

Henriette Marie Eltvik
Alf Magnus Graathen Nohr
Sigrid Warberg Vestheim

Informasjonsplattform for ansatte

Bacheloroppgave i Bachelor i ingeniørfag - data og Bachelor i programmering
Veileder: Rune Hjelsvold
Mai 2023

Henriette Marie Eltvik
Alf Magnus Graathen Nohr
Sigrid Warberg Vestheim

Informasjonsplattform for ansatte

Bacheloroppgave i Bachelor i ingeniørfag - data og Bachelor i
programmering
Veileder: Rune Hjelsvold
Mai 2023

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



Kunnskap for en bedre verden

Abstract

Evidi wanted an employee portal that presents information to the employees and provides functionality to reserve a desk in the office, as attendance has been variable. The solution is a web application where the user can view their meetings and all staff birthdays for the current month. Additionally, it is possible to reserve a desk and view others' reservations in a graphical representation. The web application requires login via a Microsoft account through Evidi as the organization. The solution also includes a backend API written in .NET. The software is automatically deployed in Microsoft Azure using automated build and release. Throughout the project, we have focused on professional development methodologies, utilizing a combination of both Scrum and Kanban.

Sammendrag

Evidi ønsket en ansattportal som presenterer informasjon for de ansatte samt funksjonalitet til å reservere en plass på kontoret da det har vært variabelt med oppmøte. Løsningen er en webapplikasjon hvor brukeren ser egne møter og alle personalbursdager for nåværende måned. I tillegg er det mulig å reservere seg en plass på kontoret og se andre sine reserveringer i en grafisk fremstilling. Webapplikasjonen krever innlogging via en Microsoft-konto gjennom Evidi som organisasjon. Løsningen inneholder også et API i backend skrevet i .NET. Programvaren blir automatisk publisert i Microsoft Azure ved hjelp av automatisk bygg og publisering. Gjennom prosjektet har vi hatt fokus på profesjonell utviklingsmetodikk ved bruk av en kombinasjon av både Scrum og Kanban.

Forord

Denne bacheloroppgaven er skrevet ved Institutt for datateknologi og informatikk ved NTNU i Gjøvik av Henriette Eltvik, Alf Magnus Graathen Nohr og Sigrid Warberg Vestheim.

Først og fremst ønsker vi å rette en hjertelig takk til vår veileder Rune Hjelsvold for hans støtte, veiledning og faglig innsikt gjennom prosjektet. Vi ønsker også å takke vår oppdragsgiver Evidi, spesielt våre veiledere Felix Leenborg Anthonessen, Abdirahman Hassen og Jørgen Mobakken for utmerket oppfølging og tilgjengelighet.

Til slutt vil vi takke hverandre for et godt samarbeid gjennom prosjektet. Prosessen har ført til mange gode diskusjoner som har bidratt til en givende og lærerik prosess.

Innhold

Abstract	iii
Sammendrag	v
Forord	vii
Innhold	ix
Figurer	xiii
Tabeller	xv
Kodelisting	xvii
Akronymer	xix
Ordliste	xxi
1 Introduksjon	1
1.1 Bakgrunn	1
1.2 Oppgavebeskrivelse	1
1.3 Mål og rammer	2
1.3.1 Resultatmål	2
1.3.2 Effektmål	2
1.3.3 Læringsmål	2
1.4 Avgrensning	3
1.5 Brukergrupper	3
1.5.1 Brukere av programmet	3
1.5.2 Videre utviklere av plattformen	3
1.5.3 Brukere av rapporten	3
1.6 Gruppens bakgrunn	3
1.7 Prosjektorganisering	4
1.7.1 Ansvarsforhold og roller	4
1.8 Om rapporten	5
2 Utviklingsprosess	7
2.1 Utviklingsmodell	7
2.1.1 Kanbanbrett	7
2.2 Møter	8
2.2.1 Interne møter	8
2.2.2 Veiledningmøter	9
2.2.3 Møter med oppdragsgiver	9
2.3 Git	9
2.3.1 Git commits	10
2.3.2 Pull request	11
2.4 Gjennomførelse av sprinter	11
2.4.1 Sammendrag av sprinter	11
3 Kravspesifikasjon	15
3.1 Funksjonelle krav	15
3.1.1 Brukermønster	15
3.2 Ikke-funksjonelle krav	18
3.3 Operasjonelle krav	19

4	Teknologier	21
4.1	Microsoft Azure	21
4.1.1	Azure Static Web App	21
4.1.2	Azure Function	21
4.2	Frontend	21
4.2.1	React	21
4.2.2	Next.js	22
4.2.3	Microsoft Graph	22
4.2.4	Microsoft Graph Toolkit	23
4.2.5	MUI	24
4.2.6	Chart.js	25
4.2.7	Cookie-cutter	26
4.3	Backend	26
4.3.1	Programmeringsspråk	26
4.3.2	Timer Trigger	26
4.4	Database	26
4.5	Testing	27
4.6	Sikkerhet	27
4.6.1	Azure AD	27
4.6.2	Microsoft Identity Platform	27
4.6.3	MSAL	27
4.6.4	Key Vault	28
4.6.5	API token	28
4.7	JSON og REST	28
5	Design	29
5.1	Systemdesign	29
5.2	Backend	30
5.2.1	Filstruktur backend	31
5.3	Brukergransesnitt	31
5.3.1	Skisser	31
5.3.2	Resultat	33
5.3.3	Filstruktur frontend	38
6	Implementasjon	39
6.1	Backend	39
6.1.1	Azure Function	40
6.1.2	Azure SQL	41
6.1.3	Modeller	41
6.1.4	Services	42
6.1.5	Endepunkter	43
6.1.6	Timer Trigger	44
6.1.7	Feilhåndtering	45
6.2	Frontend	46
6.3	Innloggingsside	46
6.3.1	Innloggingsflyt	46
6.3.2	Oppsett	47
6.3.3	Henting av token	48
6.3.4	Autorisering	49
6.3.5	Opprettelse av bruker	49
6.3.6	Logg ut	53
6.3.7	Navigasjonsmeny	53
6.4	Hjemmeside	55
6.4.1	Møter / arrangement	56

6.4.2	Bursdager denne måneden	58
6.4.3	Mine bookinger	59
6.5	Bookingsiden	61
6.5.1	Slette egne bookinger	62
6.5.2	Dato	63
6.5.3	Visuell presentasjon av bookinger	64
6.5.4	Se andre sine bookinger	66
6.5.5	Reserver en plass	67
6.6	Kommunikasjon mellom frontend og backend	68
6.7	Overgang til Next.js	69
6.7.1	Refaktorering	69
6.7.2	Bruk av miljøvariabel	69
6.7.3	Server-komponent	71
7	Utviklingsmiljøer og prosjektstyringsverktøy	73
7.1	Utviklingsmiljøer	74
7.1.1	Frontend	74
7.1.2	Backend	74
7.1.3	Git	75
7.1.4	Azure DevOps	75
7.2	Prosjektstyringsverktøy	76
7.2.1	Timeføring	76
7.2.2	Kommunikasjonsplattformer	77
8	Kvalitetssikring	79
8.1	Kodekvalitet	79
8.1.1	Dokumentasjon	79
8.2	Code Review	81
8.2.1	Parkoding	82
8.3	Testing	82
8.3.1	Integrasjonstesting	82
8.3.2	Brukertesting	83
9	Produksjonssetting	85
9.1	Bicep	86
9.1.1	Azure Static Web App	86
9.2	CI/CD	86
9.2.1	Pipelines	87
10	Diskusjon	89
10.1	Utviklingsprosess	89
10.2	Møter	90
10.3	Manglende funksjonalitet	90
10.4	Brukergrensesnitt	90
10.5	Utviklingsmiljøer	91
10.6	ChatGPT	91
10.7	Løsning	92
10.8	Evaluerer av gruppens arbeid	93
11	Konklusjon	95
11.1	Måloppnåelse	95
11.2	Videre arbeid	96
	Bibliografi	97
A	Prosjektplan	101
B	Møtereferat	115
C	Ukelogg	127
D	Brukertest skjema	133

E Brukertest resultat	141
F ChatGPT	153
G Timeføring	157
H Brukerguide	161

Figurer

1.1	Organisasjonskart (laget i lucid.app).	4
2.1	Skjerm bilde av kanbanbrettet for sprint 6.	8
2.2	Skjerm bilde tatt fra Azure Boards av <i>Retrospective</i> for sprint 4.	9
2.3	Regler for Git <i>commits</i>	10
2.4	<i>Pull request</i> : koblet <i>commits</i>	11
2.5	En aktiv <i>pull request</i>	11
3.1	Diagram for brukermønster (laget i lucid.app).	16
4.1	Struktur for å opprette home endepunkt.	22
5.1	Applikasjonens arkitektur (laget i lucid.app).	29
5.2	Konseptuell modell (laget i lucid.app).	30
5.3	Filstruktur backend.	31
5.4	Skisse av hjemmesiden (laget i lucid.app).	32
5.5	Skisse av bookingsiden (laget i lucid.app).	32
5.6	Flytskjema for applikasjonen (laget i lucid.app).	34
5.7	Hjemmesiden i webapplikasjonen.	35
5.8	Skjerm bilde av bookingsiden.	36
5.9	Fargekoder blå nyanse.	37
5.10	Fargekoder grønn nyanse.	37
5.11	Fargekoder oransje nyanse.	37
5.12	Filstruktur frontend.	38
6.1	Klassediagram av bookingfunksjonalitet (laget i lucid.app).	39
6.2	Klassediagram av ansattfunksjonalitet (laget i lucid.app).	40
6.3	Klassediagram av bursdagfunksjonalitet (laget i lucid.app).	40
6.4	Sekvensskjema for feilhåndtering av handlinger (laget i lucid.app).	45
6.5	Innloggingsiden til applikasjonen.	46
6.6	Sekvensskjema for innloggingsfunksjonalitet (laget i lucid.app).	47
6.7	Velg bursdag.	51
6.8	Navigasjonsmenyen, skjerm bilde fra webapplikasjonen.	53
6.9	Kodestruktur hjemmeside (laget i lucid.app).	55
6.10	Møter / arrangement.	56
6.11	Bursdagskomponenter, skjerm bilde tatt 20.april 2023.	58
6.12	Liste med mine bookinger.	59
6.13	Klassediagram til bookingsiden (laget i lucid.app).	61
6.14	Slette bookinger.	62
6.15	Datovelger.	63
6.16	Visuell presentasjon av bookinger.	64
6.17	Andre kollegaer sine reservasjoner.	66

6.18	Visuell presentasjon av bookinger.	67
6.19	Application Programming Interface (API)-endepunkter (fra swagger).	68
6.20	Bruk av <i>pages</i>	69
6.21	Før Next.js.	70
6.22	Etter Next.js.	70
7.1	Utviklingsmiljø og prosjektstyringsverktøy (laget i lucid.app).	73
7.2	Endepunkter i Swagger.	74
7.3	Oppbevaringssted/repo grener (laget i lucid.app).	75
7.4	Skjermbilde fra track.toggl.com.	76
8.1	JSDoc HomeBooking.	80
8.2	JSDoc forhåndsvisning kode.	80
8.3	Code Review for <i>pull request</i>	81
9.1	Ressurser i Azure (laget i lucid.app).	85
9.2	Pipelines i Azure DevOps.	87
9.3	Jobber i pipeline.	88
9.4	Oversikt over pipeline.	88
C.1	Timelogg uke 2.	127
C.2	Timelogg uke 3.	127
C.3	Timelogg uke 4.	127
C.4	Timelogg uke 5.	128
C.5	Timelogg uke 6.	128
C.6	Timelogg uke 7.	128
C.7	Timelogg uke 8.	128
C.8	Timelogg uke 9.	128
C.9	Timelogg uke 10.	129
C.10	Timelogg uke 11.	129
C.11	Timelogg uke 12.	129
C.12	Timelogg uke 13.	129
C.13	Timelogg uke 14.	129
C.14	Timelogg uke 15.	130
C.15	Timelogg uke 16.	130
C.16	Timelogg uke 17.	130
C.17	Timelogg uke 18.	130
C.18	Timelogg uke 19.	131
C.19	Timelogg uke 20.	131
F.1	Spørsmål localStorage	153
F.2	Chatgpt	154
F.3	Chatgpt	155
F.4	Split name	155
H.1	Brukerguide.	161

Tabeller

3.1	Brukermønster 1: Logg inn.	17
3.2	Brukermønster 2: Informasjon om innlogget bruker.	17
3.3	Brukermønster 3: Legge til egen bursdag.	17
3.4	Brukermønster 4: Se bursdager.	17
3.5	Brukermønster 5: Se egne møter/arrangement.	17
3.6	Brukermønster 6: Se egne bookinger.	17
3.7	Brukermønster 7: Visuell representasjon av bookinger.	18
3.8	Brukermønster 8: Booke kontorplass.	18
3.9	Brukermønster 9: Se andre sine bookinger.	18
3.10	Brukermønster 10: Slette bookinger.	18
3.11	Brukermønster 11: Logg ut.	18

Kodelisting

1	Person-komponent fra Microsoft Toolkit.	23
2	Agenda-komponent fra Microsoft Toolkit.	24
3	<i>LocalizationProvider</i> og <i>DatePicker</i>	24
4	<i>Dialog</i> sammen med <i>DialogContent</i> og <i>DialogActions</i>	25
5	Stolpediagram fra <i>Chart.js</i>	25
6	Kodeliste som viser bruk av <i>Fact</i> og <i>Theory</i> attributtene.	27
7	Bruk av miljøvariabel.	28
8	Unikt funksjonsnavn.	40
9	<i>Connection string</i>	41
10	Modell for henting av reservasjoner.	41
11	SQL-uttrykk for henting av mine bookinger.	42
12	JavaScript Object Notation (JSON)-respons fra <i>GetMyBooking</i>	43
13	Funksjon for henting av data.	43
14	Funksjon for å legge til data.	44
15	Kodeliste på <i>Timer Trigger</i>	44
16	Konfigurasjonsfilen i programvaren.	47
17	Henting av tilgangstoken ved hjelp av <i>Microsoft Authentication Library (MSAL)</i>	48
18	<i>Authenticated-</i> og <i>UnauthenticatedTemplate</i>	49
19	Kodeliste for videresending av bruker når logget inn.	49
20	Setter fornavn og etternavn i informasjonskapsel.	50
21	Kode for visning av <i>pop-up</i> -vindu.	51
22	Kode for bruk av <i>localStorage</i>	52
23	Kode for logg ut-funksjonalitet.	53
24	<i>getInfoUser.js</i> : Hente data fra <i>Microsoft Graph</i> om brukeren.	53
25	Noe av resultat fra <i>Microsoft Graph</i>	54
26	<i>NavBar.js</i> : Personkomponentet i navigasjonsmenyen.	54
27	<i>getMeetings.js</i> : Henter data fra kalenderen ved hjelp av <i>Microsoft Graph</i>	56
28	<i>HomeMeetings.js</i> : Lager mal for hvordan vi vil presentere dataen i <i>Agenda-</i> komponentet.	56
29	<i>HomeMeetings.js</i> : <i>Agenda-komponentet</i>	57
30	<i>getBirthday.js</i> : viser hvordan vi kaller på API fra backend.	58
31	<i>bookingList.js</i> : lager en liste med bookinger som skal presenteres for brukeren.	59
32	<i>deleteBooking.js</i> : kaller på API-et for sletting av reservasjoner.	62
33	<i>SelectDay.js</i> : formater datoen.	63
34	<i>SelectDay.js</i> : implementering av datovelgeren.	64
35	<i>BookingGraph.js</i> : lager datasett basert på de sorterte listene.	65
36	<i>BookingGraph.js</i> : implementering av stolpediagrammet ved hjelp av <i>Graph.js</i>	66
37	<i>createBooking.js</i> : lager en reservasjon.	67
38	Lag en ny booking JSON- <i>body</i>	68
39	<i>MyBooking.js</i> i <i>React</i>	69

40	getMyBooking.	69
41	Home-serverkomponent.	71
42	Swagger-dokumentering i Azure Function.	79
43	Kodelifting for testing av GetMyBooking funksjon.	82
44	Bicep Functionapp.	86
45	Pipeline i YAML.	87

Akronymer

API Application Programming Interface. xiv, xvii, xxii, 1–3, 15, 21–23, 26–31, 38, 41, 46, 53–55, 58, 61, 62, 68–71, 79, 82, 85, 92

HTTP Hypertext Transfer Protocol. xxii, 31, 38, 40, 68

IDE Integrated Development Environment. 74

JSON JavaScript Object Notation. xvii, 28, 41–44, 58, 68

MSAL Microsoft Authentication Library. xvii, 27, 28, 48, 49

MUI Material-UI. 24, 25, 33, 74

SSO Single Sign On. 28

URI Uniform Resource Identifier. 28, 48

WIP Work in progress. 7

Ordliste

.NET et rammeverk for C#, F# og Visual Basic. 11, 26, 27, 74

Azure Boards er en tjeneste innenfor Azure DevOps som hjelper en gruppe med planlegging, sporing og diskutering av oppgaver gjennom en utviklingsprosess [1]. Verktøy som finnes er Kanban-brett, backlog og sprint . xiii, xxi, 8, 9

Azure DevOps er et Microsoft-produkt som dekker hele livssyklusen til en applikasjon, som Azure Boards og Azure Pipelines [1]. xiv, xxi, 1, 73, 75, 87

Azure Function serverløs løsning som håndterer kjøring av kode. Funksjonene er bare trigget når de skal bli brukt[2]. 21, 28, 30, 31, 40, 41, 44, 45, 82, 85, 86

Azure Static Web App tjeneste for automatisk bygg og release av kode[3]. 21

backend er den delen av programvaren som ligger nærmest databasen [4]. v, xi, xiii, xvii, 2, 4, 5, 8, 11, 12, 18, 21, 28–31, 39–41, 45, 49, 50, 52, 55, 58, 61, 67, 68, 74, 76, 79, 82, 87, 92, 93, 95, 96

backlog er en prioritert liste over funksjonalitet som et produkt skal inneholde. xxi, 9, 75

Bicep er et domene-spesifikt språk for å publisere Azure ressurser [5]. 86

C# er et objekt-orientert programmeringspråk som inneholder kjennetegn fra C, C++ og Java [6]. 11, 26

CSS er et format for å endre stiler for HTML-dokumenter, det vil si skrifttyper, farger og linjeavstand osv. [7]. 3, 21, 23, 46

frontend er den delen av programvaren som ligger nærmest brukeren [8]. xi, xiii, 4, 5, 7, 11, 12, 18, 21, 22, 26, 29, 38, 39, 41, 45–47, 52, 68, 74, 76, 79, 82, 85–87, 92, 93, 96

