00	start testing	2:30	registrer rentalObject validation	8:30
26	Friday	1:00	Navigasjonsskuff	3:00	testing	2:00	registrer rentalObject validation	6:00
27	Saturday	0:00		0:00		0:00		0:00
28	Sunday	0:00		3:30	testing	0:00		3:30
29	Monday	5:00	Cleanup, navigasjonsskuff	3:00	testing	3:00	Research tidligere oppgaver	11:00
30	Tuesday	9:30	Navigasjonsskuff	1:15	testing	0:00		10:45
31	Wednesday	3:00	Navigasjonsskuff	2:00	testing	0:00		5:00
		145:45		133:45		121:15		400:45

April

Day of Month	Day of Week	June	What	Torbjørn	What	Sindre	What	Total
1	Thursday	1:30	Sjekklistevisning admin	2:15	testing	0:00		3:45
2	Friday	0:00		0:00		0:00		0:00
3	Saturday	0:00		1:15	testing	0:00		1:15
4	Sunday	0:00		3:30	Testing	0:00		3:30
5	Monday	6:00	Testing frontend	2:30	testing	4:00	Registrer rentalObjekt velidation	12:30
6	Tuesday	12:30	Møter, backlog chug	4:00	testing input validation	6:00	registrer rentalObjekt validation	22:30
7	Wednesday	6:00	Møte, rapportarbeid (figur + personvern)	4:00	input validaton	7:30	registrer rentalObjekt validation	17:30
8	Thursday	4:00	Møter, research	4:00	testing	5:00	registrer rentalObjekt validation	13:00
9	Friday	5:00	Møte, sjekklister for admin	4:00	testing	4:00	admin-userManagementPage	13:00
10	Saturday	0:00		0:00		0:00		0:00
11	Sunday	0:00		0:00		0:00		0:00
12	Monday	7:30	Møte, sjekkliste admin, kartvisning	7:00	backend	8:00	admin-userManagementPage	22:30
13	Tuesday	11:00	Møte, div admin funksjonalitet	6:00	backend/administrativt	7:00	admin-userManagementPage	24:00
14	Wednesday	8:30	Feedback modal	6:00	testing	7:30	admin-userManagementPage	22:00
15	Thursday	10:30	Styling form utleieobjekt	6:00	div backend cleanup	8:00	admin-userManagementPage	24:30
16	Friday	7:00	Last touches før brukertesting	4:00	div backend cleanup	4:00	admin-rentalObjectEdit	15:00
17	Saturday	0:00		0:00		0:00		0:00
18	Sunday	0:00		0:00		0:00		0:00
19	Monday	5:00	Møte, rapportskriving - teknologier	5:30	endringer i bookingController/testing	5:00	admin-rentalObjectEdit	15:30
20	Tuesday	8:30	Forbedringer ifht user feedback	5:00	rapport flytte resolved feedback	6:30	admin-rentalObjectEdit	20:00
21	Wednesday	4:00	Møte, rapportskriving - teknologier	6:00	feedback/testing	5:00	admin-rentalObjectEdit	15:00
22	Thursday	2:00	Møter	4:00	backend	4:00	admin-rentalObjectEdit	10:00
23	Friday	5:00	Møter, image post testing	4:00	backend	5:00	admin-rentalObjectEdit	14:00
24	Saturday	0:00		1:00	backend	0:00		1:00
25	Sunday	0:00		0:00		0:00		0:00
26	Monday	6:00	Møte, rentalObject post	4:00	backend	8:00	RentalObject post	18:00
27	Tuesday	5:30	Møte, checklist edit	3:00	backend	6:00	rentalObject edit	14:30
28	Wednesday	7:30	Kalender for booking overview	7:00	backend	7:00	rentalObject edit	21:30
29	Thursday	6:30	Debug frontend	7:00	backend	6:00	rentalObject mapping	19:30
30	Friday	4:30	Kalender for booking overview	7:00	cleanup - rapport	6:00	admin functionality	17:30
		134:00		108:00		119:30		361:30

Mai

Day of Month	Day of Week	June	What	Torbjørn	What	Sindre	What	Total
1	Saturday	5:00	Div. fixes fra code review/user feedback	0:00		0:00		5:00
2	Sunday	4:00	Div. fixes fra code review/user feedback	0:00		0:00		4:00
3	Monday	5:00	Rapport - Modeller	7:00	rapport - teknologier	7:00	admin funksjonalitet	19:00
4	Tuesday	6:00	Rapport - teknologier/UI	6:00	rapport- implementasjon backend	6:00	rapport - design, teknologi	18:00
5	Wednesday	8:30	Rapport - UI	5:00	rapport	7:00	UI, admin funksjonalitet	20:30
6	Thursday	3:30	Møter, rapport - UI	5:00	rapport	6:00	admin funksjonalitet	14:30
7	Friday	2:30	Møte, rapport - CI/CD	5:00	rapport	4:00	admin funksjonalitet	11:30
8	Saturday	0:00		0:00		0:00		0:00
9	Sunday	4:30	Rapport - CI/CD	0:00		0:00		4:30
10	Monday	7:30	Møte, figur - sekvensdiagram	6:00	rapport	7:30	admin funksjonalitet	21:00
11	Tuesday	5:00	Figur - sekvensdiagram	4:45	rapport	6:00	rapport	15:45
12	Wednesday	6:00	Møter, rapport	5:00	rapport	5:30	rapport	16:30
13	Thursday	6:30	Figurer, brukertesting	5:00	rapport	6:00	rapport	17:30
14	Friday	4:00	Rapport, rettlesing	5:15	rapport	5:00	rapport	14:15
15	Saturday	5:00	Rapport, figurer + diskusjon	5:00	rapport	4:00	rapport	14:00
16	Sunday	2:30	Rapport, rettlesing	2:30	rapport	2:30	rapport	7:30
17	Monday	0:00		0:00		0:00		0:00
18	Tuesday	5:30	Rapport, finpuss	7:00	rapport	6:00	rapport	18:30
19	Wednesday	8:00	Rapport, finpuss	8:30	rapport	8:30	rapport	25:00
		89:00		77:00		81:00		247:00