HTML er et markeringsspråk for å utvikle nettsider og støttes av alle vanlige nettlesere [9]. 3, 21, 25

JavaScript er det mest brukte programmeringsspråket for interaktive nettsider [10]. 3, 21, 25, 26, 46, 68, 74

Kanban er en prosjektledelsemetode som visualiserer arbeidsflyten på en tavle. v, xxi, 7, 89

Key Vault er en sky-tjeneste for å lagre og hente ut *secrets* på en sikker måte. 12, 28, 85, 92

localStorage lagrer data i nettleseren. xvii, 52, 91, 92

Microsoft Azure er en skyplattform med mer enn 200 produkter og tjenester, med blant annet infrastruktur, beregninger og databehandling i skyen [11]. v, 1–4, 11, 18, 21, 23, 73, 85, 86, 90, 91

RESTful er en stil som er laget for å designe og bruke et API som bruker Hypertext Transfer Protocol (HTTP)-forespørsler [12]. 2, 29, 30

Scrum er et iterativ og smidig rammeverk for utvikling av komplekse programvarer. v, 4, 7, 89

sprint er et bestemt tidsintervall der gruppen jobber med en definert mengde arbeid. Intervallet er vanligvis mellom 2 og 3 uker. xiii, xxi, 4, 7–9, 11, 13, 75, 89, 90

tilgangstoken er en sikkerhetslegitimasjon og identifiserer brukeren, som sier at brukeren har tilgang og hvilke rettigheter den har [13]. xvii, 12, 13, 27–30, 46–49, 53, 56, 69

xUnit er et åpent testingverktøy for .NET [14]. 27

YAML er et menneskelig lesbart data-serialiseringsspråk[15]. xviii, 87

Kapittel 1

Introduksjon

1.1 Bakgrunn

Evidi er et «nytt» selskap som er satt sammen av fem forskjellige selskaper. De er et konsultantselskap som leverer digitale løsninger til kundene sine. De har spesialisert seg i Microsoft sine teknologier og mener at styrken ligger i helheten. De er opptatte av å skape en god fremtid og satser dermed på bærekraft. Evidi har hovedkontoret sitt i Oslo, men man kan også finne kontorer i Halden, Sarpsborg, Asker, Tønsberg, Bergen og Danmark. På kontoret til Evidi har de et åpent kontorlandskap uten faste plasser. For å få en sitteplass på kontoret må man satse på at det er en ledig plass når man ankommer siden det ikke finnes noen oversikt over hvem som ønsker å jobbe på kontoret på en bestemt dag. Etter koronapandemien har det vært veldig variabelt med oppmøte på kontoret. Derfor ønsker selskapet å få til en løsning hvor man enkelt kan se hvor mange som skal bruke kontorlandskapet. Med 70 antall ansatte og et begrenset antall med kontorplasser, ønsker Evidi å utvikle en ansattportal der man har mulighet til å se ledig kapasitet på kontoret og booke seg en plass. I tillegg til dette ønsker de muligheten til å presentere felles informasjon som bursdager, møter og sosiale arrangement på en god måte for de ansatte.

1.2 Oppgavebeskrivelse

Oppdragsgiver ønsker en informasjonsplattform som kan samordne informasjon for de ansatte. Kommende møter, arrangement og personalbursdager skal hentes ut og gjøres tilgjengelig for de ansatte på en samlet plattform. I tillegg skal plattformen inneholde bookingfunksjonaliteter som gjør det mulig å reservere en plass i de forskjellige etasjene på kontoret. Denne plattformen skal publiseres i Microsoft Azure med følgende funksjonalitet:

- For å få tilgang til plattformen må brukeren først logge inn med eksisterende Microsoft-konto fra Evidi.
- Brukeren skal kunne booke seg en plass på kontoret for en bestemt dato og tid. Det skal være mulig å ha flere reserveringer for forskjellige datoer. Bookingen for en ny uke åpner søndag før den aktuelle uken begynner.
 - Ledige plasser og bookinger skal presenteres visuelt.
 - Det må være mulig å slette bookinger.
 - Brukeren skal kunne se andre sine reserveringer.
- En informasjonstavle skal være tilgjengelig med relevante møter og arrangement som oppdateres fra API-et Microsoft Graph. I tillegg skal personalbursdager vises for nåværende måned.

I tillegg skal vi bruke Azure DevOps til planlegging, kildekodekontroll og automatisert bygg og release.

1.3 Mål og rammer

For å kunne gjennomføre bacheloroppgaven, er det nødvendig å sette opp mål og definere rammer for prosjektet. I denne seksjonen blir det presentert hvilke mål og rammer som er satt for å møte problemstillingen. Disse er basert på prosjektplanen i vedlegg A.

Målene for prosjektet har vi valgt å dele inn i tre kategorier: resultatmål, effektmål og læringsmål. Resultatmål sier noe om hva prosjektet konkret skal oppnå og er dermed knyttet til prosjektet sine resultater og leveranser. Effektmål sier noe om målsettingen til prosjektet og grunnen til at vi jobber med denne oppgaven. Læringsmål beskriver hva vi som studenter forventer å lære av prosessen og oppgaven.

1.3.1 Resultatmål

I løpet av prosjektet skal det produseres en løsning som gjør det mulig for ansatte å booke en plass på kontoret samtidig som de skal få med seg felles informasjon. Målet er å lage et enkelt og intuitivt brukergrensesnitt som vil kommunisere med et RESTful API i backend. Resultatmålene for prosjektet er:

- Integre en innloggingside med eksisterende Microsoft-konto gjennom Evidi.
- Applikasjonen skal ta i bruk to eksterne tjenester; Microsoft Graph og Microsoft Identity Platform.
- Applikasjonen skal inneholde de spesifiserte funksjonalitetene nevnt i oppgavebeskrivelse 1.2.
- Presentere et intuitivt brukergrensesnitt.

1.3.2 Effektmål

Målet er at det skal være enklere for de ansatte å kunne se om det er ledig plass på kontoret og å kunne få med seg felles informasjon, som møter og bursdager. Resultatet av disse målene:

- Skal gjøre det mer effektivt for brukeren å bestille plasser.
- Det skal øke det sosiale samværet mellom de ansatte.

1.3.3 Læringsmål

Gjennom prosjektet ønsker vi å lære mest mulig om hvordan det er å arbeide smidig som team og å lære å bruke Microsoft Azure sine verktøy for utvikling av nye systemer. Vi vil fokusere på å:

- Ta i bruk smidig utviklingsmetodikk.
- Bruke Microsoft Azure og dens teknologier.
- Utvikle webapplikasjon med tilhørende backend.
- Lage intuitive brukergrensesnitt.
- Skaffe oss mer erfaring i testing og kvalitetskontroll av programvare.
- Automatisert bygg og release(CI/CD).

1.4 Avgrensning

- Selv om Evidi har flere kontorer rundt om i Norge, tar vi kun for oss ett kontor i Oslo som ligger på Helsefy.
- I prosjektet vil vi se nærmere på to eksterne tjenester; Microsoft Graph og Microsoft Identity Platform.
- Språket for applikasjonen vil være på engelsk.

1.5 Brukergrupper

1.5.1 Brukere av programmet

Dette er de direkte brukerne som skal ta i bruk programmet jevnlig. Programmet skal være tilpasset for alle ansatte ved Evidi og er derfor skrevet på engelsk. Programmet skal være enkelt å forstå og bruke, noe som gjør at de fleste ansatte kan ta i bruk funksjonaliteten så fort som mulig.

1.5.2 Videre utviklere av plattformen

Et av ønskene til Evidi, er at programmet skal være lett å sette seg inn i for nye utviklere som ønsker å videreutvikle programmet. Vi fokuserte på å dokumentere koden godt, slik at de neste utviklerne ikke skal bruke for lang tid på å sette seg inn i programmet.

1.5.3 Brukere av rapporten

Lesere vil være sensorer, ansatte hos Evidi eller studenter med lignende oppgaver. Innholdet vil dermed være tilpasset disse leserne og vi forventer at leserne vil forstå de fleste begrepene i rapporten. Spesifikke begrep brukt i prosjektet vil være forklart i teksten eller listet i ordlisten.

1.6 Gruppens bakgrunn

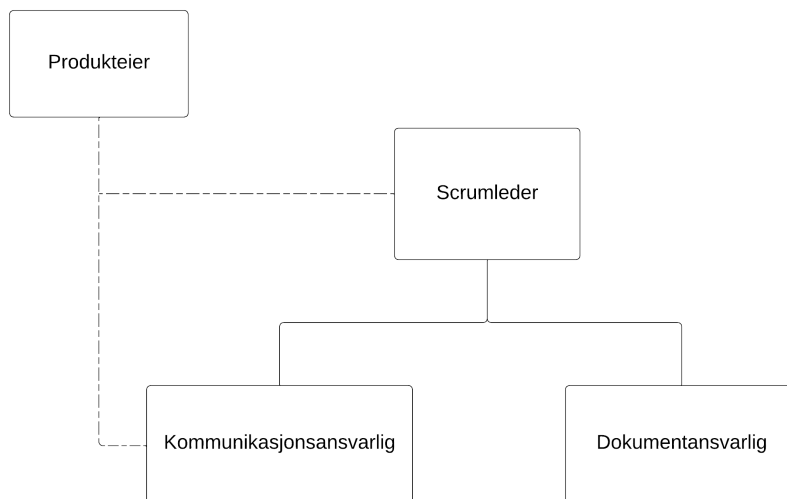
Gruppen er en tverrfaglig gruppe som består av tre studenter som kommer fra to ulike retninger. Alf studerer bachelor i programmering, mens Henriette og Sigrid studerer bachelor i ingeniørfag - data. Alle ved NTNU i Gjøvik. Vi har alle hatt flere av de obligatoriske fagene sammen. Det som skiller på de to linjene, er at dataingeniørene har flere real- og ingeniørfaglige emner enn programmeringsstudentene.

Gjennom studietiden har vi lært om programmering, databaser, algoritmer, utvikling av API, matematikk, informasjonssikkerhet og systemutvikling. I tillegg har vi tilegnet oss kunnskap innenfor design, planlegging, implementasjon og testing av programvare. Vi har lært lite om HTML, CSS og JavaScript, så dette er kunnskap vi må sette oss inn i. Vi har heller ikke vært borte i Microsoft Azure før. Med et godt grunnlag fra utdanningen, mener vi at vi skal ha gode sjanser på å sette oss inn i ny teknologi for å takle denne oppgaven, samt videre arbeidsliv.

1.7 Prosjektorganisering

For å kunne effektivisere arbeidet i gruppen, har vi definert gitte roller og ansvarsområder som definert i 1.7.1.

1.7.1 Ansvarsforhold og roller



Figur 1.1: Organisasjonskart (laget i lucid.app).

- **Produkteier** - Jørgen Mobakken
 - Representant for Evidi. Under møtene som ble avholdt hver andre uke, på slutten av hver sprint, kunne han komme med innspill om Evidi sine visjoner og ønsker.
- **Scrumleder** - Alf Magnus Graathen Nohr
 - Ansvarlig for progresjon og at utviklingen ble utført i samhold med forutsatte krav.
 - Håndterte konflikter og iverksatt tiltak hvis nødvendig.
 - Innkalte og ledet Scrum-relaterte møter som sprint-planlegging, review og *retrospective*.

Siden vi er en liten gruppe, kan noen av oppgavene til scrum master sees på som typiske prosjektlederoppgaver.

- **Dokumentansvarlig** - Sigrid Warberg Vestheim
 - Fulgte opp dokumenter og dokumenterte møter.
 - Organiserte utvikling av brukerguide og fulgte opp dokumentasjon av systemet.
- **Kommunikasjonsansvarlig med veiledere** - Henriette Marie Eltvik
 - Sendte ut møteinnkallelser og agenda for møtene til både produkteier, veileder og gruppen internt.

I tillegg til internrollene som vist i figur 1.1, tok alle gruppemedlemmene på seg roller som utviklere for å bidra med utviklingen ut fra utviklingsmodellen beskrevet i seksjon 2.1. For å sikre fremgang i utviklingen, fordelte vi ulike ansvarsområder. Alf tok hovedansvaret for infrastrukturen og publisering i Microsoft Azure. Henriette tok hovedansvaret for backend, refaktorering av kode og autorisering gjennom Microsoft. Sigrid tok hovedansvaret for design og implementering av frontend, samt integrering av Microsoft Graph.

1.8 Om rapporten

Rapporten er strukturert for å lese fra start til slutt, men enkeltkapitler kan også leses individuelt. Selv om rapporten er skrevet på norsk, vil den inneholde noen engelske begrep. Grunnen til dette er at det ikke alltid finnes gode norske begrep for det engelske begrepet.

Rapporten sin struktur:

1. Introduksjon

- Presenterer bakgrunn for oppgaven, oppgavebeskrivelsen og mål og rammer for prosjektet. I tillegg presenteres gruppens bakgrunn og prosjektorganisering.

2. Utviklingsprosess

- Introducerer valgt utviklingsmodell, samt hvordan vi har jobbet gjennom prosjektet. Her presenteres ulike typer møter og bruken av Git. Sluttet av kapitlet inkluderer et sammendrag av sprintene som ble holdt.

3. Kravspesifikasjon

- Presenterer kravene for løsningen vår og brukermønster.

4. Teknologier

- I kapitlet blir de ulike teknologiene som er brukt i prosjektet beskrevet.

5. Design

- Viser en oversikt over designet til systemet. I tillegg presenteres en beskrivelse på valg av filstruktur og design i selve programvaren. Skisse og resultat av brukergrensesnittet er også presentert.

6. Implementasjon

- Presenterer hvordan backend og frontend er implementert. Kapitlet gir en oversikt over alle komponentene i systemet.

7. Utviklingsmiljøer og prosjektstyringsverktøy

- Inneholder en beskrivelse av hvilke miljøer og verktøy vi har brukt i løpet av prosjektet og hvordan disse er brukt.

8. Kvalitetssikring

- Inkluderer en beskrivelse på hvordan vi har forsikret oss om at koden i programmet er av god kvalitet.

9. Produksjonssetting

- Beskriver hvordan vi har produksjonsatt applikasjonen ved hjelp av CI/CD.

10. Diskusjon

- I kapitlet diskuteres ulike deler av løsningen, utviklingsprosessen og valg underveis.

11. Konklusjon

- Konkluderer om løsningen svarer på problemstillingen. Videre arbeid blir også presentert.

Kapittel 2

Utviklingsprosess

I prosjektplanen, vedlegg A, er det beskrevet hvordan utførelsen av prosjektet skal være. Dette kapitlet bygger videre på denne planen, samtidig som det oppsummerer gjennomføringen.

2.1 Utviklingsmodell

Kjennetegn ved prosjektet som var avgjørende for valg av modell

- Funksjonaliteten til programmet skulle bestemmes underveis
- Åpen avgrensning
- Oppdragsgiver ønsker å være involvert i prosessen

Argumentasjon for valget av utviklingsmodell

Oppdragsgiver hadde få krav til sluttproduktet og ga inntrykk av at de var veldig fleksible med tanke på funksjonalitet og fremgangsmåte. Dette førte til at vi måtte velge en utviklingsmodell som gjorde det mulig å legge til oppgaver fortløpende når nye ønsker om funksjonalitet kom frem.

Vi så derfor på Kanban som fokuserer på kontinuerlig arbeid uten fastsatte tidslinjer. Fordelen med Kanban er at man kan legge til *issues* og utføre oppgaver fortløpende etter prioritet. Det settes en Work in progress (WIP)-grense slik at arbeidsmengden per person ikke blir for stor i forhold til kapasiteten. Ulempen med en slik modell er at det er lite struktur da det er fri flyt for å ta på seg oppgaver. Det er heller ingen fastsatte tidsrammer da man leverer funksjonalitet kontinuerlig.

En annen modell vi så på, var Scrum som har fastsatt lengde på sprintene. Her blir oppgavene fastsatt for hver sprint slik at det blir lettere å definere og jobbe frem mot et felles mål. Ulempen er at tidslinjen kan bli stram, men det holdes regelmessige møter som kan hjelpe og løse problemer på en rask og effektiv måte.

For å få det beste fra både Kanban og Scrum valgte vi å bruke en kombinasjon mellom disse to. Vi tar i bruk det typiske kanbanbrettet der vi har mulighet til å legge til oppgaver underveis i sprinten. Fra Scrum bruker vi sprint med fastsatt lengde på to uker.

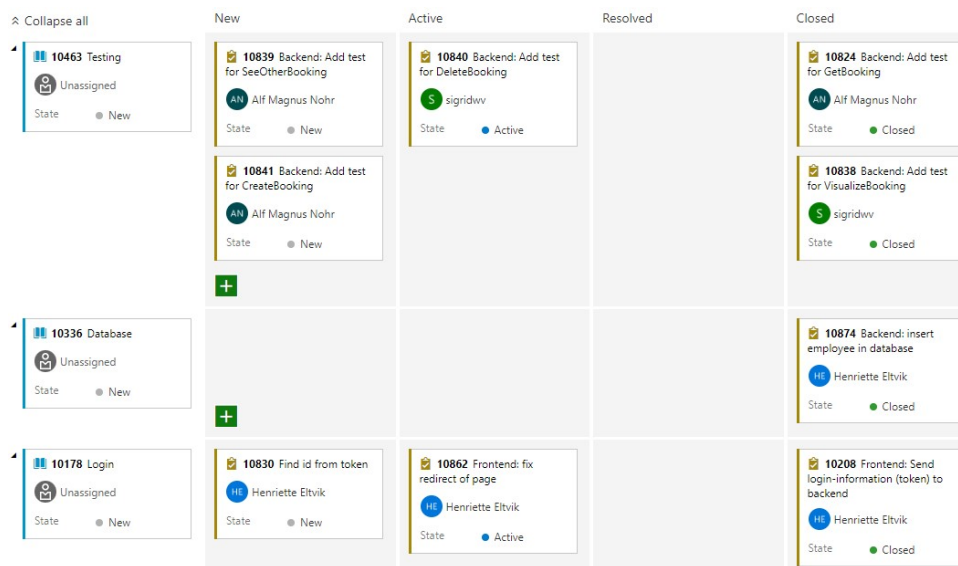
2.1.1 Kanbanbrett

Hver rad består av en *user story*, mens kolonnene representerer forskjellige stadier i prosessen. Disse kolonnene er *New*, *Active*, *Resolved* og *Closed*. *New* er nye oppgaver som må gjøres, *Active* er oppgaver som blir jobbet med eller som inneholder en *bug* og *Resolved* er kode som må kvalitetssikres og som er klar for testing. Koden som ligger under *Closed* er klar til å publiseres [16].