Vedlegg H

Brukertesting

Merk at gjennomgående i dette vedlegget er navn og annen identifiserbar informasjon er fjernet fra tekstene, dette er markert [slik].

H.1 Utsendt informasjon til testere

Den følgende teksten ble sendt ut til testerne:

> Takk for at du vil være med å teste den nye utleiesiden!
> Merk at alt dere foretar dere ikke blir lagt inn i det aktive systemet. Det trekkes ikke lønn, og bookingene er ikke gyldige. Lenken finner dere her: https://[nettside].azurewebsites.net/
> Vi har noen known bugs listet lenger ned, men utover dette ønsker vi tilbakemelding på alt av bugs, forbedringspotensiale osv dere kommer på! :)
> Dette kan enten sendes via tilbakemeldingsskjemaet, eller til Kaya som vidresender det til oss. For vidreutvikling som den del av bachelorperioden ønsker vi naturlig nok tilbakemelding så snart som mulig, men før utgangen av uke 16. For senere utvikling kan dette løpende legges inn i tilbakemeldingsskjemaet.
> Noen ting vi ønsker at dere ser på:
>
> - Generell navigering.
> - Legge til ny booking (både snart og litt frem i tid).
> - Kansellere booking (kan gjøres for bookinger frem i tid).
> - Teste sjekkliste (kan gjøres for bookinger som har startdato i dag eller senere).
> - Sende inn tilbakemelding.
>
> I tillegg ønsker vi at admin-testere ser litt på følgende:
>
> - Se at bookingoversikt og sjekklister oppfører seg som forventet.
> - Legge til ny lokasjon. Merk at nye lokasjoner ikke faktisk blir lagt til slik at de kan vises i etterkant. Enn så lenge er det opplevelsen av å fylle ut skjemaet som er sentralt.
> - Legge til nye brukere (her må det benyttes faktiske communicate.no-brukere, andre fungerer ikke) - de vil ikke motta noe epost eller noe slikt om at de er lagt til.

Known bugs/limitations:

- Systemet er ikke optimalisert ifht. hva som lastes, og kjører også på free-tier webapps, så det vil nok oppleves noe tregere enn det vil gjøre ved endelig utrulling.
- Informasjonen som ligger i de ulike utleieobjektene ikke stemmer med virkeligheten og er bare testdata (mye er f.eks. nå likt mellom lokasjonene). Dette vil naturlig nok endres før faktisk bruk - dette er bare for å vise hvordan sidene vil bli seende ut, og hvilken type informasjon som vil ligge der.
- Kart og bilde-karusell oppdateres ikke når det navigeres mellom lokasjoner i sidebaren.
- Det er ingen Før avreiseseksjon i visningen av lokasjoner enda.
- Ingen oversiktskalender over bookede uker i lokasjonsvisning enda.
- Bekreftelsesepost kommer fra en gmail, blir byttet med en ordentlig epostadresse ved utrulling.

Admin:

- Kan ikke enda legge til bookinger for andre enda.

Med vennlig hilsen, bachelorgruppa bestående av Sindre, Torbjørn og June

H.2 Oppsummert tilbakemeldinger

Oppsummert var fra tilbakemeldingene var det ønsker om følgende:

- Oppretting i at et felt ikke oppdaterte seg korrekt i RentalObject-skjemaet.
- Gjøre sjekklisten mer brukervennlig.
- Retting av språkfeil på en knapp.
- Mulighet for å sjekke navigasjon i kartene.
- Automatisk utsendelse av informasjon før avreise.
- Mer informasjon om sengeplasser, bad, stuer osv.
- Fritekstfelt i sjekklisten.
- Gjøre alle cards like store.
- Epostbekreftelse ved kansellering.
- Tooltip om hvor sjekklisten sendes.
- Ikke tillate overlappende bookinger.
- Fikse at bildekarusell og kart ikke oppdaterer seg ved navigasjon mellom utleieobjekter.
- Ønske om rik tekst i tekstfeltene for registrer lokasjon.
- Ønske om 24-timersklokke istedenfor AM/PM i registrering av lokasjon.
- Ønske om god håndtering av klokkeslett.
- Fjerne piltaster for pris i registrer lokasjon.
- Restriksjon på størrelser og format på bilder på utleieobjekt.
- Tilbakemelding ved registrering av utleieobjekt.

- Avbryt-knapp på booking.
- Totaloversikt over bookinger.
- Administratorkommentar på bookinger.
- Egen pil for valg av årstall på brukeradministrasjon.
- Annen sortering av innsendte sjekklister.
- Fjerning av adresse som ikke er i bruk.

Mer om hvordan tilbakemeldingene er håndtert i kapittel 7.4.2.