Hver arbeidsoppgave er representert med et kort der det er beskrevet kort hva oppgaven går ut på. Et kort inneholder en *issue* hvor man ser hvem som har fått tildelt denne oppgaven. For å organisere kortene, brukte vi stikkord for å indikere om oppgaven hørte til frontend eller

backend. Kortene ble brukt for å kategorisere og visualisere status på oppgavene. Når et av medlemmene tok på seg en oppgave, lagde man et nytt kort, skrev en kort beskrivelse og førte navnet sitt på kortet. Etter hvert som man jobbet med oppgaven kunne man flytte kortet til en ny kolonne.

Figur 2.1 viser noe av kanbanbrettet for sprint 6. Hvert kort fikk automatisk en ID, etterfulgt av et stikkord og en kort beskrivelse av oppgaven. Man kan også se hvilke gruppemedlemmer som hadde blitt tildelt de ulike oppgavene.



Figur 2.1: Skjermbilde av kanbanbrettet for sprint 6.

2.2 Møter

I løpet av prosjektet, har gruppen hatt tre typer møter. Disse møtene var møte mellom gruppen internt, møte sammen med veileder og møte sammen med oppdragsgiver. Alle fikk vite status på prosjektet og hvordan neste steg i utviklingen ville se ut. De forskjellige møtene er beskrevet under. Se vedlegg B for eksempler på møtereferat.

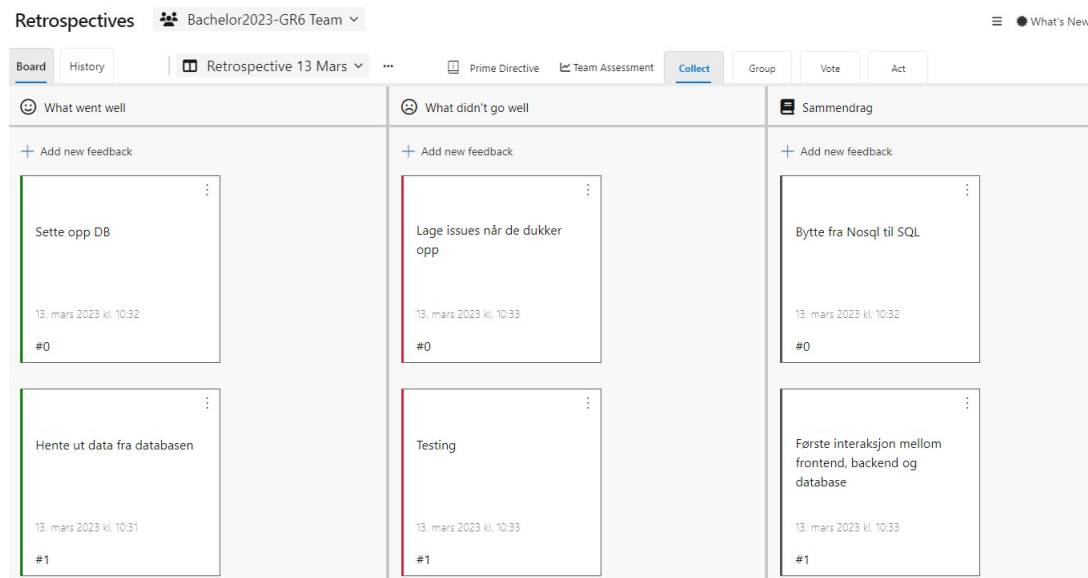
2.2.1 Interne møter

Daily scrum

Hver morgen, fra mandag til fredag kl. 09:00, ble det avholdt et morgenmøte fysisk på campus for å høre om status på ulike deler av prosjektet. Her ble det også lagt frem hva enkeltpersoner hadde gjort dagen før og hva de tenkte å gjøre den kommende dagen. Møtetiden var maksimalt 15 minutter og det var Scrumleder ansvar å avslutte møtet når tiden var nådd. Disse møtene gjorde det enklere å se helheten i prosjektet, samtidig som man får en oversikt over hva de andre gruppemedlemmene jobbet med. Det var også enkelt å spørre om det var noe man lurte på, eller å be om en ekstra hånd på en vanskelig oppgave.

Sprint retrospective

Etter hver sprint ble det holdt *sprint retrospective*-møte som ble ledet av Scrumleder. Siden Azure Boards tilbyr en integrert komponent med navn *Retrospective*, kunne vi bruke denne for å skrive evaluering for hver sprint. Her gikk vi gjennom den foregående sprinten for å finne ut hva som gikk bra og hva som gikk dårlig, før vi lagde et sammendrag av sprinten. Under de tre kolonnene «Hva gikk bra?», «Hva gikk dårlig?» og «Sammendrag» kunne vi sortere de forskjellige komponentene, som vist i figur 2.2.



Figur 2.2: Skjermbilde tatt fra Azure Boards av *Retrospective* for sprint 4.

2.2.2 Veiledningmøter

Hver torsdag kl. 13:00 var det møte med gruppens veileder Rune Hjelsvold. Møtene ble avholdt fysisk på campus, der gruppen informerte om progresjonen og diskuterte spørsmål som hadde kommet opp den siste uken.

2.2.3 Møter med oppdragsgiver

Sprint review og sprint-planlegging

På slutten av hver sprint ble det holdt et møte med veilederene fra Evidi for review av gammel sprint og planlegging av ny sprint. Dette møtet havnet på annenhver mandag kl. 10:00 og ble holdt over Teams. Møtene ble ledet av Scrumleder. Her gikk vi gjennom det som hadde blitt gjort i den gjeldende sprinten, samt hvilke ting som kunne forbedres og legges til i backloggen. Gruppen holdt demoer for å kunne diskutere nåværende og kommende funksjonaliteter. Disse møtene var svært nyttige, siden mange av dem førte til at vi kom frem til ny eller forbedret funksjonalitet. Fra møtet 13. februar diskuterte vi ulike design og funksjonaliteter på booking-siden. Vi kom frem til at det ikke var nødvendig for brukerne å selv bestemme tidspunkt, men at man får tre valg; «Før lunsj», «Etter lunsj» og «Hele dagen». Se vedlegg B for møtereferat. På slutten av møtene planla vi den kommende sprinten og hvilke hovedfunksjonaliteter som skulle implementeres og fordelte arbeidsoppgaver. Siden vi også hadde muligheter til å legge til *issues* i løpet av sprinten, brukte vi ikke lang tid på planleggingen sammen med oppdragsgiver. I tillegg til de «vanlige» møtene, ble det også holdt lynkurs i bruk av Azure pipelines og et kurs forbeholdt brukertesting.

2.3 Git

Git er et versjonskontrollsystem som er brukt for å spore endringer i mapper eller filer [17] [18]. I dette tilfellet er Git brukt for å oppdatere og lagre endringer i koden til prosjektet. Git inkluderer mange nyttige funksjonaliteter, som å sjekke ut eller å gå tilbake til tidligere *commits*. Under er det nevnt andre Git-funksjonaliteter som gruppen brukte under utviklingen av programvaren.

2.3.1 Git commits

For at det skal være enkelt å se tilbake på tidligere endringer i prosjektet, må *commit*-meldingene være enkle og tydelige. Det er også viktig at en *commit* skal gjelde en funksjonalitet. Det vil si at en *commit* ikke burde være avhengig av andre *commits* for å gi mening. Disse måtene å skrive *commit*-meldinger på er hentet fra «Git Best Practices» og er brukt i prosjektet [18].

Git-regler

I prosjektet valgte vi å innføre regler for Git *commits* [19]. Reglene er å bruke stikkord for å identifisere hvilke typer endringer som er gjort i koden. Ved å standardisere disse, ble det lettere å forstå hva som hadde blitt endret for de andre gruppe-medlemmene. I figur 2.3 vises stikkordene brukt i Git *commit*ene.

Git rules

 sigridwv 26 January

<https://github.com/angular/angular/blob/22b96b9/CONTRIBUTING.md#-commit-message-guidelines> 

- All commit messages in lowercase
- Merge develop branch into your feature branch before creating PR to merge into develop

```
build: changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)
ci: changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)
docs: documentation only changes
feat: a new feature
fix: a bug fix
perf: a code change that improves performance
refactor: a code change that neither fixes a bug nor adds a feature
style: changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)
test: adding missing tests or correcting existing tests
```

commit message example:

```
"
feat: add function to receive bookings from backend

fixes #10204
"
```

```
git commit -m "feat: add function to receive bookings from backend" -m "fixes #10204"
```

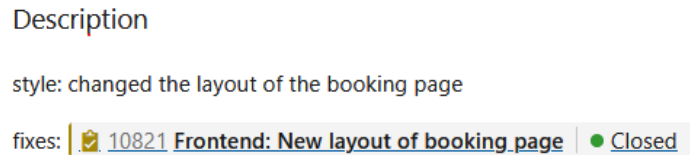
Figur 2.3: Regler for Git *commits*.

Smart-commits

Det ble brukt *smart-commits* for at medlemmene ikke trengte å manuelt oppdatere statusen for tilhørende *issue*.

For at dette skal skje automatisk, må det skrives en *commit*-melding som inneholder ordet *fixes* etterfulgt av ID-en til en *issue*. Et eksempel på en slik *commit* vises i figur 2.3 under *commit message example*.

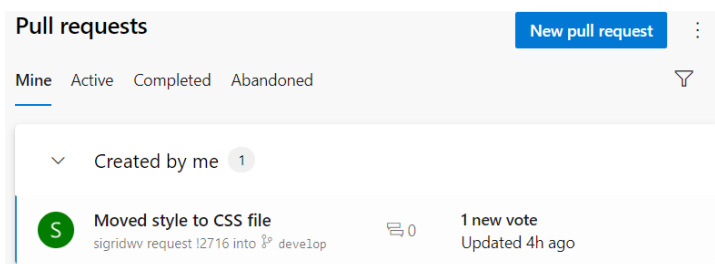
Når en *issue* har blitt linket til en *commit*, vil den dukke opp i tilhørende *pull request* som vist i figur 2.4. Når *pull requesten* blir godkjent og senere *merget* inn i *main*, blir tilhørende *issue* automatisk flyttet til *Closed* i kanbanbrettet ved fullført *merge*. I tillegg har vi valgt å skrive alle *commit*-meldingene på engelsk slik at alle kan forstå meldingene for eventuell videreutvikling av programvaren.



Figur 2.4: *Pull request*: koblet *commits*.

2.3.2 Pull request

Når et av gruppe medlemmene var ferdig med å utvikle koden i en sidegren, ble det lagd en *pull request* der de andre medlemmene måtte se over koden for å godkjenne. Her beskrev man kort hva forespørselen gikk ut på, som figur 2.5 viser. For de andre gruppe medlemmene var det mulig å se *commit*-meldinger og hvilke filer som var endret. Når et av medlemmene ga tilbakemeldinger, var det viktig å være konstruktiv og gi klare tips for forbedring[20].



Figur 2.5: En aktiv *pull request*.

2.4 Gjennomføring av sprinter

Før vi startet på en sprint, gikk vi gjennom hvilke kort som måtte prioriteres, der hvert medlem kunne fortelle hva de selv mente. For å kunne planlegge lengre frem i tid lagde vi mange *user stories* og oppgaver som var en større del av prosjektet.

2.4.1 Sammendrag av sprinter

Prosjektplanlegging (12.januar - 30. januar)

I begynnelsen av prosjektet lagde gruppen en prosjektplan. Hele januar måned gikk med for å lage denne der vi brukte mye tid på både planlegging av prosjektet og rapporten.

Sprint 1 (18.januar - 29.januar)

I den første sprinten brukte vi mye tid på å lese oss opp på ulike teknologier samtidig som vi skulle bli trygge på bruken av Microsoft Azure. Vi hadde fokus på å få en oversikt over hvilke teknologier vi kunne velge mellom og leste om blant annet C#, .NET, React og infrastruktur. Selv om denne sprinten gikk samtidig som prosjektplanleggingen, førte den til at vi kom i gang med prosjektet og ble kjent med Azure.

Sprint 2 (30.januar - 12.februar)

Første interaksjon mellom backend og frontend ble implementert. Vi fikk også satt opp standarder for hvordan vi ville ha *commit*-kommandoene våre. Oppsett av pipelines for infrastruktur ble også gjort for å få serverne opp å gå. Første brukergrensesnittet ble også implementert med

riktig fargemal fra Evidi sine nettsider, slik at det kunne vises til oppdragsgiver for tilbakemelding. I denne sprinten innså vi at vi hadde for store oppgaver og ble dermed enige om at vi skulle dele opp i mindre deloppgaver.

Sprint 3 (13.februar - 26.februar)

Nå startet utviklingen for fullt. Mye av funksjonaliteten på booking-siden ble ferdigstilt. Etter diskusjon med Evidi, endret vi en del på designet med for eksempel å ha bestemte tidspunkter man kan booke i stedet for å velge frie tidspunkt selv. Vi brukte mye tid på frontend, siden vi ikke har drevet så mye med det fra før. I backend fikk vi laget funksjoner for GET, POST og DELETE. I programmet er GET brukt for å hente ut reserverasjoner, POST er brukt for å legge til reserverasjoner og DELETE er brukt for å slette reserverasjoner.

Sprint 4 (27.februar - 12.mars)

I denne sprinten satte vi opp en Azure SQL database for prosjektet. Vi brukte også tid på å hente ut data fra databasen. I tillegg fikk vi implementert den første interaksjonen mellom frontend, backend og databasen, der vi både kunne hente ut og slette data fra databasen. Mye av tiden gikk med på å fremstille data fra backend på en leselig måte i frontend. Å implementere innloggingsfunksjonaliteten og å hente ut brukeren sin tilgangstoken, var en større oppgave enn vi trodde og tok mye tid.

Sprint 5 (13.mars - 26.mars)

Her fikk vi implementert siste funksjonalitet på bookingsiden - grafisk visualisering av andre bookinger. Innlogging med Evidi sine Microsoft-kontoer ble ferdigstilt. Her klarte vi å hente ut tilgangstoken og bruker-ID fra den innloggete brukeren. Etter tilbakemeldinger fra Rune, som kom med tanker der brukeren bør kunne se når en booking ikke blir gjennomført og eventuelle andre feilmeldinger fra backend, jobbet vi en del med å hente ut eventuelle feilmeldinger og vise de på en enkel måte for brukeren. Det gikk også mer tid på å refaktorere kode i både backend og frontend. Vi fjernet kode som ikke var brukt og navnene til filene ble forandret for å støtte en statisk struktur. I frontend og backend ble koden flyttet til egne mapper for å få en bedre oversikt over dataen i filene og for å gjøre det enklere å gjenbruke kode.

Sprint 6 (27.mars - 9.april)

I denne sprinten jobbet vi med å presentere møter, navn og e-post på hjemmesiden. Disse dataene ble hentet fra Microsoft Graph og ble presentert ved hjelp av Microsoft Graph Toolkit. Dette var en større oppgave enn forventet og det gikk mye tid på feilsøking. Noe av problemet var at vi ikke hadde tilgang til å hente ut data på grunn av feil tilgangstoken. I tillegg jobbet vi med å finne en god måte å presentere bursdager på. Vi hadde også problemer med å referere til Key Vault via React. Det ble konkludert ved å bruke Next.js sine serverkomponenter for å løse Key Vault-problemet. Dette kan leses om i seksjon 6.7. All funksjonalitet på nettsiden er snart ferdigstilt, hvor det gjenstår oppdateringer på koden for å optimalisere den.

Sprint 7 (10.april - 23.april)

Pipelines for både frontend og backend ble satt opp i denne sprinten. Mye av tiden har gått med for å ta i bruk Next.js. Dette var mer omfattende enn først antatt og det ble brukt en del tid på å sette seg inn i funksjonaliteten vi ønsket å ta i bruk. Vi har kommet godt i gang med rapporten og funnet en god flyt på skrivingen. Vi har også måttet fikse litt småfeil i programvaren som for eksempel vise riktig dag på bookinger og bursdagsrekkefølge.

Sprint 8 (24.april - 7.mai)

Hovedfokuset vårt i denne sprinten har vært på rapporten. Vi har fått mange tilbakemeldinger fra Rune og brukt mye tid på å få med de viktigste aspektene inn i rapporten. Ved siden av dette, har vi laget skjema for brukertesting og fått utført dette med veiledere fra Evidi. Vi har også fikset feilen som oppstod med tilgangstoken fra sprint 6.

Sprint 9 (8.mai - 22.mai)

I denne sprinten har vi jobbet med rapporten. Vi har fått ferdigstilt mange deler og fått tilbakemeldinger fra andre som har lest over den. Vi har også koblet inn vedlegg og klargjort rapporten til levering. Mot slutten av sprinten gikk tiden til å lese over rapporten. Rapporten ble levert.

Kapittel 3

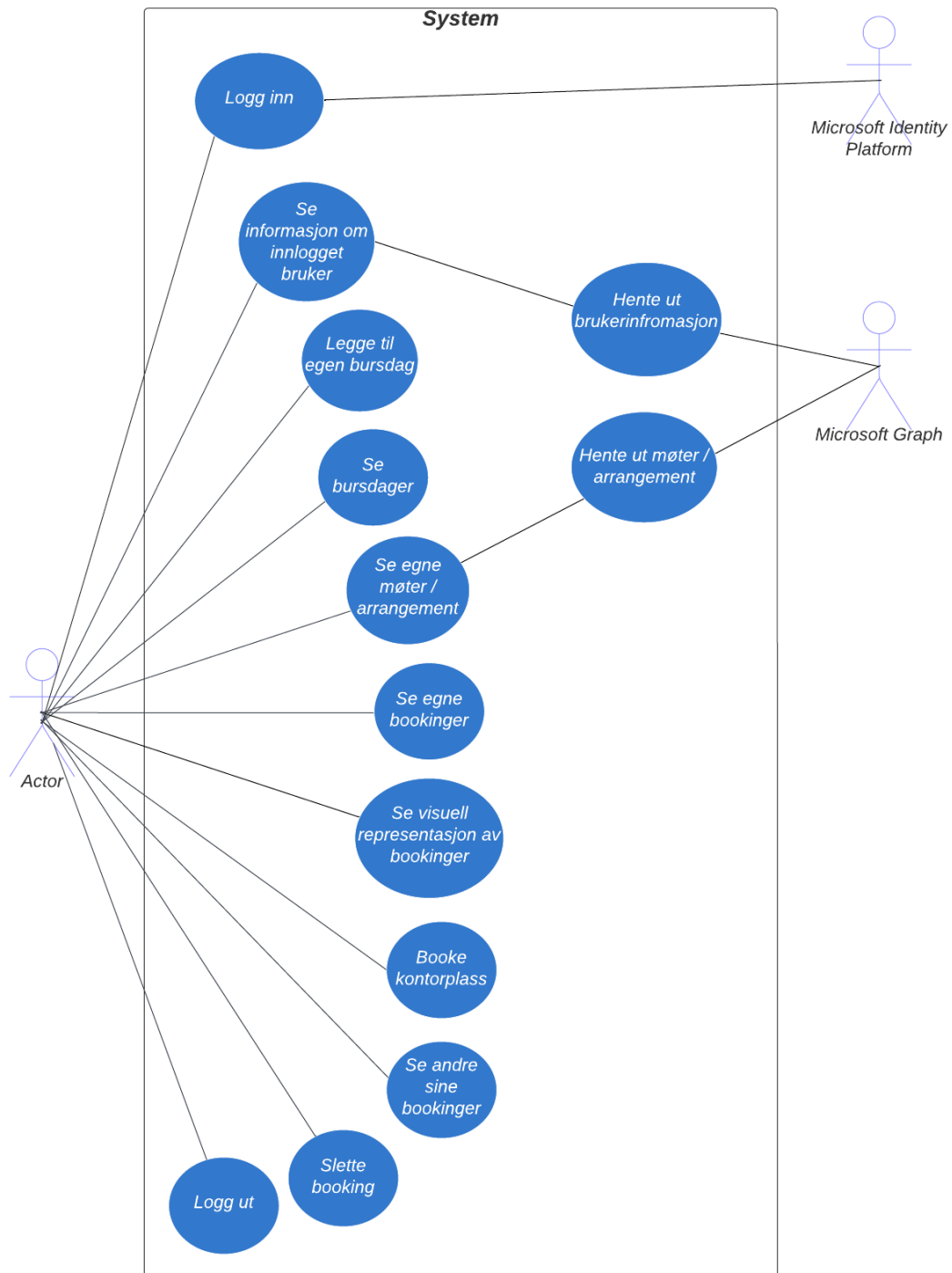
Kravspesifikasjon

Fra oppgavebeskrivelsen, mål og rammer definert i kapittel 1, har vi laget krav til løsningen vår. Disse kravene blir presentert i dette kapittelet og er delt inn i funksjonelle og ikke-funksjonelle krav. Funksjonelle krav inkluderer en beskrivelse av funksjonalitetene til programmet. Ikke-funksjonelle krav er krav som vi anså som viktige å inkludere i utviklingen av programmet. Operasjonelle krav er krav Evidi ikke hadde spesifisert, men som vi ønsket å inkludere. Dette er krav som beskriver hva systemet må være i stand til å gjøre for å oppfylle sine formål.

3.1 Funksjonelle krav

3.1.1 Brukermønster

Fra oppgavebeskrivelsen 1.2 har vi laget et sett med brukermønster som er blitt presentert for oppdragsgiver. Se figur 3.1 for brukermønster-diagrammet. Diagrammet inneholder aktører som er innloggete brukere og eksterne API-er. Den inkluderer også funksjonaliteten som er tilgjengelig for brukeren. Tabellene under seksjon 3.1.1 presenterer høynivå av brukermønster. Brukermønstrene i tabellene ble brukt som utgangspunkt for utviklingen når ny funksjonalitet ble bestemt.



Figur 3.1: Diagram for brukermønster (laget i lucid.app).

Tabell 3.1: Brukermønster 1: Logg inn.

Use case navn	Logg inn.
Aktører	Ansatte hos Evidi.
Mål	Logge inn med eksisterende Microsoft-konto gjennom Evidi.
Beskrivelse	De ansatte hos Evidi skal kunne logge inn på applikasjonen med eksisterende Microsoft-konto som de har fra før.

Tabell 3.2: Brukermønster 2: Informasjon om innlogget bruker.

Use case navn	Se informasjon om innlogget bruker.
Aktører	Ansatte hos Evidi.
Mål	Bruker skal se eget navn og e-post.
Beskrivelse	Brukerinformasjonen blir brukt for å vise at brukeren er innlogget.

Tabell 3.3: Brukermønster 3: Legge til egen bursdag.

Use case navn	Legge til egen bursdag.
Aktører	Ansatte hos Evidi.
Mål	Legge til bursdag.
Beskrivelse	Første gang brukeren logger inn legger man inn bursdagen sin.

Tabell 3.4: Brukermønster 4: Se bursdager.

Use case navn	Se bursdager.
Aktører	Ansatte hos Evidi.
Mål	Se dagens bursdag i tillegg til bursdager i nåværende måned.
Beskrivelse	Se oversikt over dagens bursdager, samt kommende bursdager i nåværende måned.

Tabell 3.5: Brukermønster 5: Se egne møter/arrangement.

Use case navn	Se egne møter / arrangement.
Aktører	Ansatte hos Evidi.
Mål	Se dagens møter eller arrangement.
Beskrivelse	Se en oversikt over møter og arrangement for i dag og i morgen.

Tabell 3.6: Brukermønster 6: Se egne bookinger.

Use case navn	Se egne bookinger.
Aktører	Ansatte hos Evidi.
Mål	Se en oversikt over egne bookinger.
Beskrivelse	Ansatte skal få en oversikt over egne bookinger.

Tabell 3.7: Brukermønster 7: Visuell representasjon av bookinger.

Use case navn	Se visuell representasjon av bookinger.
Aktører	Ansatte hos Evidi.
Mål	Se hvilke kontorplasser som er ledige og opptatte.
Beskrivelse	Hver enkelt ansatt skal kunne se et diagram over alle opptatte kontorplasser for hver etasje på en bestemt dato.

Tabell 3.8: Brukermønster 8: Booke kontorplass.

Use case navn	Booke kontorplass.
Aktører	Ansatte hos Evidi.
Mål	Booke en kontorplass.
Beskrivelse	Hver enkelt ansatt skal kunne booke en ledig kontorplass på kontoret med et gitt tidsintervall.

Tabell 3.9: Brukermønster 9: Se andre sine bookinger.

Use case navn	Se andre sine bookinger.
Aktører	Ansatte hos Evidi.
Mål	Se en oversikt over andre sine bookinger.
Beskrivelse	Ansatte skal få en oversikt over andre sine reserverasjoner for en bestemt dato og etasje.

Tabell 3.10: Brukermønster 10: Slette bookinger.

Use case navn	Slette egne bookinger.
Aktører	Ansatte hos Evidi.
Mål	Å kunne slette en booking.
Beskrivelse	De ansatte skal ha muligheter til å slette sine egne bookinger.

Tabell 3.11: Brukermønster 11: Logg ut.

Use case navn	Logg ut.
Aktører	Ansatte hos Evidi.
Mål	Logge ut av nettsiden.
Beskrivelse	Gjør det mulig å logge ut av applikasjonen.

3.2 Ikke-funksjonelle krav

Under er en liste med krav som inngår i kvaliteten til systemet og designet av programvaren.

- Tjenesten skal kjøres i Microsoft Azure.
- Webgrensesnittet skal fungere på nettleserne Firefox 109.0 og Google Chrome 109.0.5414.76, samt nyere versjoner av disse.
- Mulighet for enkel videreutvikling av programmet i backend og frontend.
- Løsningen skal ha et enkelt og intuitivt brukergrensesnitt. Det vil si at de ulike funksjonalitetene på applikasjonen skal være entydige og forklarende.

3.3 Operasjonelle krav

Evidi hadde ingen spesifikke krav på opptiden og responstiden til programmet. De kravene som er nevnt er krav gruppemedlemmene ønsket å inkludere i programmet. Grunnen til dette er at det er viktig å fokusere på å optimalisere kjøringen av programmet, samtidig som man skal implementere alle nødvendige funksjonaliteter.

Det er ikke mange brukere som kommer til å bruke nettsiden samtidig. Dermed er opptiden satt til å være aktiv når programmet er i bruk. Responsen fra programmet skal være umiddelbar, slik at brukeren kan utføre sine oppgaver uten problemer.

Under er en liste med mer konkrete mål for de operasjonelle kravene:

- Programmet skal håndtere minst 30 brukere samtidig.
- Responstiden skal være mindre enn ett sekund.

Kapittel 4

Teknologier

4.1 Microsoft Azure

Microsoft Azure er en skytjeneste laget av Microsoft der man kan bygge, administrere og distribuere applikasjonene hvor som helst [21]. Skyen leverer tjenester for servere, nettverk, lagring og applikasjoner. Azure tilbyr tre tjenestemodeller: Infrastruktur som tjeneste, Plattform som tjeneste og Programvare som tjeneste [22]. I løsningen vår har vi benyttet oss av Plattform som tjeneste, siden applikasjonen og dataen er vårt ansvar mens skyen tar ansvar for blant annet servere, nettverk og kjøretid. Fordelen ved å bruke en skytjeneste, er at det er enkelt å skalere serverne. I tillegg slipper man også unna drift og vedlikehold av serverne.

4.1.1 Azure Static Web App

Azure Static Web App er en tjeneste som automatisk bygger og publiserer koden vi gir den[3]. Dette brukes for å automatisk installere nødvendige pakker, samt slippe manuelt oppsett av infrastrukturen. Det gir også en enkel løsning for å integrere kontinuerlig bygging. Hvordan vi har tatt i bruk dette forklares nærmere i kapittel 9.

4.1.2 Azure Function

Azure Function er en serverløs løsning som gjør det enkelt for utviklere å lage API-er i backend[2]. Løsningen gjør at man slipper å håndtere infrastrukturen og gjør det mulig å skalere tjenesten etter behov. Selve koden til funksjonene blir kjørt i isolasjon når det blir sendt inn en forespørsel. I vårt tilfelle vil funksjonene kjøre når frontenden sender en forespørsel til et av endepunktene i backend.

4.2 Frontend

4.2.1 React

HTML, CSS og JavaScript er hovedteknologien som blir brukt for å lage webapplikasjoner. HTML og CSS blir brukt for den overflatiske presentasjonen, mens JavaScript trengs for funksjonalitet og interaksjon.

Det finnes mange JavaScript-rammeverk som gjør det enklere å bygge interaktive webapplikasjoner. React er et av de mest kjente rammeverkene med flere millioner utviklere [23]. React er et JavaScript-bibliotek som brukes for å lage og bygge et brukergrensesnitt av individuelle deler [24]. Disse delene kalles komponenter og er JavaScript-funksjoner.

Når vi valgte rammeverk for frontend-applikasjonen, var det viktig at den skulle være rask å ta i bruk samtidig som den skulle ha fokus på brukervennlighet. I tillegg var det viktig at den

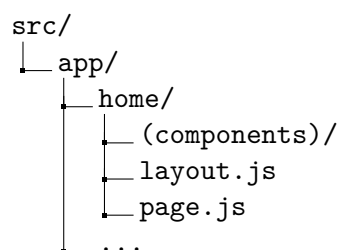
var godt dokumentert, siden ingen av gruppemedlemmene har jobbet med frontend-utvikling tidligere.

4.2.2 Next.js

Next.js er et React-rammeverk for utvikling av webapplikasjoner[25]. Next.js gir ekstra struktur, funksjonalitet og optimalisering til React. De største fordelene er separasjon av klient- og server-side-rendering, god støtte for *layouts* og navigering, samt serverkomponenter.

Fordelen med serverside-rendering er at nettsiden blir klargjort på server-siden før den blir sendt til klienten. Dette gjør det mulig å hente miljøvariabler under kjøretiden slik at disse ikke blir eksponert på klientsiden. Ved klientside-rendering blir alt dette gjort hos klienten og det vil ikke være mulig å bruke miljøvariabler på samme måte. En annen fordel med server-side-rendering er at nettsiden krever mindre ressurser fra klienten som vil minske innlastningstiden.

For å definere de forskjellige endepunktene for nettsiden, blir mappestrukturen og filen *page.js* brukt. Hver mappe under app-mappen som inneholder en *page.js*-fil blir ett eget endepunkt. Utenom det er *page.js*-filen også brukt for å definere all kode som tilhører *home*-siden. Dette ser vi i figur 4.1 hvor *page.js*-filen ligger under *home/*. En annen ting å legge merke til er bruk av parenteser rundt mapper som (*components*). Dette er Next.js sin måte å bruke mapper på uten å opprette nye endepunkt.



Figur 4.1: Struktur for å opprette home endepunkt.

En annen positiv side er bruk av *layouts*. *Layouts* er komponenter som pakker alle sidene og muliggjør å definere ønsket struktur. Dette gjør det mulig å definere komponenter man ønsker skal være på alle sidene, som for eksempel en navigasjonsbar eller bunntekst. I figur 4.1 ser vi at *home*-siden bruker en slik *layout.js*-fil. I prosjektet har vi brukt denne filen til å sjekke autorisering siden brukeren må være autorisert til en hver tid for å bruke applikasjonen.

4.2.3 Microsoft Graph

Microsoft Graph er en programmerbarhetsmodell som kan brukes for å få tilgang til data i Microsoft 365 sine tjenester [26]. Tjenesten har tre hovedkomponenter for å aksessere data.

Microsoft Graph API

Tilbyr et enkelt endepunkt; <https://graph.microsoft.com> som gir innsikt i Microsoft sin sky, inkludert Microsoft 365 og Windows. Dette brukes for å bygge applikasjoner som støtter Microsoft 365. I tillegg inkluderer den et sett av tjenester som administrerer bruker- og enhetsidentitet, tilgang, sikkerhet og bidrar til å beskytte organisasjoner mot datalekkasje eller tap.

Microsoft Graph connectors

Leverer data utenfor Microsoft-skyen til Microsoft Graph-tjenester og -applikasjoner for å forbedre Microsoft 365-opplevelser.

Microsoft Graph Data Connect

Utgjør et sett med verktøy for å forme en sikker og skalerbar levering av data fra Microsoft Graph. Dataene blir ofte brukt til Microsoft Azure sine utviklingsverktøy for å bygge intelligente applikasjoner.

Til sammen driver disse tre komponentene plattformen Microsoft 365. Ved å få tilgang til data fra Microsoft Graph, kan man utvide opplevelser i Microsoft 365 og bygge unike og intelligente applikasjoner. Noen av kjernetjenestene til Microsoft 365 er OneDrive, OneNote, Outlook kalender og Teams.

4.2.4 Microsoft Graph Toolkit

Microsoft Graph Toolkit er en samling av gjenbrukbare, rammeverkskomponenter for arbeid med Microsoft Graph [27]. Komponentene er fullt funksjonelle som både autentiserer med og henter data fra Microsoft Graph. Dette verktøysettet gjør det enkelt å bruke Microsoft Graph i applikasjonen. Microsoft Graph Toolkit inneholder en samling av web-komponenter for de mest brukte funksjonene fra Microsoft Graph sitt API. Verktøysettet lar deg raskt og enkelt integrere data fra Microsoft Graph inn i applikasjonen din. Fordeler med å bruke dette verktøysettet:

- Kutte ned på utviklingstiden
 - Ved å bruke eksisterende verktøy, sparer man mye tid på utviklingen.
- Fungerer overalt
 - Komponentene i settet er basert på nettstandarder og fungerer med alle moderne nettlesere og nettrammeverk som for eksempel React.
- Designet er enkelt og fleksibelt
 - Komponentene er designet for å være brukervennlige, men de kan også tilpasses ved å brukes egendefinerte CSS-egenskaper.

I løsningen vår, er det brukt to komponenter fra Microsoft Graph Toolkit; *Person* og *Agenda*.

Person

Denne komponenten blir brukt for å presentere en person med navn og e-post [28]. Fra endepunktet <https://graph.microsoft.com/v1.0/me>, kan vi hente informasjon om brukeren. For å sette persondetaljene, kan man bruke *person-details*-attributten for å manuelt sette detaljene, som vist i kodelisting 1. Det er mulig å endre komponenten i en egen CSS-fil. Tillatelse som trengs for å bruke denne komponenten, er *User.Read*.

Listing 1: Person-komponent fra Microsoft Toolkit.

```
1 import {Person, PersonViewType} from '@microsoft/mgt-react';
2
3 <Person
4   personDetails={
5     displayName: "Ola Nordmann",
6     mail: "ola.nordmann@gmail.com"
7   }
8 />
```

Agenda

Agenda-komponenten presenterer ulike eventer fra brukeren sin kalender [29]. Som standard presenterer den kalenderen til den innloggede brukeren. Komponentene bruker endepunktet `https://graph.microsoft.com/v1.0/me/calendarview` for å hente eventer fra kalender til brukeren. Videre må man definere en mal `eventTemplate` som brukes for å visualisere dataen fra `events`, som vist i kodelisting 2. Tillatelse som trengs for å bruke komponenten, er `Calendars.Read`.

Listing 2: Agenda-komponent fra Microsoft Toolkit.

```

1 import {Agenda} from "@microsoft/mgt-react";
2
3 <Agenda
4   events={dataToDisplay}
5   eventTemplate={selfMadeTemplate}
6 />
```

4.2.5 MUI

Material-UI (MUI) er et React-komponentbibliotek med åpen kildekode som implementerer Googles Material Design [30]. Biblioteket inkluderer en samling av forhåndsbygde komponenter som er klare til bruk. De har et enkelt design og en rekke tilpasningsalternativ som gjør det enkelt å implementere dem til et eget system.

Datepicker

`DatePicker` er en komponent i MUI-biblioteket som gjør det mulig for brukeren å velge en dato. Komponentene er bygd opp ved å bruke `DateField`, som gjør det mulig å skrive inn dato fra tastaturet, og `DateCalendar` som gjør det mulig å velge dato fra kalenderen [31]. For å forsikre at `DatePicker`-komponenten bruker de samme lokaliseringssinnstillingene som resten av programmet, brukes `LocalizationProvider` sammen med `DatePicker`-komponenten. Se kodelisting 3. Dette fører til at datovelgeren blir lokalisert til webserverens region slik at det blir riktig dato- og klokkeslettformat [32]. `AdapterDayjs` er et bibliotek brukt sammen med `LocalizationProvider` som gjør det mulig å bruke Day.js biblioteket for håndtering av dato- og tidsformattering [33]. `TextField` brukes for å returnere et tekstfelt med input-variablene `params`.

Listing 3: `LocalizationProvider` og `DatePicker`.

```

1 import { DatePicker } from '@mui/x-date-pickers/DatePicker';
2 import { LocalizationProvider } from '@mui/x-date-pickers/LocalizationProvider';
3 import { AdapterDayjs } from '@mui/x-date-pickers/AdapterDayjs';
4 import TextField from '@mui/material/TextField';
5
6 <LocalizationProvider dateAdapter={AdapterDayjs}>
7   <DatePicker
8     renderInput={({params}) => <TextField {...params} />
9     ...
10   />
11 </LocalizationProvider>
```

Dialog

Dialog er en komponent i MUI-biblioteket som er brukt for å vise tilleggsinformasjon i et vindu. Når vinduet blir vist, blir alt annet av funksjonalitet på applikasjonen deaktivert. Avhengig av hvilken informasjon som blir vist, kan brukeren velge mellom en rekke bekreftelsesvalg; lukk, send inn, aksepter osv. Et eksempel på et *Dialog*-vindu, er når brukere må akseptere bruk av informasjonsskapsler på en nettside.