H.3 Rådata

H.3.1 Via tilbakemeldingsfunksjon

Følgende tilbakemeldinger kom via tilbakemeldingsfunksjonaliteten på siden:

Hvilken side gjelder det?
Registrer lokasjon
Hva er feil, eller hva kan forbedres?
Hvis en først har skrevet en ugyldig karakter i feltet "Stednavn på engelsk", så er det ikke mulig å rette det opp igjen uten å laste registreringssida på nytt. Det forblir en rød feil-markering uansett hva en prøver å rette det til.

Hvilken side gjelder det?
Sjekkliste under "Mine bookinger"
Hva er feil, eller hva kan forbedres?
Hvis jeg åpner sjekklista, krysser av noen av punktene, sender inn, lukker og så åpner den på nytt, så er alle tidligere avkrysninger vekk. Det kunne kanskje være kjekt å enkelt kunne se hva en har igjen å gjøre.

Hvilken side gjelder det?
Ny booking (alle lokasjoner)
Hva er feil, eller hva kan forbedres?
Liten språkfeil: "Forespørr" skulle vært "Forespør" (én r).

Hvilken side gjelder det?
Generelt
Hva er feil, eller hva kan forbedres?
Bookingen ser generelt veldig fin ut! Enkel å navigere rundt i og oversiktelig å finne informasjon. Dere har vært veldig flinke!

Noen tilbakemeldinger:
Sjekklisten: Etter man har sendt inn så endres ingenting, man kan fremdeles fylle inn og sende inn på nytt. Det bør kanskje stå "sjekkliste sendt

inn" og så ikke mulig å bruke den mer?

Kartet er veldig fint! Noe som hadde gjort det enda mer brukervennlig var hvis vi kunne skrevet inn veibeskrivelser og søkt i kartet osv. Jeg ville f eks sjekke hvor langt det var fra leiligheten i Spania til flyplassen, men da måtte jeg kopiere inn adressen i et annet kart for å sjekke selv.

Vi har pleid å bli reservert inn ([hytta i Valdres], kanskje spania og?) en uke om gangen, selv om man kanskje bare booker fra tors-søn. Sikkert fordi det koster 1500kr per uke. Skal det fremdeles være sånn?

[Administrator] sender ut litt ekstra info (utvask, måking ++) når man f eks leier hytta. Hvis dette er helt standard informasjon så kan man kanskje få dette tilsendt automatisk før avreise så hun ikke trenger å gjøre det?

Til slutt noe jeg har tenkt på før også men som ikke nødvendigvis er så enkelt for dere å fikse, eller kanskje det er det :-) Hvis man ikke har vært på en av stedene før så syns jeg det er litt lite info om hvor mange mennesker det er plass til. Det står ingenting om hvor stor boligen er, hvor mange bad/stuer og hvor mange soveplasser (med/uten madrasser på gulvet). Det syns jeg hadde vært fint hvis man ikke har vært et sted før :-)
[navn]

Hvilken side gjelder det?
Sjekkliste
Hva er feil, eller hva kan forbedres?
Om mulig, et fritekstfelt hvor man kan skrive inn litt. Ellers synes jeg dette ser veldig bra ut :)

H.3.2 Via epost

Følgende tilbakemeldinger kom på epost:

Først og fremst, bra jobbet med utbedrelse av utleie siden til Communicate :) Dette er helt klart en forbedring av det vi har i dag!!

Tilbakemelding:

- Første som fanger øye mitt når jeg går inn på siden er at oversikten over hva som kan leies, ikke er helt symetrisk. Øynene våre liker symetri, så å få disse «Card viewene» i lik størrelse ville gjort mye for førsteinntrykket på landingssiden:
- Svanet email bekreftelse ved kanselering av booking.
- Ikke helt klart hvor denne sjekklisten havner. Hvem er det som får

denne? Kunne vært løst med noen «tooltips»/informasjonsbokser, med forklaring på sjemaet.
- Det er også mulig å åpne sjekklisten og sende inn samme ting flere ganger på samme booking.
- Jeg kan også overlappe bookinger, noe som ikke burde gått an.

Av disse punktene ville jeg prioritert sistnevnte. Fikset problemet med overlappende bookinger, samt fikset oppdatering av bilde karrusellen og kartene.

Veldig mye bra her også :) dette er bare de tingene jeg la merke til som trenger utbedring!

Hei!

Ser kjempefin ut :) :) :).

Registrer Lokasjon

- Mulig å få fet/understreket skrift, avkryssingsbokser, osv i tekstfeltene?
- Klokkeslett: Mulig å bruke kl 0500 og 1700, i stedet for AM/PM? (I systemet vi bruker per nå så booker jeg inn ansatte via Google-kalenderen, og der har vi sett at klokkeslettet iblant forskyver seg en time el.l, og det er visstnok ikke noe vi selv kan styre). Spiller vel ikke sånn stor rolle om det plutselig står utsjekk kl 2300 når det skulle stått kl 1200 i selve bookingen sålenge det står riktig klokkeslett i reglene da, men går det an å unngå er det fint :).
- Bra at man kan skrive inn summen selv. Unødvendig med pil opp/-ned i feltet for «Pris», siden det starter på 1 (tungvindt å trykke seg til kr 1.500)
- La inn noen bilder, både store (KB) og små. Bør sikkert være en maks størrelse? Stående/liggende?
- Etter jeg la inn siden med klokkeslett, priser osv, og trykket Ferdig, så ble jeg bare stående der i samme bildet, fikk ingen tegn til at det ble registrert, og gikk heller ikke automatisk videre til noen ny side.

Booking

- Kanskje en «Avbryt»-knapp, slik at man enkelt kan begynne på nytt hvis man trykket feil dato før man har sendt inn? Kan hende man plundrer litt før man skjønner at man skal velge både en avreise- og retur-dato i samme bildet :)
- Kan ikke kansellere bookinger i den uken man er (altså booking 20.-26. april kan ikke kanselleres 22. april, slik at andre kan se at den

faktisk er «ledig til helgen»). Blir vel krøll ifm betaling da kanskje. Hvis andre booker i stedet slipper jo den som har avbooket etter fristen å betale
- Hva med en totaloversikt som viser hvem som er der og når det er ledig? Mulig det handler om personvern, men kanskje fornavn kunne kommet opp når man holder musepekeren over helgen/uken det gjelder idet man skal søke?

Mine bookinger

- Etter at jeg har sendt inn sjekklisten ligger det fortsatt mulighet for å sende sjekklisten inn flere ganger (jeg ser ikke at jeg har sendt den inn tidligere, kanskje vil jeg sjekke om jeg husket å melde fra om noe). Her synes jeg den burde ligge der med det som er avkrysset/skrevet, uten mulighet for å redigere/sende inn på nytt. Evt redigeringsmulighet, men da må administrator få beskjed om at den er innsendt på nytt (i fall noen kommer med tilleggsopplysninger uker/måneder etter oppholdet).
- Jeg sendte inn søknad om samme enhet på samme dato 2 ganger. Er det mulig å forhindre det, at man får beskjed om at man allerede har søkt den enheten den uken? La seg dobbelt både her og under Bookingoversikten.

Bookingoversikt
Føler at det er litt uoversiktlig å finne ut hvem som skal til de forskjellige enhetene til helgen, neste helg osv. Kanskje det går an å fordele bookingene per enhet/en rad for hver enhet?
Mulig for Administrator å legge inn kommentarer på hver booking, f.eks ting som må huskes å informeres om, og som bare Administrator ser?

Sjekklister
Fint om den som er sist innsendt kommer øverst på listen.

Brukeradministrasjon
Mulig å få en egen pil for å velge årstall? Praktisk når man skal legge inn personer som ble ansatt for 25 år siden :).

Adresse
Adressen i Bergen må bort, vi har pt ikke noe kontor der.

Vedlegg I

Møtereferater

11.01.2021 Internmøte

Første møte, opprettelse av intern gruppekontrakt. Lastet ned prosjekt i Overleaf. Enighet rundt kodestandard. Første møte med bedriftsveileder.

- Ikke binde oss til noe som eksisterer.
- Microsoft Account brukes til innlogging

12.01.2021 - Veileder NTNU

Agenda:

- Utviklingsprosessen vi har sett for oss (bakgrunnskunnskap – design, høynivå, lavnivå - utvikling
- Hyppighet av møte med veileder
- Hva ønsker du i forkant av møter?
- Når skal vi ha neste møte?

Referat:

- Skiller A og B
    - Refleksjon og selvstendighet
    - Jobbe videre med råd han kommer med/refleksjon
    - Jobbe videre med råd han kommer med/refleksjon
    - Gir oppdragsgiver noe de ikke har tenkt på før (gir vi er, får vi mer)
- Viktigste produkt – rapport
- Beste modellen – proof of consept
- Utgangspunkt med møte 1 gang i uka
- Kjapt svar - send mail
- Vi styrer møtet
- sender mail dagen før som inneholder
    - Hva har vi gjort siden sist
    - Hva ønsker vi å diskutere
    - Hva er planen for neste periode?
- Plan for hvem som har ansvar for hva
- Plan for problemhåndtering
- Møte torsdager kl. 14

18.01.2021 - Internmøte

Bestemmelser:

- June er valgt til prosjektleder
- Avtale møte med Kaya tirsdag (i morgen)
- Oppgaven fokuserer på User Experience og High testability and maintainability

Til neste møte onsdag:

- Undersøke smidige utviklingsmodeller
- Gjøre en vurdering på ulike risikoer ved prosjektet

19.01.2021 - Communicate

- .NETCore3
- C# i front-end løsningen pga autentisering
- hovedplattform pc/pad - dynamisk som fungerer på mobil
- To muligheter
    - Web-app som kjører hele tiden- dyrere
    - Azure functions - API call - warm up/cool down
- Ønsker en løsning som er så billig som mulig
- Hvor mye trafikk er det vanligvis
- Komprimere og gjøre innhold mer oversiktlig
- Soveplasser ok - soverom?
- Ha med all informasjon, men kan omrokkeres
- Bør ha et design som gjør det enkelt å legge til nytt feriested
- brukere kan gis admin rettigheter (access control)
- førstemann til mølla utenom ferier
- Hovedmål er å automatisere mesteparten av prosessen når det kommer til utleie delen. Mindre administrative kostnader
- Innstillinger
- Bekreftelsesmail når hytte er tildelt sendes bekreftelsesmail
- Tips: finne kalender som tar høyde for norske helligdager
- Ha en feedback knapp på forsiden - lagres i en blob storage, hvor det kan markeres som behandlet/ubehandlet
- Brukergruppe som tester underveis?
- Azure portalen har alle komponenter som benyttes for å kjøre løsningen
- Ett repo per deployment
- Viktig å navngi ressurser i Azure slik at vi vet forskjell fra den andre bachelor gruppen
- Script som sjekker om ressurser eksisterer, dersom de ikke gjør det, opprettes disse automatisk
    - Veldig viktig for disaster recovery

- Samme script for dev/test
- Ordet dev i alle navn
- app settings puttes i git ignore (VIKTIG)
- Vi bruker template fordi det er en mer reel situasjon
- Kan være lurt å oppgradere template til angular 11
- Storage account inneholder forskjellige måter å lagre data på, foretrekkes fremfor cosmos pga kostnad
- bilder er naturlig å lagres som blob-container
- table storage for å holde de ulike utleiesidene
- noSQL kan gjøre det aktuelt å bruke cosmos
- NGX - bootstrap
- Møter tirsdager klokka 13

20.01.2021 - Internmøte

Hva gjør vi resten av uka?