Når vinduet kommer opp, må brukeren godkjenne et valg før de kan gå videre til applikasjonen sin hjemmeside [34]. Kodeliste 4 viser til *DialogContent* og *DialogActions* som ligger inne i *Dialog* komponenten. *DialogContent* inkluderer innholdet som blir vist på *dialogen*. *DialogActions* spesifiserer brukerhandlingene inkludert på *Dialogen*.

Listing 4: *Dialog* sammen med *DialogContent* og *DialogActions*.

```

1 import Dialog from "@mui/material/Dialog";
2 ...
3 <Dialog>
4   <DialogContent >
5     ...
6   </DialogContent>
7   <DialogActions>
8     ...
9   </DialogActions>
10 </Dialog>

```

4.2.6 Chart.js

Chart.js er et gratis JavaScript-bibliotek for å lage HTML-baserte diagrammer [35]. Chart.js ble opprettet i 2013 og har gått en lang vei siden den gang [36]. Biblioteket gir et sett med ulike innebygde diagramtyper og tilpasningsalternativer, noe som gjør det enkelt å få komponentene til å passe til hver enkel applikasjon. Chart.js er i tillegg kompatibel med flere JavaScript-rammeverk, som feks React. Biblioteket er godt dokumentert og det skal være enkelt å ta i bruk. Kodelisting 5 viser hvordan et stolpediagram fra Chart.js blir implementert. I diagrammet defineres datasettet og hvilke alternativer som skal brukes. Linje 5 registrerer tilleggsmodulene som finnes i alternativene til diagrammet.

Listing 5: Stolpediagram fra Chart.js.

```

1 import { Chart, registerables } from "chart.js";
2 import { Chart as ReactChartJs } from "react-chartjs-2";
3 import "chartjs-adaptor-date-fns";
4
5 Chart.register(...registerables);
6
7 <ReactChartJs type="bar"
8   data={data}
9   options={options}
10 />

```

4.2.7 Cookie-cutter

Cookie-cutter er et simpelt JavaScript-bibliotek som gjør det mulig å sette informasjonskapsler for nettsiden [37]. Informasjonskapsler blir brukt for å lagre usensitive data slik at det enkelt kan hentes fra andre komponenter ved behov.

4.3 Backend

4.3.1 Programmeringsspråk

C# er et objekt-orientert programmeringsspråk som består av en blanding av Java, C og C++. C# blir ofte brukt sammen med .NET. Språket gjør at vi kan lage robuste og sikre programmer ved hjelp av *garbage collection* og feilhåndtering. *Garbage collection* håndterer automatisk frigjøring og allokering i minnet. Feilhåndtering hjelper med å fange opp og håndtere feil under kjøring. Språket støtter også asynkroniserte operasjoner som gjør det enklere å håndtere prosessering av data mellom databasen og frontend [6].

.NET er en plattform brukt for å bygge forskjellige typer applikasjoner, som for eksempel API-er. I tillegg er plattformen kompatibel med ulike biblioteker som for eksempel SqlClient-biblioteket. Noen av godene som .NET inkluderer er:

- *Cross platform støtte*; kan kjøres på Windows, MacOS og Linux.
- Den inneholder standardiserte biblioteker og API-er.
- Den er kompatibel med C#.

4.3.2 Timer Trigger

Timer Trigger er en triggertype som blir brukt for å spesifisere tiden for kjøring av en kodeblokk. Tiden blir spesifisert ved hjelp av NCRONTAB-biblioteket. NCRONTAB-uttrykket inkluderer følgende verdier; {second} {minute} {hour} {day} {month} {day-of-week} [38].

Funksjonaliteten er brukt for å slette alle reserverasjoner fra databasen på et gitt tidspunkt for å gi plass til nye reserverasjoner i databasen.

4.4 Database

Azure SQL-databasen er en «Database som en tjeneste» (DBaaS). DBaaS er godt egnet for moderne skyapplikasjoner og bruker de seneste SQL server-funksjonalitetene. For at dataene i programmet skal holde seg konstant, har vi tatt i bruk Azure SQL. Databasen er godt egnet for å hente ut, legge inn og endre dataene som ligger i databasen. Årsaken til valg av denne type database, er at informasjonen som skal bli lagret var noe uavhengig av hverandre og det ville bli mer oversiktlig å ha en struktur på dataene som skulle hentes ut.

Ved å bruke Azure SQL, betaler man bare for det som blir brukt. Det er mulighet for oppskalering av databasen, som betyr at databasen har mulighet for å lagre mer data [39].

4.5 Testing

I applikasjonen bruker vi rammeverket xUnit som er det nyeste innen enhetstesting for .NET. Rammeverket er utviklet av den originale oppfinneren av testrammeverket NUnit [14]. xUnit inneholder mange nyttige verktøy og instruksjoner. Instruksjonene blir delt inn i *Facts* og *Theory*. *Facts* deklarerer bare positive tester. *Theory* deklarerer tester som er sanne for noen spesifiserte testdata [40]. Kodeliste 6 viser hvordan attributtene er brukt ved testing av programmet.

Testene i xUnit er uavhengige av hverandre. Det vil si at når en testmetode blir kjørt, vil ikke denne testen ha noe å si for utkommet av en annen test.

Listing 6: Kodeliste som viser bruk av *Fact* og *Theory* attributtene.

```
1 [Fact]
2 public async Task TestGetMyBookingWithoutQueryValue()
3 {
4     ...
5 }
6
7 [Theory]
8 [InlineData("abc-df-1234-5678")]
9 public async Task TestGetMyBookingWithID(string id)
10 {
11     ...
12 }
```

4.6 Sikkerhet

4.6.1 Azure AD

Azure AD er en skybasert identitets- og tilgangsstyringstjeneste. Tjenesten er brukt for å gi brukere og programmer tilgang til eksterne ressurser som for eksempel Microsoft-365. Den inkluderer en enkel pålogging, flerfaktoraутentisering og betinget tilgang for å beskytte mot angrep [41]. Det er her man setter hvilke type grupper som har tilgang til applikasjonen og hvilke autentiseringsretningslinjer som skal bli brukt.

4.6.2 Microsoft Identity Platform

Microsoft Identity Platform er en plattform som inkluderer tjenester som utviklere kan bruke for å tilpasse innloggings-/autoriseringsfunksjonaliteten [42]. Den gjør det enkelt å autorisere tilgang til eksterne og egne API-er. Plattformen består av ulike komponenter som for eksempel støtte til MSAL-biblioteket og presentasjon av ulike innloggingsalternativer.

4.6.3 MSAL

MSAL er et bibliotek fra Microsoft som blir brukt til å håndtere innlogging av brukere og uthenting av tilgangstoken og bruker-ID. Applikasjonen vår bruker Microsoft Identity Platform sammen med MSAL for å autentisere brukere. For at utviklerne skal slippe å tenke på livsløpet til et tilgangstoken vil MSAL håndtere *token cache*. Det vil si at den henter ut et tilgangstoken som ikke er utløpt og oppdaterer det for oss når utløpstiden nærmer seg [43].

Biblioteket inkluderer også muligheten for å velge hvilken type konto vi vil at brukeren skal logge inn med; Microsoft, Google, osv.

Det er mulighet for å sette opp innloggings- og utloggingsfunksjonaliteten med en konfigurasjonsfil i en applikasjon.

Andre funksjonaliteter som MSAL inkluderer er [44]:

- Single Sign On (SSO) støtte. Denne støtten resulterer i at brukeren ikke trenger å logge inn flere ganger med samme konto hvis de er allerede innlogget.
- MSAL oppdaterer seg med de nyeste sikkerhetsfunksjonene automatisk. Dermed forsikrer utviklerne seg om at funksjonaliteten de tar i bruk er sikker.
- MSAL er kompatibel med de fleste rammeverk, som for eksempel React. Biblioteket «msal-react» er brukt sammen med React for håndtering av innlogging. MSAL inkluderer også «msal-browser» som er brukt for å utføre autentisering i React *single-page*-applikasjoner [45].

4.6.4 Key Vault

For å lagre data som ikke skal være tilgjengelig for brukeren av applikasjonen, bruker vi Key Vault [46]. Key Vault er en skybasert sikkerhetstjeneste fra Microsoft som kan lagre *secrets*, for eksempel passord og tilgangstoken, på en sikker og kryptert måte. Hver *secret* får sin egen Uniform Resource Identifier (URI) som senere kan refereres til for å hente ut ønsket *secret*.

4.6.5 API token

Det kreves et tilgangstoken med riktige rettigheter for å få tilgang til endepunktene fra API-et i backend. Dette tokenet blir sendt via headeren *x-functions-key* ved å hente miljøvariabelen som vist i kodelisting 7. Navnet på denne headeren må være *x-functions-key* da Azure Function bruker denne slik at det hindrer uautorisert bruk av API-et.

Listing 7: Bruk av miljøvariabel.

```
1  const options = {
2    ...
3    headers: {
4      'x-functions-key': process.env.REACT_APP_FUNCTION_HOST_KEY
5    }
6  }
7  export async function myFunction(){
8    const res = await fetch(api-url, options);
9  }
```

4.7 JSON og REST

JSON er et populært format som blir brukt sammen med sending og lagring av data. Årsaken for populariteten til formatet, er at det gjør det enklere å lese dataene. I tillegg trenger formatet mindre kode for å bli opprettet og behandlet, og det blir behandlet fortere av maskiner [47].

REST er et standard programvarearkitekt-design som ofte blir brukt under utvikling av API-er, som gjelder kommunikasjonen mellom klienten og serveren [12].

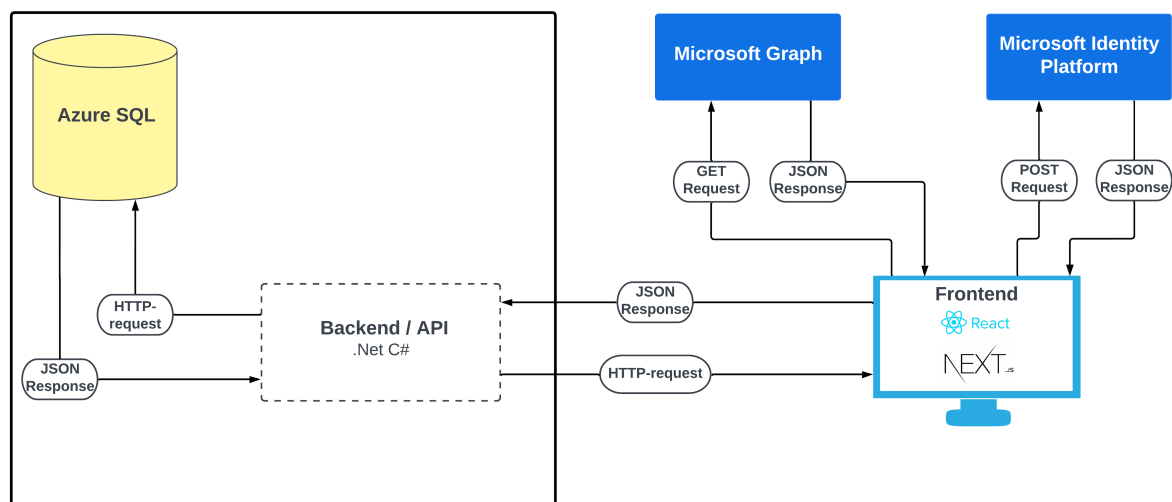
Kapittel 5

Design

5.1 Systemdesign

Løsningen vår er en moderne RESTful API-tjeneste som har en backend-server som kommuniserer med en database. Klienten, i dette tilfellet frontenden, sender en forespørsel til backend. Forespørselen utløser en funksjon som endrer eller henter ut data fra databasen. Denne interaksjonen av data er illustrert i figur 5.1. Modellen skiller mellom eksterne og interne løsninger, hvor de eksterne løsningene er Microsoft Graph og Microsoft Identity Platform.

Vi bestemte oss for å ha en relativ tynn klient med et valg der henting av data fra Microsoft Graph ble gjort direkte fra klienten og ikke via backend-serveren. Grunnen til dette var at vi ikke lagrer noe av dataen fra Microsoft Graph, samt at det var tenkt å hente ut tilgangstokenet fra Microsoft Identity Platform fra frontend. Den endelige applikasjonsarkitekturen til prosjektet er nærmere diskutert i kapittel 10.

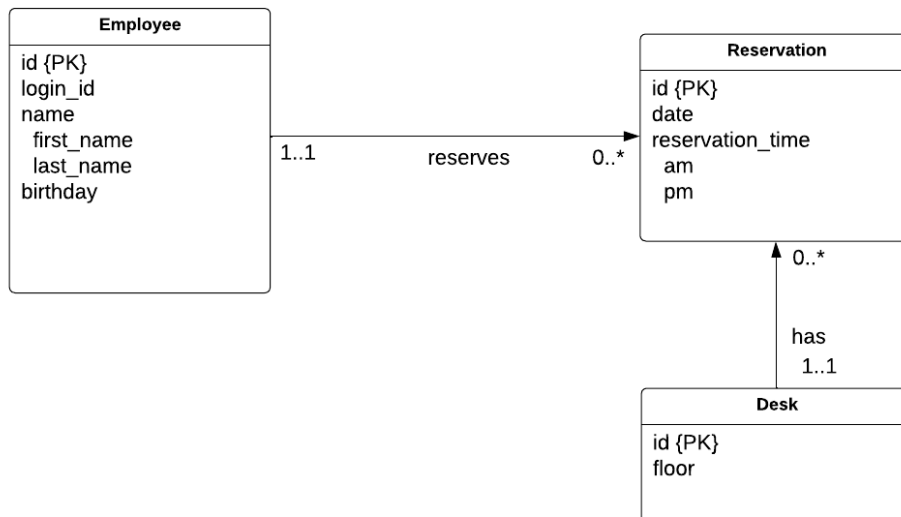


Figur 5.1: Applikasjonens arkitektur (laget i lucid.app).

5.2 Backend

Som nevnt tidligere er backenden en RESTful API-tjeneste. Tjenesten baserer seg på Azure Function som håndterer hovedlogikken til applikasjonen med blant annet opprettelse av reserveringer og henting av bursdager. Fordelen med denne arkitekturen er at mesteparten av dataprosesseringen foregår på en sentralisert server, slik at klienten kun trenger å håndtere hvordan dataen skal presenteres. Azure Function er derfor et godt valg for API-et ettersom den bare kjører nødvendig kode når den blir kalt. Dette gjorde at vi kunne fokusere på funksjonaliteten til programmet i stedet for miljøet rundt. Azure Function tok altså hånd om miljøet, noe som sparte oss for både tid og ressurser.

For lagring og henting av data, kommuniserer backenden med Azure SQL-databasen slik at databasen ikke trenger å bli eksponert til klienten. I den konseptuelle modellen vist i figur 5.2, vises det en oversikt over tabellene som er i databasen og hvilke relasjoner den har. Dataen inkludert i hver tabell er visualisert ved hjelp av attributter i modellen. Den konseptuelle modellen inkluderer tabeller for brukere, reserveringer og pulter. Hver tabell inneholder en primærnøkkel som skal identifisere en spesifikk tabell. Relasjonene mellom tabellene er brukt for å identifisere forholdet mellom tabellene og hjelper med å angi hvilke tabeller som er avhengige av hverandre.



Figur 5.2: Konseptuell modell (laget i lucid.app).

De binære attributtene *am* og *pm* i *Reservation* representerer tidspunktene «Før lunsj» og «Etter lunsj». Hvis *am* er satt til 1 og *pm* er satt til 0 vil reserveringen gjelde før lunsj. For å reservere hele dagen må begge attributtene være satt til 1. Attributtet *login_id* i *Employee* representerer bruker-ID hentet fra tilgangstokenet fra brukeren.

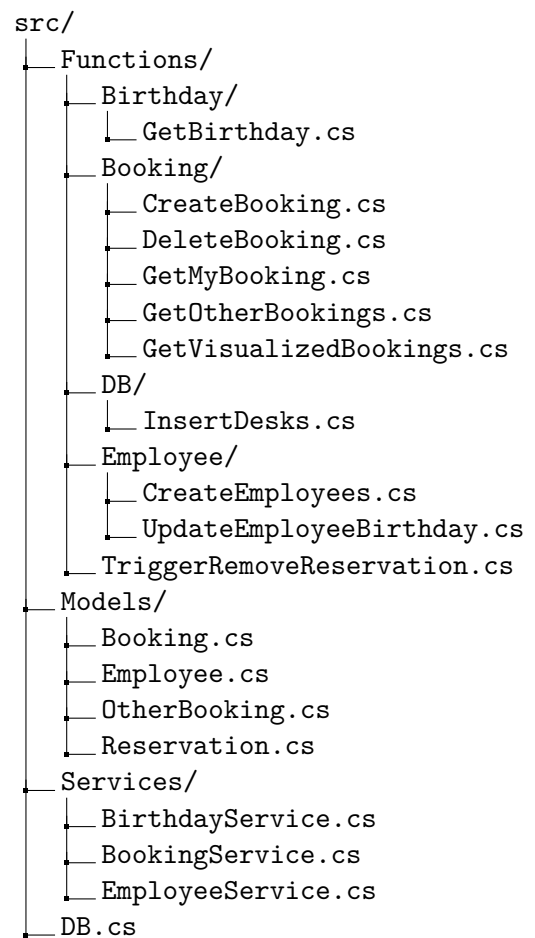
5.2.1 Filstruktur backend

Ettersom tjenesten baserer seg på Azure Function som beskrevet nærmere i 4.1.2, valgte vi å plassere inngangs-funksjonene under en egen *Functions*-mappe. Inngangs-funksjonene er funksjonene Azure Functions utfører når HTTP-kall blir mottatt [48].

Dette medførte at vi enkelt kunne legge til nye endepunkt, samt få en oversiktlig samling av alle inngangs-funksjonene. Vi valgte videre å dele de tilhørende funksjonene for endepunktene under hver sin mappe som vist under *Functions*-mappen i figur 5.3. Her ser vi at funksjoner relatert til booking-API-et er plassert under Booking-mappen, med filer som inneholder koden for endepunktene.

For representering og konstruksjon av data, lagde vi egne modell-filer under *Models*-mappen. Disse filene blir brukt for å opprette et objekt basert på data fra forespørselen, noe vi forklarer videre i 6.1.3.

Ved å separere logikken i egne filer, ble koden til inngangs-funksjonene mer oversiktlig slik at de kun håndterer det som er nødvendig for selve API-kallet som respons og feilmeldinger. Vi valgte å plassere logikken som de forskjellige inngangs-funksjonene utfører under *Services*-mappen hvor hvert endepunkt har sin egen *service*-fil.



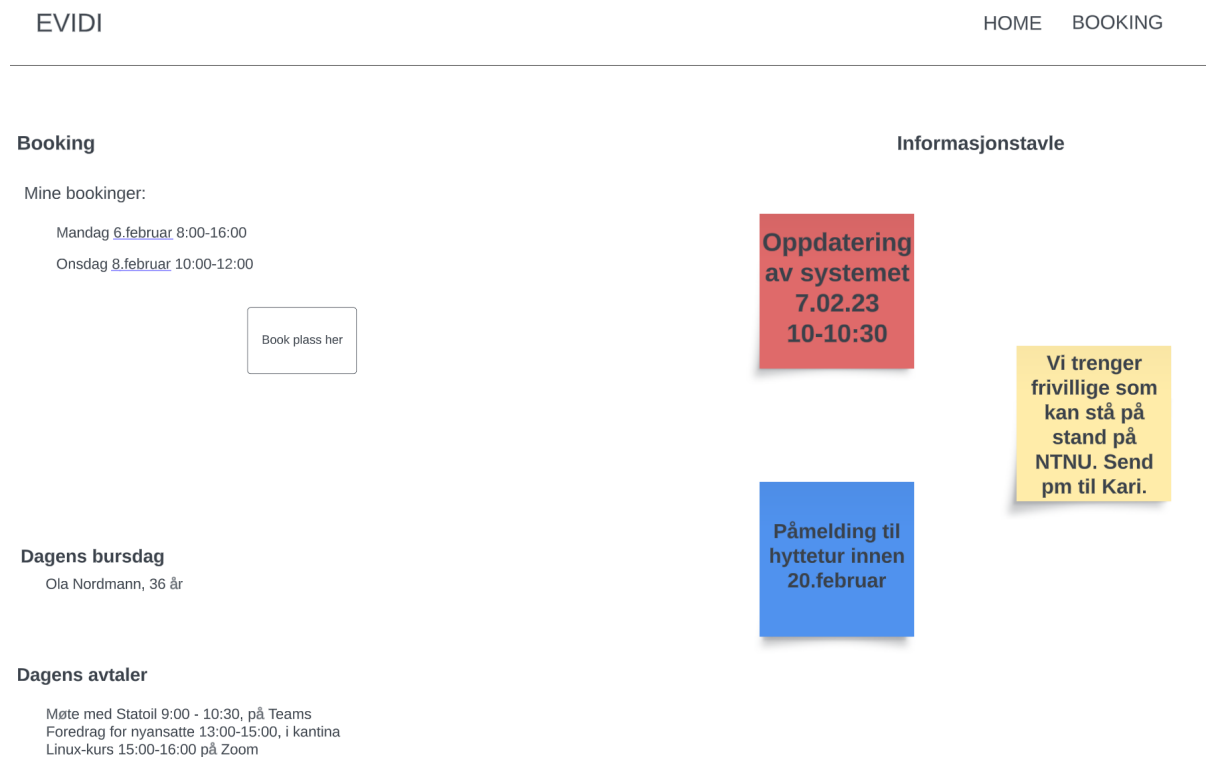
Figur 5.3: Filstruktur backend.

5.3 Brukergrensesnitt

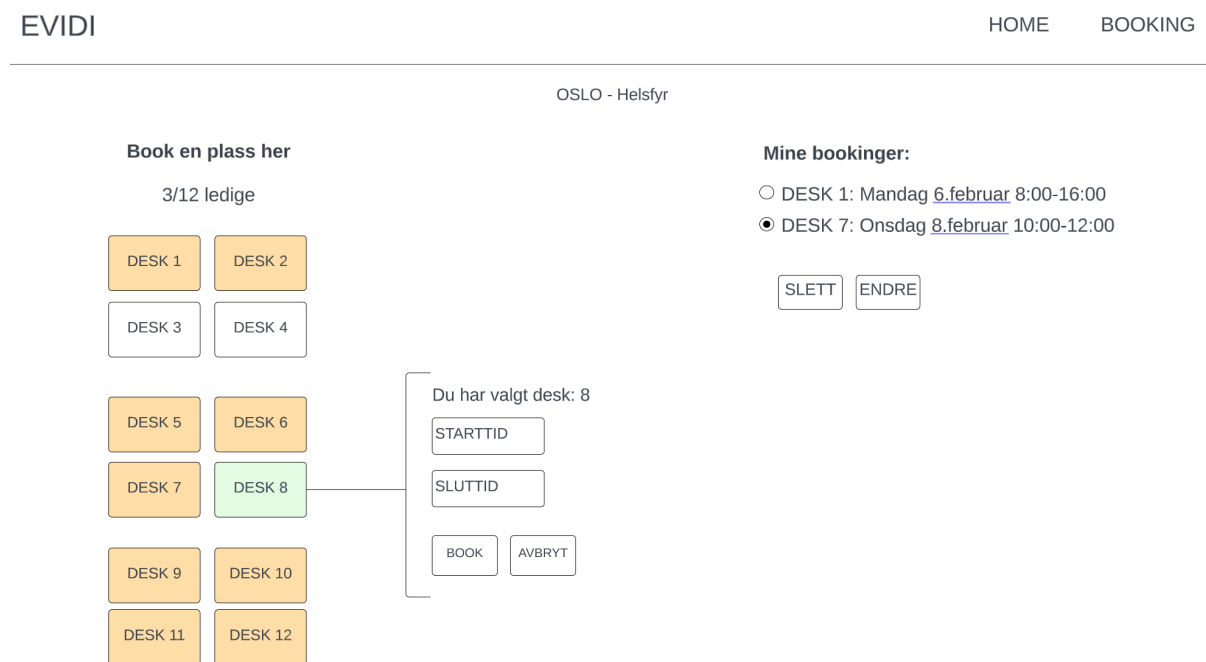
Tidlig i utviklingsfasen ble det vurdert om vi skulle utvikle en webapplikasjon eller en mobilapplikasjon. Ønskene til Evidi var at løsningen skulle brukes som en informasjonstavle der alle ansatte lett kunne logge seg inn for å se. Å utvikle en webapplikasjon som publiseres i Azure, ville være med på å fylle dette kravet. En webapplikasjon vil alltid være tilgjengelig for de ansatte uten at det er nødvendig å laste ned noe. Det er også enkelt å utvikle en webløsning ved hjelp av teknologiene beskrevet i kapittel 4. På grunn av disse fordelene og ønskene til Evidi, valgte vi å utvikle en webapplikasjon.

5.3.1 Skisser

Tidlig i prosjektfasen ble det laget skisser for hvordan gruppen tenkte at brukergrensesnittet kunne se ut. Disse representerer ikke det endelige produktet og er presentert i figurene 5.4 og 5.5. Skissene ble brukt for å gi oppdragsgiver en enkel presentasjon før utviklingen startet. I løpet av utviklingsprosessen ble det gjort flere endringer, men idéene ble hentet ut fra skissene.



Figur 5.4: Skisse av hjemmesiden (laget i lucid.app).



Figur 5.5: Skisse av bookingsiden (laget i lucid.app).

5.3.2 Resultat

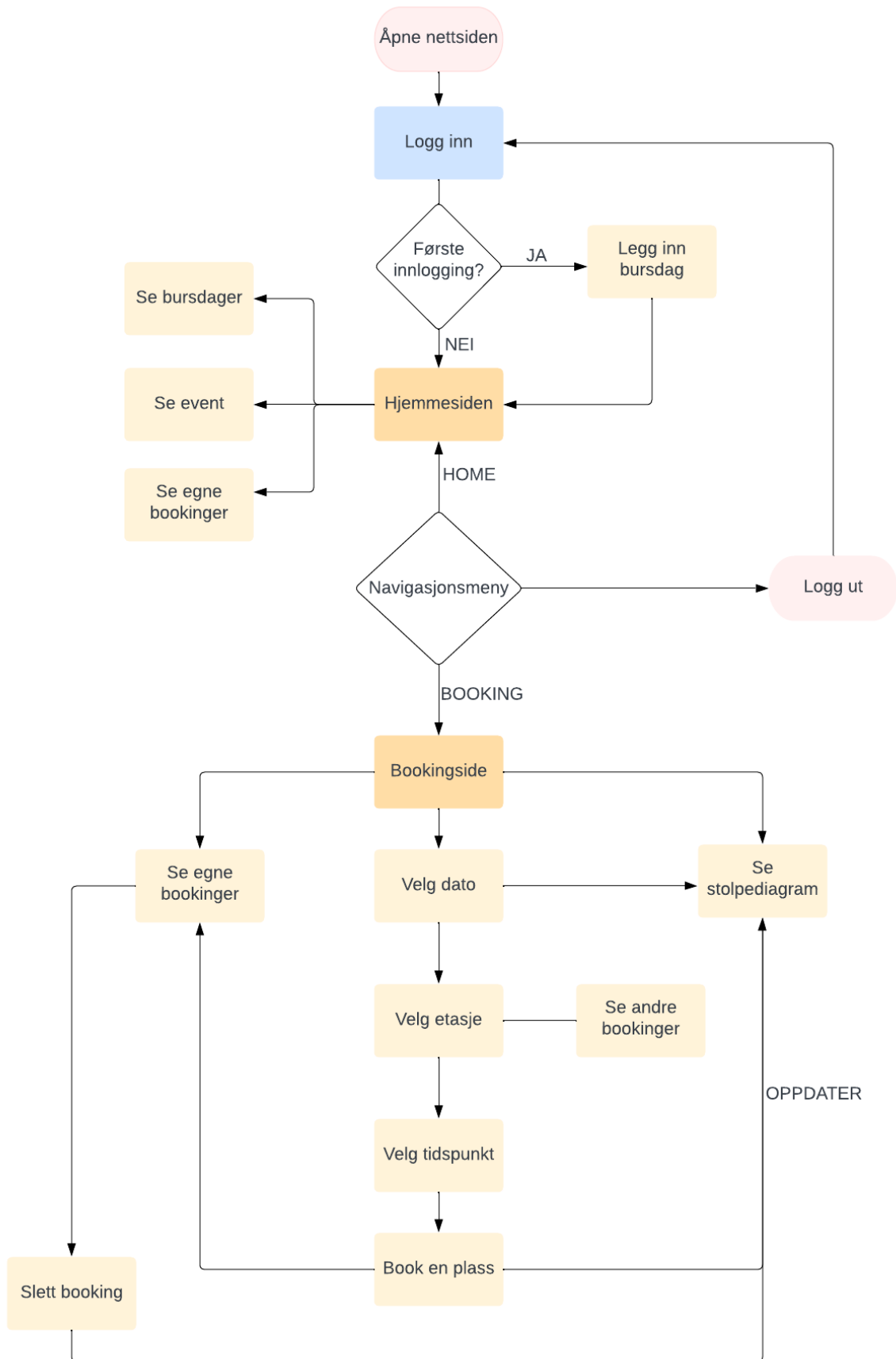
Webapplikasjonen har kun de ansatte hos Evidi som brukere, noe som førte til at vi fokuserte mer på brukervennlighet enn design. Vi har ikke fått noen retningslinjer til designet eller spesifikke krav fra Evidi. Dette har ført til at vi har designet grensesnittet for frihånd. Vi kunne ha brukt ferdige maler for hele nettsiden fra MUI, men vi ønsket å lage egne komponenter som passet funksjonaliteten til programmet. Felles for både hjemmesiden og bookingsiden er at begge har en navigasjonsmeny øverst på siden og et hovedområde som presenterer data. Det er i dette området man finner hovedfunksjonaliteten til programmet.

Innlogging

Innloggingen er avgrenset til brukere som har en Microsoft-konto gjennom Evidi som var et krav fra oppdragsgiveren. En slik konfigurering av applikasjonen gjør det vanskelig for anonyme brukere å komme seg inn på webapplikasjonen. Webapplikasjonen vår er også en dynamisk og knappbasert nettside, som vil si at brukeren ikke trenger å skrive inn informasjon i input-felter. De eneste handlingene inkludert på nettsiden er klikkbare handlinger. Slike typer funksjonaliteter bidrar med å redusere sjansene for web-angrep som for eksempel SQL-angrep og feltmanipulering.

Flytskjema

Figur 5.6 viser en oversikt over all flyten i programmet. Figuren viser at all interaksjon fra brukeren foregår gjennom brukergrensesnittet. Starten av programmet er å åpne nettsiden og avsluttes når man logger ut av applikasjonen.



Figur 5.6: Flytskjema for applikasjonen (laget i lucid.app).

Hjemmesiden

På hjemmesiden, vist i figur 5.7, finner man informasjon om egne bookinger, kommende møter og hvilke ansatte som har bursdag den nåværende måneden. Hovedområdet består av tre komponenter. Disse tre er

- *My Bookings* - Viser bookinger til brukeren for den nåværende uken. Komponenten viser ikke bookinger tilbake i tid, som vil si at man for eksempel ikke kan se gårsdagens booking. Dette valget tok vi for å gjøre komponenten mer oversiktlig for brukeren.
- *Birthdays* - Lister opp alle ansatte som har bursdag den nåværende måneden. Hvis noen har bursdag i dag, flyttes de på toppen av lista og bakgrunnsfargen endres for å fremheve komponenten. Komponenten viser navn, alder og bursdagsdato for hver ansatt.
- *Meetings / Events* - Viser alle eventer som ligger i Outlook-kalenderen til brukeren. Komponententen viser eventer to dager frem i tid.



Evidi Home Booking SW Sigrid Warberg Vestheim sigridwv@stud.ntnu.no LOG OUT

My bookings

- Thursday, 4/20/2023, Glassburet, After lunch
- Friday, 4/21/2023, Glassburet, The whole day

Birthdays

Today's birthdays

Alf Magnus Nohr
26 years old
Date: 4/20/1997

Sigrid Warberg Vestheim
23 years old
Date: 4/2/2000

Henriette Eltvik
22 years old
Date: 4/17/2001

Meetings / Events

Thursday, 20 April 2023

09:00 - 09:15 Morgenmøte
@ Datalab

13:00 - 14:00 Bachelor møte med veileder

Friday, 21 April 2023

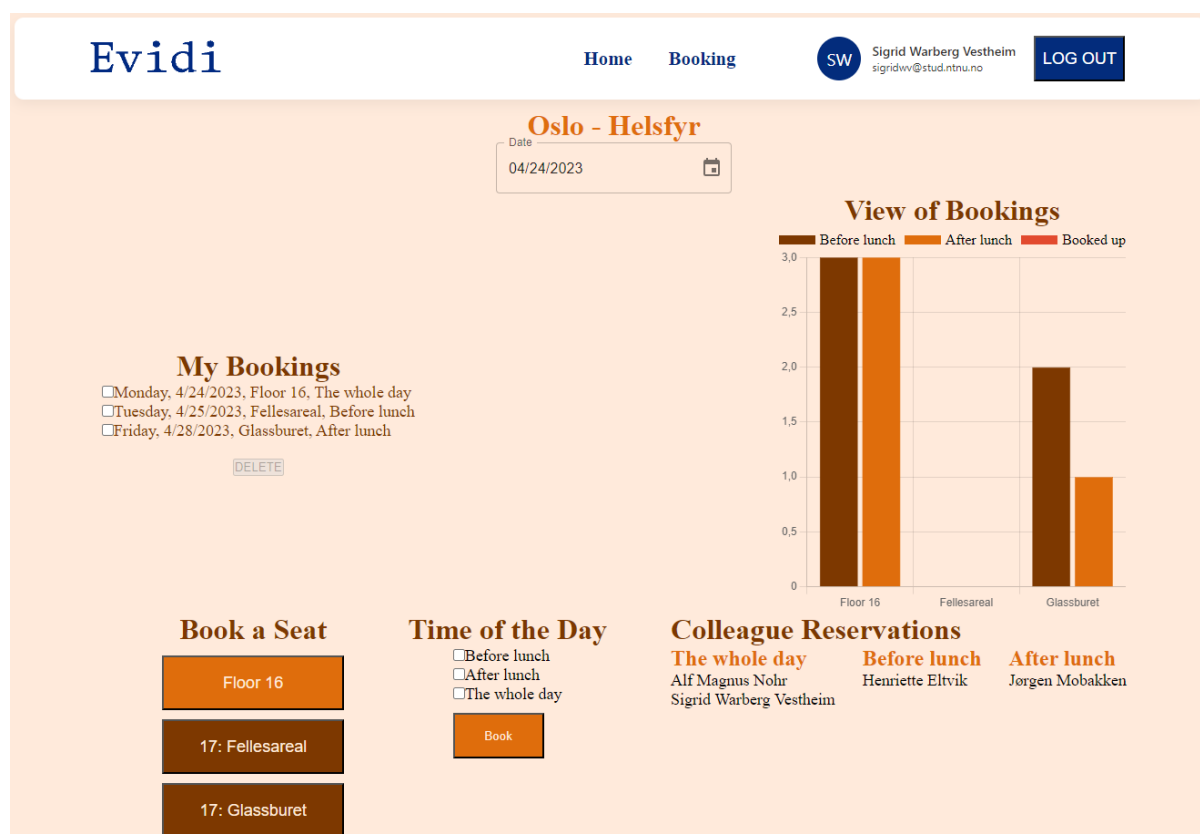
09:00 - 09:15 Morgenmøte
@ Datalab

Figur 5.7: Hjemmesiden i webapplikasjonen.

Bookingsiden

På denne siden finner vi seks komponenter på hovedområdet i tillegg til navigasjonsmenyen som vist i figur 5.8. De seks komponentene er:

- *Date* - Brukeren kan velge dato fra en kalender. Denne datoen blir senere brukt for å reservere og se andre bookinger på denne datoen. Standardverdien til datoen vil være «datoen i dag». Det er ikke mulig å gå tilbake i tid i kalenderen og maksverdien er søndagen i nåværende uke. Hver søndag åpnes en ny uke, som vil si at maksverdien settes til neste søndag igjen.
- *My Bookings* - Her listes reserverasjonene til brukeren for nåværende uke. Hvis en eller flere markeres, er det mulig å slette disse reserverasjonene.
- *View of Bookings* - En enkel måte å holde oversikt over hvor mange som har reservert en plass på en bestemt dato, er å bruke en grafisk fremstilling. Vi har valgt å bruke et stolpediagram for å visualisere dette. Hver stolpe representerer tidsrommet «Før lunsj» og «Etter lunsj». Disse er videre fordelt utover på de ulike etasjene der Evidi har kontor. Stolpediagrammet viser hvor mange reserverasjoner som finnes på en bestemt dato. Disse reserverasjonene inkluderer alle reserverasjoner i alle etasjene, inkludert brukeren sine egne. Når det er fullt i en av etasjene på et gitt tidspunkt, blir søylen rød.
- *Book a Seat* - Velg hvilken etasje man ønsker å sitte i.
- *Time of the Day* - Velg tidspunktet for reserverasjonen og trykk på knappen for å reservere.
- *Colleague Reservations* - Ved å velge en etasje, kan man se andre sine reserverasjoner i valgt etasje for den valgte datoen.