- Kanban som overordnet SU-modell - June skriver i prosjektplan
- Organisasjonskart - Sindre skriver i prosjektplan
- Gannt diagram - Torbjørn
- Risikoanalyse - June
- Avgrensning av oppgaven - Torbjørn

21.01.2021 - Veileder NTNU

Referat:

- Viktigste hvordan vi har fordelt rollene istedenfor formell struktur
- Dokument-ansvarlig
- Noen ser spesielt på sikkerhet, teknologi osv mer interessant.
- Argumentasjon for hva vi har valgt (viktig)
- Viktigere å fordele dynamisk
- Felle: man noterer ned det man ble enig om, men ikke hvorfor det ble sånn
- Tenke hovedtema allerede fra start (overordnet nivå) i forhold til release osv.
- End-to-end basis før det legges til noe mer
- Kravspesifikasjon bør være ferdig så fort som mulig
- Detaljnivå kommer i utviklingsprosessen
- Rapportskriving bør vi bruke ca 1 uke på til slutt, gitt av vi har dokumentert masse underveis.
- Referansestil:
    - Viktig at man får med all nødvendig informasjon til å finne tilbake til riktig kilde.

    - Holde oss til en stil

- En glad sensor er en god sensor
- Gi bedre tid til forberedelse for Rune
- Konkret på hva vi ønsker tilbakemelding på

21.01.2021 - Internmøte

- Få på plass innlogging og kalender i første bolk
- Finjustering av hvordan vi viser frem informasjon i andre bolk
- Gjør ferdig mest mulig på prosjektplan ila uka.
- Agenda mandag:
    - Ferdigstille Gantt diagram
    - Gå gjennom hele prosjektplan
    - Kravspesifikasjon start-up

26.01.2021 - Internmøte

- Gjennomgang av Gantt-skjema
- Torbjørn skriver lite tillegg på valg av front-end rammeverk
- Gjennomgang av use case modell som Sindre har laget
- Funksjonelle krav spesifiseres
- Operasjonelle krav spesifiseres
- Oppgaver frem til neste møte er fordelt
    - Torbjørn ser på arkitektur
    - Sindre ser på domenemodell
    - June ser på funksjonelle krav

26.01.2021 - Communicate

- Generelt sett chrome, firefox og edge, men ingen krise om edge ikke er med
- Mail om at booking er foretatt, lonn@communicate.no som utgangspunkt. Kaya sjekker om det finnes direkte løsning mot lønnssystem.
- Når trekkes poengene? - Kaya sjekker
- Kan leie ved null poeng
- Sjekkliste ønskes å være mobilvennlig, lagres midlertidig?
- Språk: Norsk - engelsk kan gjerne implementeres
- Opprette og slette booking, - ikke endre
- Alle søkander samles inn frem til frist, kalkulasjon. anta første mann til mølla etter den fristen.
- Mail til nåværende løsning skal legges inn manuelt
- Hva skal varsles - undersøkes

- Azure garanterer oppetid for oss så lenge vi velger en subscription som gir denne muligheten
- kan være lurt å sette opp en test som sjekker oppetid (+ andre funksjonaliteter)

27.01.2021 - Internmøte

- Kanban boards ble satt opp
- Repo ble opprettet
- Eventuelle spørsmål ved kravspesifisering

28.01.2021 - Veilder NTNU

Presiseringer til prosjektplan:

- Dagens løsning er i stor grad... for hva?
- Prosjektmål: Hvem skal lære?
- Vanlig arbeidsdag for hvem?
- Effektmål: Hva er det bedriften ønsker rent overordnet? Mer fornøyde med ansatte, redusere driftskostnader etc. Hvorfor ønsker vi resultatet?
- Læringsmål: NTNU sine læringsmål for bacheloroppgaven, evt konflikt mellom bedrift og NTNU
- Frontend rammeverk kan flyttes rett inn i hovedrapporten
- Avgrensning: Velge å opprettholde arbeidsflyten, eller lage en ny arbeidsflyt (frister etc.)
- Valg av metodikk: anbefaler i praksis at vi kaller det SCRUM? - ikke fremstille alternativene som opplagt mye dårligere enn de er. Fremstille det hele litt mer balansert. Kan være en lur løsning og si "vi tar elementer"
- Kvalitetssikring: Liste i prosjektplan
- Risikoanalyse: Gjennomførbarhetsanalyse - hvordan skal vi lykkes? Underbygge i tekst. Ikke helt definert hva vi må levere på, mulighet for nedskalering, litt mer grunnlag for hvorfor ting er lite sannsynlig at skjer.
    - Tabell må støttes av tekst, kompetanse etc
    - Risikoer hver for seg i tekst
    - Tabell oppsummerer alt
    - Vanlig praksis: Hva er risikofaktoren? Må ikke ha tiltak på alt
    - Litt overordnede
- Tidsplan: Spre forbedringsbolk litt utover i prosjektet slik at man kan vise forbedring til brukeren
- Prosjektplanen er et levende dokument
- Hen er innafor