Figur 5.8: Skjerm bilde av bookingsiden.

Fargekoder

Fra Evidi har vi fått tilsendt hvilke fargekoder vi skal bruke. Det er disse fargekodene som også brukes på nettsiden til Evidi [49]. Ved å bruke disse, vil brukeren kjenne igjen fargene og forstå at webapplikasjonen tilhører Evidi. Siden nettsiden til Evidi har flere forskjellige fargenyanser, har vi valgt å ta i bruk tre av disse. Navigasjonsbaren har fått den blå nyansen, akkurat som på siden til Evidi. Se figur 5.9.

HEX #002981	Coated	HEX #0172E2	Coated	HEX #D2F2FF	Coated
RGB 0 / 41 / 129	Pantone C 294	RGB 1 / 114 / 226	Pantone C 2172	RGB 210 / 242 / 255	Pantone C 656
NCS S 3560-R80B	CMYK CP 100/74/0/9	NCS S 1565-B	CMYK CP 81/43/0/0	NCS S 0515-R80B	CMYK CP 10/2/0/0
	Uncoated		Uncoated		Uncoated
	Pantone U 2747		Pantone U 2172		Pantone U 657
	CMYK UP 100/92/0/3		CMYK UP 71/30/0/0		CMYK UP 13/3/0/0

Figur 5.9: Fargekoder blå nyanse.

Startsiden til Evidi har den grønne nyansen, noe som vi har valgt å bruke på hjemmesiden til applikasjonen. Se figur 5.10. Her valgte vi også å bruke den mørkeste nyansen som bakgrunn for å skape et rolig inntrykk av siden.

HEX #1C4E50	Coated	HEX #00AE79	Coated	HEX #B1FFE4	Coated
RGB 28 / 78 / 80	Pantone C 7476	RGB 0 / 174 / 121	Pantone C 2414	RGB 177 / 255 / 228	Pantone C 331
NCS S 5030-B30G	CMYK CP 89/25/38/54	NCS S 1060-G	CMYK CP 74/0/68/0	NCS S 520-B90G	CMYK CP 26/0/20/0
	Uncoated		Uncoated		Uncoated
	Pantone U 4167		Pantone U 346		Pantone U 346
	CMYK UP 75/27/40/38		CMYK UP 64/0/65/0		CMYK UP 64/0/65/0

Figur 5.10: Fargekoder grønn nyanse.

For å skape en kontrast mellom hjemmesiden og bookingsiden, valgte vi den oransje fargen som bakgrunn fra figur 5.11. Her valgte vi en lys bakgrunn, for at de forskjellige elementene skulle komme bedre frem.

HEX #7D3800	Coated	HEX #DF6D0C	Coated	HEX #FFEADB	Coated
RGB 125 / 56 / 0	Pantone C 7567	RGB 223 / 109 / 12	Pantone C 158	RGB 255 / 234 / 219	Pantone C 4029
NCS S 4550-Y70R	CMYK CP 11/59/80/41	NCS S 2070-Y50R	CMYK CP 0/62/97/0	NCS S 0907-Y50R	CMYK CP 3/15/30/0
	Uncoated		Uncoated		Uncoated
	Pantone U 1535		Pantone U 144		Pantone U 4029
	CMYK UP 0/61/100/36		CMYK UP 0/47/100/0		CMYK UP 0/14/28/0

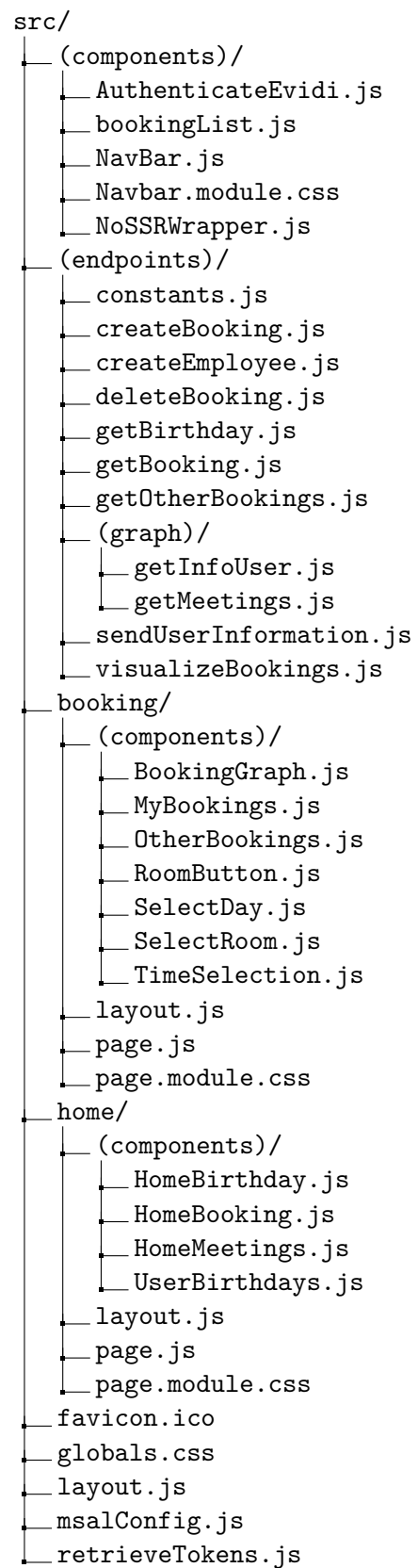
Figur 5.11: Fargekoder oransje nyanse.

5.3.3 Filstruktur frontend

Vi har valgt å dele opp frontenden i forskjellige mapper for å holde en oversiktlig struktur. Som nevnt i seksjon 4.2.2, bruker Next.js en spesiell mappestruktur for å definere endepunktene. Dette gjorde at vi valgte å legge de tilhørende komponentene for *home* og *booking* under hver sin (components)-mappe.

For de globale komponentene, som navigasjonsmenyen, lagde vi en tilsvarende (components)-mappe i rotkatalogen da disse ikke hørte hjemme under en spesifikk side.

Vi valgte også å gruppere alle funksjonene som kalte på API-tjenestene under (endpoints)-mappen der vi videre separerte Microsoft Graph-relaterte API-funksjoner under (graph)-mappen. For å gjøre det enklere å se hvilke HTTP-forespørsel den aktuelle filen håndterer, prefikset vi filnavnene med CRUD-operasjonene som Create, Update og Delete.



Figur 5.12: Filstruktur frontend.

Kapittel 6

Implementasjon

I dette kapitlet skal vi se nærmere på hvordan vi har implementert de forskjellige delene i systemet vårt. Implementeringen av backend presenteres først, deretter presenteres implementeringen av frontend. Videre presenteres kommunikasjonen mellom frontend og backend. På slutten av kapitlet blir overgangen til Next.js forklart.

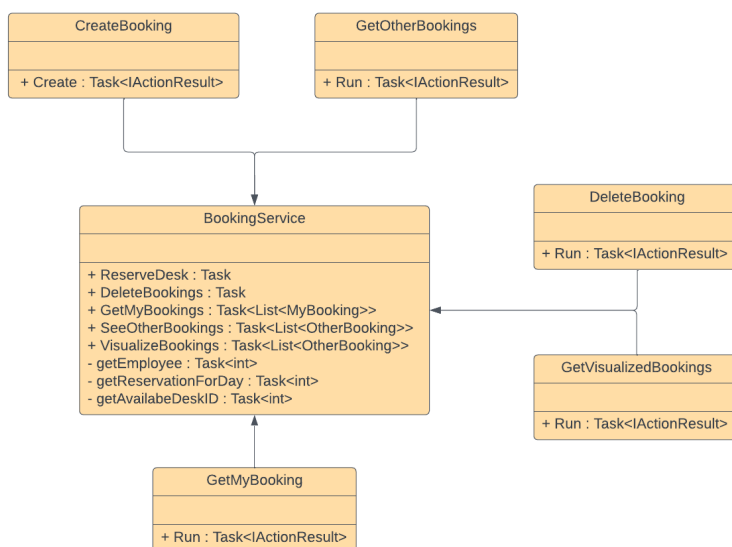
Siden applikasjonen er på engelsk, vil også datoformatet være på formatet MM/DD/YYYY. Dette vil gjelde uansett datoformat på brukerens datamaskin. Årsaken til dette er at webapplikasjonen kjører på en server med engelsk datoformat.

I koden er alle kommentarene på engelsk for å gjøre det lettere for videreutvikling. I tillegg er alle *commit*-meldinger og *pull requester* på engelsk.

6.1 Backend

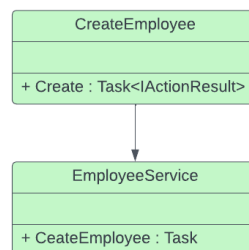
Det er lagd tre klassediagrammer som viser Azure-funksjonene og *servicene* som de bruker. En overordnet beskrivelse av filstrukturen til backend ligger i seksjon 5.2.1.

Figur 6.1 inneholder klassene som går under bookingfunksjonen. Klassene i denne figuren bidrar til å hente og slette reservasjonene som ligger i databasen.

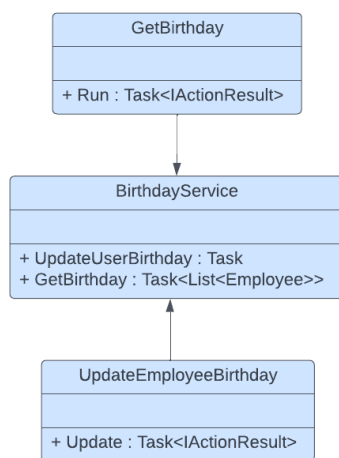


Figur 6.1: Klassediagram av bookingfunksjonalitet (laget i lucid.app).

Figur 6.2 inneholder klasser som går under ansettfunksjonen. Klassen er brukt for å legge til en ansatt automatisk i databasen når de logger inn. Når brukeren blir lagt til i databasen blir bursdagen i databasen satt til en standardverdi. Denne blir senere oppdatert av brukeren manuelt.



Figur 6.2: Klassediagram av ansettfunksjonalitet (laget i lucid.app).



Figur 6.3: Klassediagram av bursdagfunksjonalitet (laget i lucid.app).

Figur 6.3 viser til klassene under bursdagsfunksjonen. Denne funksjonen inneholder to funksjonaliteter: oppdatering av bursdager og henting av bursdager. Klassene bidrar med å visualisere og oppdatere bursdagene som ligger i databasen.

6.1.1 Azure Function

Hver Azure Function må identifiseres med attributten *FunctionName* for å navngi funksjonen i Azure. Se linje 1 i kodeliste 8. Hvert *FunctionName* må være unikt.

For å spesifisere endepunktene og HTTP-metodene som Azure-funksjonene skal lytte til, må *Route* og HTTP-metoden inkluderes i Azure-attributten *HttpTrigger*. Se linje 4 i kodeliste 8. *Route* settes til stien som skal lyttes til og HTTP-metoden spesifiseres for å begrense hvilke typer operasjoner som skal aksepteres. I kodeliste 8, ser vi bruken av dette for endepunktet til *delete-booking*. Her sendes et *delete*-kall til endepunktet i backend.

Listing 8: Unikt funksjonsnavn.

```

1 [FunctionName("delete-booking")]
2 ...
3 public static async Task<IActionResult> Run(
4     [HttpTrigger(AuthorizationLevel.Function, "delete", Route = "booking")]
5     HttpRequest req, ILogger log
6 )
  
```

6.1.2 Azure SQL

For lagring av data, ble det satt opp en Azure SQL-database som bare backenden har tilgang til. For å koble seg opp til databasen, må det oppgis en *connection string* for å åpne tilkoblingen. Denne *stringen* inneholder diverse informasjon, blant annet navnet på databasen og passordet. For at passordet ikke skal synliggjøres i koden, lagret vi passordet i en miljøvariabel som nevnt i 4.6.4.

Listing 9: *Connection string.*

```
1 public static string ConnectionString =
2     $"Server=tcp:sql-db-server-bachelor2023-gr6.database.windows.net, ..."
```

Når API-et i backend ønsker å snakke med databasen, åpnes det en ny tilkobling for hver spørring. Dette gjøres for å følge praksisen om å ikke ha tilkoblingen oppe lenger enn nødvending, men også fordi Azure Function kun kjører når de blir etterspurt. Det ville derfor ikke være mulig å ha en statisk tilkobling over lengre tid. Mer om bruken av SQL forklares under 6.1.4.

6.1.3 Modeller

For at dataene som sendes skal være på et konstant format, blir modeller brukt. En modell er en struktur som Azure-funksjonene i programmet bruker for å spesifisere hvilke data som skal sendes og hvilke som skal bli godtatt. Et eksempel på en modell kan sees i kodeliste 10. Først blir forespørselen fra frontend deserialisert for å hente ut verdiene fra *bodyen*. Deretter blir databaseresponsen serialisert til en JSON-string etter at verdiene er mottatt i databasen. Til slutt blir JSON-*stringen* sendt ut som en respons til frontend.

Listing 10: Modell for henting av reserverasjoner.

```
1 public class MyBooking
2 {
3     public string Date    { get; set; }
4     public bool Am       { get; set; }
5     public bool Pm       { get; set; }
6     public string Floor  { get; set; }
7     public int Id        { get; set; }
8
9 }
```

6.1.4 Services

Som nevnt i 5.2.1 er service-filene de filene som inneholder logikken og håndteringen av SQL-uttrykkene. I kodeliste 11 viser vi hvordan SQL-uttrykket blir sendt til databasen og hvordan dataene blir hentet ut og behandlet. Som vist i kodelisten blir en liste med MyBooking-modeller initialisert basert på resultatet fra SQL-uttrykket.

For å hente nødvendig data fra databasen, ser vi i kodelisting 11 at det gjøres klar en SQL-tilkobling på linje 4. Den blir forsøkt åpnet på linje 6, før vi videre på linje 8 initialiserer SQL-uttrykket. Til slutt blir SQL-uttrykket forsøkt kjørt på linje 20 og om noe skulle gå galt vil disse SQL-funksjonene kaste en feilmelding som programmet senere kan fange.

Om forespørselen er vellykket blir det sendt en respons med en JSON-liste med reserverasjoner som vist i kodelisting 12.

Listing 11: SQL-uttrykk for henting av mine bookinger.

```

1 public static async Task<List<MyBooking>> GetMyBookings(string id)
2 {
3     ...
4     using (var connection = new SqlConnection(DB.ConnectionString))
5     {
6         await connection.OpenAsync();
7         // Sql statement for retrieving reservations.
8         string sql = @"
9             SELECT * FROM [dbo].[Reservation]
10            INNER JOIN [dbo].[Desk]
11                ON [dbo].[Reservation].[d_id] = [dbo].[Desk].[id]
12            INNER JOIN [dbo].[Employee]
13                ON [dbo].[Employee].[id] = [dbo].[Reservation].[u_id]
14            WHERE [dbo].[Employee].[login_id] = '{id}' AND
15            CAST(_date AS DATE) >= CAST(GETDATE() AS DATE)
16            ORDER BY _date";
17
18        using (var command = new SqlCommand(sql, connection))
19        {
20            var reader = await command.ExecuteReaderAsync();
21            while (reader.Read())
22            {
23                var row = new MyBooking()
24                {
25                    // Format the data by using a model.
26                    Date = (string)reader["_date"].ToString(),
27                    Am = (bool)reader["am"],
28                    Pm = (bool)reader["pm"],
29                    Floor = (string)reader["floor"],
30                    Id = (int)reader["id"]
31                };
32                myBookings.Add(row);
33            }
34        }
35        ...
36    }
37 }

```

Listing 12: JSON-respons fra GetMyBooking.

```
1 [{"Date": "05.03.2023 00:00:00", "Am": 0, "Pm": 1, "Floor": "17", "Id": 29}, ...]
```

6.1.5 Endepunkter

GET

Klassene med SQL-uttrykk og klassene med endepunktene er skilt fra hverandre. Dette gjør at selve koden til endepunktene har samme oppbygging. Endepunktene som er lagd inneholder forskjellige metoder og operasjoner som de utfører. For uthenting av data, bruker applikasjonen spørringer. Kodeliste 13 viser hvordan spørringene er implementert i programmet. Først mottar endepunkt-koden en bruker-ID gjennom en forespørsel sendt fra brukeren. Deretter blir ID-en sendt til en metode som sender verdien videre til databasen.

Listing 13: Funksjon for henting av data.

```
1 public static async Task<IActionResult> Run(  
2     [HttpTrigger(AuthorizationLevel.Function, "get", Route = "employee/booking")]  
3     HttpRequest req, ILogger log  
4 )  
5 {  
6     //Retrieve id from query.  
7     string ?id = req.Query["id"];  
8     ...  
9     myBookings = await BookingService.GetMyBookings(id);  
10    ...  
11    //Serialize response from database to JSON format.  
12    var data = JsonConvert.SerializeObject(myBookings);  
13    ...  
14 }
```

POST

Når vi ønsker å slette eller legge til data i databasen blir dataen sendt i en *JSON-body*. Dette gjør det enkelt å behandle og hente ut verdier fra forespørselen. For å hente ut dataene fra en *body* blir den deserialisert til å passe en modellstruktur. Kodeliste 14 viser hvordan dataene blir hentet ut og brukt.

Listing 14: Funksjon for å legge til data.

```

1 public static async Task<IActionResult> Create(
2     [HttpTrigger(AuthorizationLevel.Function, "post", Route = "booking")]
3     HttpRequest req, ILogger log
4 )
5 {
6     // Get body from POST Request.
7     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
8     // Convert body to Reservation model.
9     dynamic data = JsonConvert.DeserializeObject<Reservation>(requestBody);
10    ...
11    // Use values from request body.
12    await BookingService.ReserveDesk(data.UserId, data.Floor,
13        data.Date, data.AM, data.PM);
14    ...
15 }

```

6.1.6 Timer Trigger

Timer Trigger er brukt for å slette alle reservasjonene fra databasen for et gitt tidspunkt. Biblioteket *NCRONTAB* blir brukt for å spesifisere hvilken dato Azure Function skal utløses på. I dette uttrykket kan man spesifisere hvilken dag og tid en operasjon skal bli utløst. Kodelisting 15 viser at *NCRON*-uttrykket er satt til å slette alle reservasjoner kl. 00:00 hver lørdag [38].