29.01.2021 - Internmøte

- Gjennomgang av tilbakemeldinger fra møtet med Rune
- Satt opp forslag til introduksjon
- Neste uke blir planlagt

01.02.2021 - Internmøte

- Satt opp frontend repo
- Statusoppdateringer

02.02.2021 - Internmøte

- Diskusjon web app/function app

03.02.2021 - Veileder NTNU

Tilbakemelding på kravspesifikasjoner:

- Beskrive automatisk tildeling som en prosess (tegn piler mellom prosessene)
- Hvordan endre på informasjon om utleieobjektene (bør være et eget use case, og ikke hardkodet)
- Tankekors: Noe som viser større utfordringer enn hva som er gjennomsnittet
- Effektmål og læringsmål ser bra ut
- Wire frame for prototyper? Diskuter med oppdragsgiver
- Prøv og være kreative og aktive

05.02.2021 - Communicate

- Gjennomgang av eksempelprosjekt spesielt fokus på kommunikasjon mellom lag
- Planlagt å gå gjennom arkitektur tirsdag med veileder
- Veien videre: dybdeintervju eller spørreundersøkelse mtp feedback fra bruker

16.02.2021 - Communicate

- Gjennomgang av logg inn problemer
- Database diskusjon - tables
- pipelines - replace tokens ved rent angular prosjekt

18.02.2021 - Veileder NTNU

Hovedpunker:

- Gjennomgang av sekvensdiagram for logg inn
- Gjennomgang av komponentmodell

Tilbakemeldinger:

- Prosess/produkt skriving
- Overodnet systemarkitektur og flyten mellom komponentene hver for seg
- Rapporten skal skrives på en måte slik at utenforstående forstår
- Kommentere dersom noe er standard, mer hensiktsmessig å bruke original funksjoner i slike tilfeller
- Få på plass ende til ende tidlig
- Viktig å få ting ferdig 100 % før man går videre til neste

01.03.2021 - Internmøte

- Status
- Evaluering av perioden
- Torbjørn fortsetter på endepunkter fra forrige periode

02.03.2021 - Communivate

- Tilbakemelding på minimumsløsning:
- Bruke hele skjermen på desktop
- Link til reglement under bekreftlese
- Umiddelbar visuell kalender
- Prøve å forhindre mye scrolling opp og ned
- Kan komme nye retningslinjer for design fra communicate
- Hvordan partisjonere må vurderes mtp responstid

04.03.2021 - Veileder NTNU

- Status
- Viktig å gjøre justeringer i forhold til estimatet som ble gjort tidlig
- Endring i API kan påvirke epost
- Innholdsfortegnelse er forvirrende. Forslag: Alt design i eget kapittel, alt av implementasjon i eget kapittel.
- Enten gå mer i dybden detaljmessig. Eller mer overordnet.
- Diskutere ekstra lag, mye mer detaljert på arkitektur og struktur
- Pass på at dobbelapostrof
- Se på microservices som en kombinasjon
- Tegne API håndtering som en brannmur

- Husk å skrive i bildeteksten hvor vi har hentet den fra

09.03.2021 - Communicate

- Beholde booking data for å føre statistikk på det senere, alternativt er å flytte data til en blob-storage
- Flytte det som ikke blir allokert til blob-storage
- Admin må kunne blokkere en tidsperiode (f.eks. vedlikehold)
- Instillinger for admin, blokkere daterange med kommentar.
- Oversikt over norske helligdager,-> legg til helligdager (ferieobjekt.
- Bestilling gir fratrekk i poeng, -> kan endre seg
- booking@communicate.no
- Se opp for at noe innformasjon ikke kan tømmes/slettes (branninfo etc)
- Logikk som sier hva en standard leieperiode for et objekt
- Dette justeres i feriene.
- Dobbeltsjekker egne avtaler

16.03.2021 - Communicate

- Gjennomgang av svar fra forrige møte
- Nye regler kommer på bordet :)))))
- Hvor langt frem i tid kan man søke?

23.03.2021 - Communicate

- Layout fra selskapet
- Direktelink til hovedsider fra på alle mindre sider (mine bookinger)
- Bruke font-awesome til ikoner
- venstresentrering av sjekkliste
- Faner er egne valg - egne sider i en venstreMeny

06.04.2021 - Communicate

- Svar på bugs
- Undersøke brukertesting

08.04.2021 - Veileder NTNU

- Lage skjema for brukertesting (systematisere)
- Modellene ser bra ut
- 5.1 figuren fungerer, bør beskrives godt i klartekst
- diskusjon av styrker og svakheter i systemet, og prosessen
- Teststrategi bør dokumenteres

- Samlet bilde på codeReview - lære på et mer generelt nivå

12.04.2021 - Internmøte

- Evaluering av arbeid
- planlegging neste periode

13.04.2021 - Communicate

- bruke AD for admin bestemmelser

20.04.2021 - Communicate

- Prioriter tilbakemeldinger
- Få på plass wiki side for de som skal ta over systemet

22.04.2021 - Veileder NTNU

Gjennomgang rapport

- Brukertesting må dokumenteres
- Code review og brukertesting inn i prosess
- Vise ulike komponenter i Azure i en figur
- Litt kjapt forbi argumentasjon - begrunne valg med konsekvenser av alle alternativer
- Se på vurderingskriteriene i emnet
- Trenger vi å lagre ansettelsesår?
- Brukere må kunne slettes

03.05.2021 - Intern evaluering

- Gjennomgang av arbeid
- Planlegging og oppstart rapportskriving
- Sindre fullfører påbegynt oppgave i UI

06.05.2021 - Veileder NTNU

Gjennomgang rapport

- klassedigram i design for å underbygge implementering
- hva er rett frem og opplagt, trekker frem utfordringer
- implementasjon er valg som ikke er diskutert i design
- Viktig og ikke gjenta seg selv: inkonsistent tekst, gjenta med mindre deltajer - referere

- Diskutere exceptions vs retur verdi

12.05.2021 - Veileder NTNU

Gjennomgang rapport:

- engelske uttrykk
- utdype prioritering av testing (tydeliggjøre unit-testing)
- Mock-rammeverk (skrive i implementasjon)
- Få frem brukertesting
- Konklusjon

18.05.2021 - Veileder NTNU

Gjennomgang rapport:

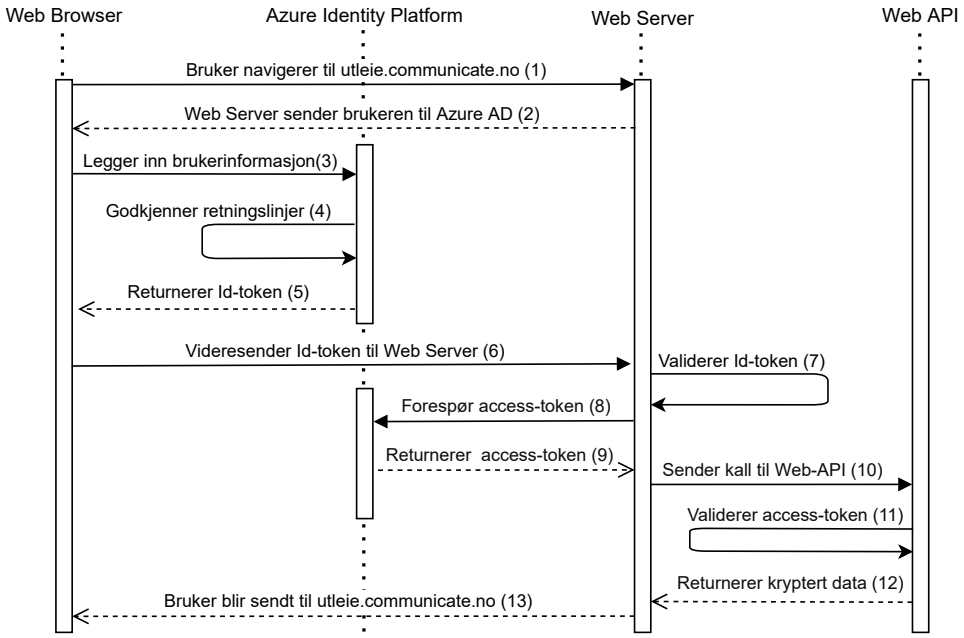
- konklusjon -> kvalitet
- beskriv hele utviklingsmodell-valget
- Utviklingsperioder er uinteressant -> hovedtrekk -> vedlegg
- Use Case -> MSAL ekstern aktør
- Use Case -> automatiserte prosesser
- Skrive mer om hva som skjedde med tilbakemeldingene fra brukertesting i drøfting og konklusjon
- bookinger langt frem i tid -> få communicates kvittering
- Kommentere/fylle sjekkliste
- hva er de viktigste føringene vi har lagt for testdesign Ferdig
- caching -> utdype diskusjon
- klassediagram -> backend implementasjon
- testing implementasjon -> mer utfyllende evt referanse -> si mer om dekningsgrad

Vedlegg J

Arbeidsfigurer

Dette er figurer med beskrivelser som er laget underveis, enten for å øke egen forståelse, eller som ble laget til oppgaven som endte opp uten en naturlig plassering.

J.1 Innlogging med Azure Active Directory



Figur J.1: Sekvens for suksessfull innlogging via Azure AD, i stor grad basert på Microsoft (2020*i*)

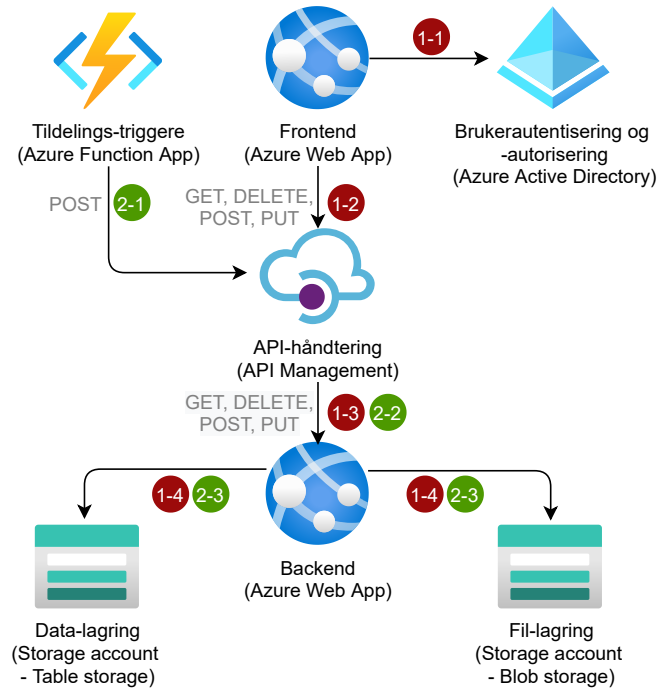
1. Nettleser sender et kall til Web server om at en bruker ønsker å besøke utleie.communicate.no
2. Web Server benytter redirect URI, og sender brukeren til Microsoft Innlogging via Azure AD og OAuth2. Privat kommunikasjon mellom bruker og Identity plattformen blir nå opprettet. Det vil si at Web API, ikke vet at dette foregår.
3. Bruker autentiserer seg selv gjennom å legge inn brukernavn og passord

for sin Microsoft konto tilknyttet Communicate
4. Bruker godkjenner at Azure Ad leser brukerinformasjon
5. Azure AD returnerer id-token for brukeren. Dette innebærer flere viktige data (Microsoft 2020*f*):
    - En signatur fra Azure AD, det er dermed ingen andre som har opprettet et falskt Id-token
    - Tidsstempel for når det ble utsendt, og når denne Cookie-Session ikke lenger er gjeldende
    - oid - en unik Id for brukeren innenfor denne applikasjonen
    - tid - unik nøkkel som viser at denne nøkkelen er ment for denne applikasjonen
6. Nettleseren videresender Id-token til Web Server
7. Web erver validerer Id-token
8. Web server forespør access-token i Azure Key-Vault
9. Microsoft Key-Vault genererer og returnerer access-token til Web server. Et access-token er en ikke-lesbar nøkkel som genereres av Id-token, og brukes for autorisering i Web API.
10. Web server videresender dette mot autorisering i Web APIet
11. Web API gjennomfører både validering av access-token og autorisering (sammenfallende prosess)
12. Web API returnerer beskjed om at brukeren er autorisert (logget inn), og kryptert informasjon angående brukeren (f.eks. rettigheter: admin)
13. Bruker blir sendt til forespurt side; utleie.communicate.no.

Sekvensdiagrammet (figur J.1) gir en oversikt over kommunikasjonen dersom en innloggingsprosess gjennomføres slik som den er ment å gjøre. Her er det viktig å understreke at operasjonen kan gi feilmelding på flere ulike punkter. Det mest åpenbare er feilkontroll av mailadresser som brukes til innlogging. Potensielt kan det oppstå feilmelding også ved retur av Id-token (f.eks. dersom bruker har en valid microsoft-konto som ikke tilhører Communicate). Dersom session-cookie har utgått under punkt 7, kan brukeren bli sendt tilbake til punkt 3. Validering av access-token er også et mulig bruddpunkt (spesielt dersom man prøver å generere et falskt access-token på egenhånd) (Microsoft 2020*e*).

J.2 Flyten mellom ulike Azure komponenter i produksjon

En illustrasjon over hvordan flyten mellom de ulike Azure-komponentene som benyttes kan sees i figur J.2. Figuren har overlapp med arkitekturmodellen i figur 5.1, hvor frontend, backend og lagring er det samme. I tillegg inneholder denne figuren API-håndtering og function apps som ikke er vist som en del av arkitekturmodellen. Gruppa regner ikke disse som en del av hovedarkitekturen,



Figur J.2: Flyten mellom de ulike Azure-komponentene

da de har en rolle mer som støttekomponenter. API-hånteringen legger til et sikkerhetsnivå, mens function appen gjør det enklere å trigge kall basert på tid.

Figur J.2 viser to hovedflyer - flyt 1 (markert 1-x i rødt i figuren) og flyt 2 (markert 2-x i grønt). Flyt 1 er flyten som følges når en bruker benytter nettsiden. Flyt 2 viser flyten for de automatiske triggerene for tildeling av utleieobjekter før oppstart av ny utleieperiode.

Flyt 1 Denne flyten trigger når en bruker åpner nettsiden.

- 1-1: Første steg er å autentisere brukeren. Dette skjer gjennom et REST-kall fra frontend som benytter Microsoft Authentication Library for Angular (forkortet til msal) til Azure Active Directory, hvor det er definert at applikasjonen kun skal tillate innlogging fra brukere som tilhører Communicate. Ved godkjent innlogging sendes brukerinformasjon i retur til frontend.
- 1-2: Etter at brukeren er autentisert gjør frontend et REST-kall til API-håndteringskomponenten. API-håndteringen er konfigurert til å kun videreføre kall fra godkjente kilder og brukere, slik at applikasjonen som utfører kallet må være autentisert mot komponenten.
- 1-3: API-hånteringen viderefører REST-kallet til backend.
- 1-4: Backend henter nødvendig informasjon fra databasene.
- Deretter sendes naturlig nok den hentende informasjonen som respons fra backend til API-håndteringen, og deretter tilbake til frontend.

Flyt 2 Det er satt opp en Function App som er konfigurert til å trigge en gang hver dag, når denne trigges starter denne flyten.

- 2-1: Function appen består av en timer-trigger som sender et kall til en controller i backend.
- 2-2: Etter at API-håndteringen har sjekket at kallet er fra en godkjent kilde, videresender det kallet til backend.
- 2-3: Controlleren utfører logikk basert på dagens dato.
- Dersom en funksjon trigges, og det oppstår feil vil feilen logges.

Hansen, Hjelle, Næsvold

Internutleie - Communicate

NTNU
Norwegian University of
Science and Technology