Listing 15: Kodeliste på Timer Trigger.

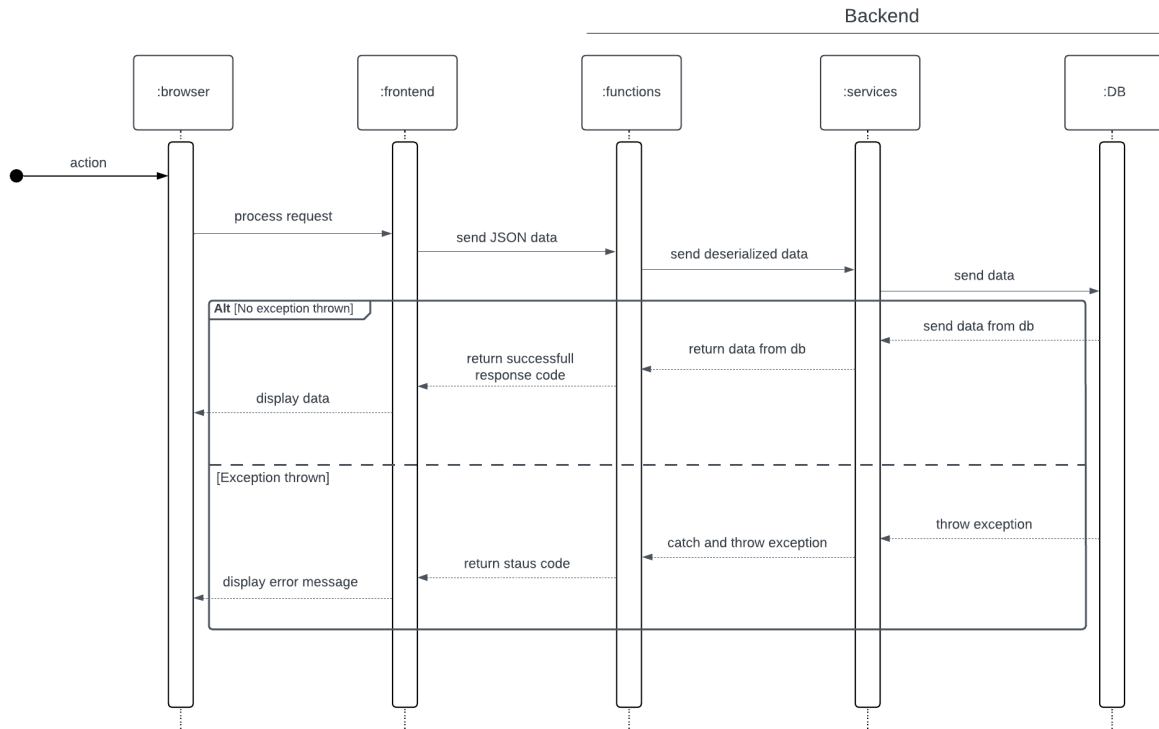
```

1 //Remove bookings at 00:00 on Saturday.
2 [FunctionName("remove-reservation")]
3 public static async Task Run([TimerTrigger("0 0 0 * * Sat")] TimerInfo myTimer,
4 ILogger log)
5 {
6     ...
7     //Remove reservations.
8     await TriggerReservations();
9     ...
10 }

```

6.1.7 Feilhåndtering

Problemer som kan oppstå i SQL-uttrykkene av programmet, vil kaste en feilmelding til Azure Function. Funksjonene vil dermed håndtere disse feilmeldingene ved bruk av en *try-catch*-blokk. Når feilmeldingen er behandlet, vil funksjonene i backend returnere en statuskode. Statuskoden blir brukt i frontend for å varsle brukeren om en ugyldig handling eller om en feil i backend eller databasen. Figur 6.4 illustrerer hvordan feil blir håndtert mellom databasen, backend, frontend og nettleseren.



Figur 6.4: Sekvensskjema for feilhåndtering av handlinger (laget i lucid.app).

6.2 Frontend

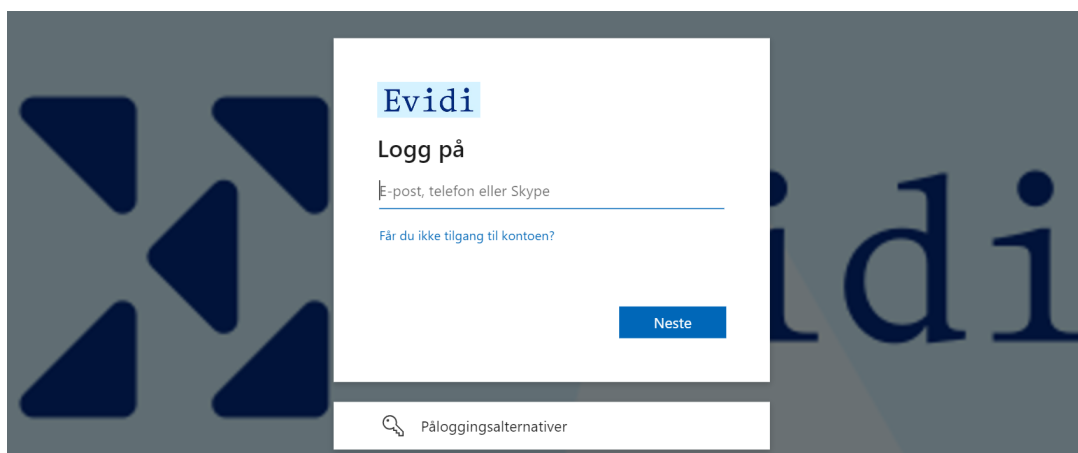
Vi har brukt React og Next.js som rammeverk, som er beskrevet henholdsvis i seksjon 4.2.1 og 4.2.2. Noen av komponentene har vi selv designet, mens andre er hentet fra MUI Datepicker, MUI Dialog, Microsoft Graph Toolkit og Chart.js. Disse er henholdsvis beskrevet i seksjon 4.2.5, 4.2.4 og 4.2.6. Komponentene i frontend blir nærmere forklart under i dette kapittelet.

Koden i frontend er fordelt på komponenter. For eksempel «My Bookings»-komponenten vil vise alle reserverasjoner som den innloggede brukeren har. Årsaken til at vi delte koden inn i komponenter, var at filstrukturen skulle bli mer oversiktlig samt å få en modulær struktur. Se filstruktur i seksjon 5.3.3.

Vi skilte også mellom *styling*-verdier og funksjonell kode. *Styling*-verdiene ble plassert i en CSS-fil, mens all funksjonell kode ligger i JavaScript-filer. Denne fordelingen gjør det enklere for videreutviklere å forandre på *stylingen* hvis nødvendig.

6.3 Innloggingsside

For å få tilgang til webapplikasjonen, må man logge inn med eksisterende Microsoftkonto gjennom Evidi. Figur 6.5 viser et skjermbilde av det første møtet brukeren ser av webapplikasjonen. Etter vellykket autentisering, vil brukeren bli sendt videre til hjemmesiden.



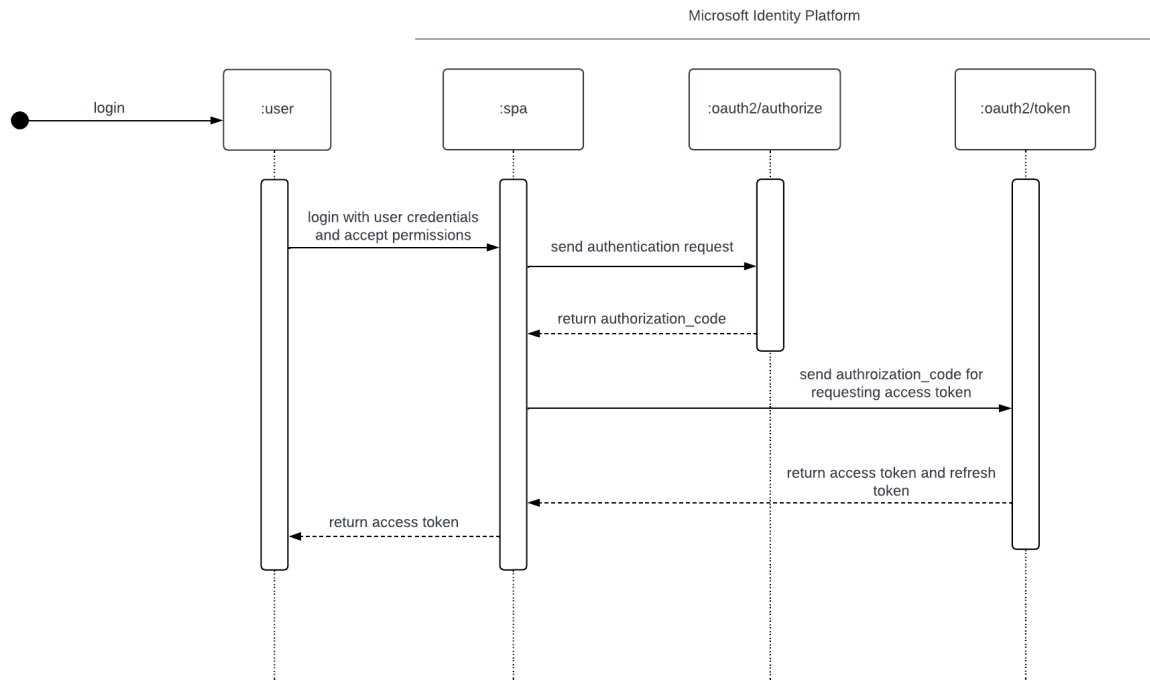
Figur 6.5: Innloggingssiden til applikasjonen.

6.3.1 Innloggingsflyt

Figur 6.6 viser innloggingsflyten mellom brukeren og Microsoft Identity Platform. Tegningen er inspirert av sekvensdiagrammet fra Microsoft [50].

Sekvensdiagrammet inkluderer objektene:

- *User* som er brukeren av webapplikasjonen.
- *SPA* som står for *Single-Page Application*. Den representerer den registrerte applikasjonen i Azure AD.
- To endepunkter som Microsoft Identity Platform tilbyr under autentiseringsprosessen av brukeren [51]:
 1. `/authorize` som verifiserer brukeren og sender en autoriseringskode etter vellykket autentisering.
 2. `/token` som sender en gyldig tilgangstoken, som kan brukes sammen med eksterne API-er.



Figur 6.6: Sekvensskjema for innloggingsfunksjonalitet (laget i lucid.app).

Diagrammet viser først en bruker som har skrevet inn brukerinformasjonen sin til *SPA*. Deretter sender *SPA* en autoriseringsforespørsel til *authorize*-endepunktet. Etter brukeren er verifisert returnerer *Authorize*-endepunktet en autoriseringskode som blir brukt for å be om en tilgangstoken fra *token*-endepunktet.

6.3.2 Oppsett

For at applikasjonen skal ha mulighet til å hente ut tilgangstoken og bruker-ID, må applikasjonen være registrert i Azure AD. Dette er for å opprette tillit mellom Microsoft Identity Platform og applikasjonen vår [52].

Etter opprettelsen av applikasjonsregistreringen, lager vi en konfigurasjonsfil i frontend. Denne filen inneholder data som må være spesifisert for at applikasjonen vår skal være koblet opp mot Microsoft Identity Platform. I kodelisting 16 vises konfigurasjonsfilen brukt i applikasjonen. Variablene *clientId* og *tenantId* er hentet ut fra applikasjonsregistreringen i Azure AD.

Listing 16: Konfigurasjonsfilen i programvaren.

```

1 export const msalConfig = {
2   auth: {
3     // ID for the application.
4     clientId: "<clientId>",
5     // ID for the organization.
6     authority: "https://login.microsoftonline.com/<tenantId>",
7     redirectUri: "/home",
8   }
9 }

```

```

10
11 // Permissions used in Microsoft Graph.
12 export const loginRequest = {
13   scopes: [
14     'User.Read',
15     'Calendars.Read.Shared',
16     'Calendars.Read',
17     'Calendars.ReadBasic',
18   ]
19 };

```

Nedenfor følger en liste som gir en detaljert beskrivelse av parametrene i *msalConfig* og konstanten *loginRequest* som er inkludert i konfigurasjonsfilen [53].

- **authority** er en URI for organisasjonen som MSAL bruker for å autentisere og autorisere brukeren.
- **clientId** er appID-en for applikasjonen. Dette er ID-en som blir generert når en applikasjons registrering blir opprettet i Azure AD.
- **redirectUri** må være definert for å motta en autoriseringskoderespons på URI-en som blir spesifisert. URI-en definert i konfigurasjonsfilen må også være definert i applikasjonsregistreringen på Azure AD.
- **loginRequest** er en konstant som inneholder tillatelser som brukeren må godkjenne for å hente ut data fra Microsoft Graph.

6.3.3 Henting av token

For å hente ut tilgangstoken og bruker-ID, brukte vi biblioteket «msal-react» [54]. Biblioteket inkluderer funksjoner som *instance* og *accounts*. *Instance* er en instanse av grensesnittet *PublicClientApplication*, som blir brukt for å konfigurere tilgangen til Microsoft Identity Platform og hente ut tilgangstoken [55] [56]. *Accounts* holder informasjonen til den autoriserte brukeren [57]. I kodeliste 17 vises det hvordan et tilgangstoken blir hentet ut fra applikasjonen ved hjelp av «msal-react» biblioteket.

Listing 17: Henting av tilgangstoken ved hjelp av MSAL.

```

1  const [accessToken, setAccessToken] = useState('');
2  //Initialize the variables instance and accounts from <<msal-react>>
3  const {instance, accounts} = useMsal();
4
5      useEffect(() => {
6        // Request including config file and account information.
7        const request = {
8          ...loginRequest,
9          account: accounts[0]
10       };
11
12  const fetchData = async () => {
13    //Fetch a token without prompting the user for multiple sign in requests.
14    await instance.acquireTokenSilent(request).then((res) => {
15      setAccessToken(res.accessToken);
16    });
17  }
18  fetchData()
19 }, [accounts, instance]);

```

På linje 14 valgte vi å bruke `acquireTokenSilent` som henter ut et tilgangstoken fra brukeren, uten at brukeren trenger å skrive inn innloggingsinformasjonen sin på nytt. Dette gjør det mer brukervennlig ettersom brukeren bare trenger å logge inn når de besøker webapplikasjonen. Metoden `acquireTokenSilent` tar inn en `request`. `Request` er en konstant som inkluderer parametrene som er spesifisert i konfigurasjonsfilen og informasjonen fra profilen til brukeren.

6.3.4 Autorisering

`PublicClientApplication`-grensesnittet fra «msal-browser» blir brukt for å initialisere konfigurasjonsfilen som vist på linje 1 i kodeliste 18. `MSALProvider` er nestet rundt alle komponentene som bruker MSAL-biblioteket [58]. Dette er komponenter som trenger autorisering for å kunne bli aksessert.

For å sjekke om brukeren har fått tilgang eller ikke, inkluderer «msal-react» `AuthenticatedTemplate`- og `UnauthenticatedTemplate`-komponenter [59]. Disse komponentene blir brukt i `layout.js` filene for å knytte alle sidene som trenger autorisering. Hvor `layout.js` befinner seg i forhold til programmet, er forklart i seksjon 5.2.1.

`AuthenticatedTemplate`- og `UnauthenticatedTemplate`-komponentene vil sjekke om brukeren har rettigheter og bare vise innholdet som er nestet i komponentet. I dette tilfellet blir `{children}` nestet under `AuthenticatedTemplate` som gjør at alle komponentene til webapplikasjonen blir lastet inn. For `UnauthenticatedTemplate` blir ikke applikasjonen vist og man blir sendt til innloggingsiden. Kodelisting 18 viser en oversikt over implementering av `AuthenticatedTemplate` og `UnauthenticatedTemplate`.

Listing 18: `Authenticated`- og `UnauthenticatedTemplate`.

```

1  const pca = new PublicClientApplication(msalConfig);
2  ...
3  <MsalProvider instance={pca}>
4    <!--Displays nothing.-->
5    <UnauthenticatedTemplate>
6      <AuthenticateEvidi/>
7    </UnauthenticatedTemplate>
8
9    <AuthenticatedTemplate>
10   <!--Displays site if authenticated.-->
11     <AuthenticateEvidi/>
12     {children}
13   </AuthenticatedTemplate>
14 </MsalProvider>

```

6.3.5 Opprettelse av bruker

Når brukeren logger inn for første gang blir fornavn, etternavn og bruker-ID til brukeren sendt til backend. Bruker-ID er en unik ID for hver bruker, som kan bli hentet ut fra tilgangstokenet til brukeren eller direkte fra brukerprofilen. I applikasjonen har vi gjort sistnevnte. Kodeliste 19 viser hvordan applikasjonen henter ut bruker-ID fra profilen til brukeren og setter den inn i en informasjonskapsel. Siden det ikke blir sendt inn en bursdag til backend når brukeren blir opprettet, vil databasen sette en standardverdi for bursdagen. Brukeren må senere oppdatere denne verdien gjennom et `pop-up`-vindu. Implementeringen av `pop-up`-vinduet blir beskrevet i underseksjon 6.3.5.

Listing 19: Kodeliste for videresending av bruker når logget inn.

```

1  if (!isAuthenticated && inProgress === InteractionStatus.None) {
2      instance.loginRedirect(loginRequest);
3  }else{
4      // Get account that is signed in.
5      let accounts = instance.getAllAccounts();
6      if (accounts){
7          let userID = accounts[0].localAccountId
8          // Set userID as a cookie for later use.
9          cookieCutter.set('userID', userID)
10     }
11 }

```

Kodeliste 20 viser hvordan navnet til brukeren blir hentet fra Microsoft Graph sitt endepunkt <https://graph.microsoft.com/v1.0/me>, som er forklart i detalj i seksjon 6.3.7. Videre blir både fornavn og etternavn til brukeren lagt til i informasjonskapselen.

Listing 20: Setter fornavn og etternavn i informasjonskapsel.

```

1  getInfoUser(response.accessToken).then((res)=>{
2      // DisplayName contains both firstname and surname.
3      const lastSpaceIndex = res.displayName.lastIndexOf(' ');
4      const firstNames = res.displayName.substring(0, lastSpaceIndex);
5      const lastName = res.displayName.substring(lastSpaceIndex + 1);
6      // Set the name of the user in cookie.
7      cookieCutter.set('userFirstNames', firstNames)
8      cookieCutter.set('userLastName', lastName)
9  })

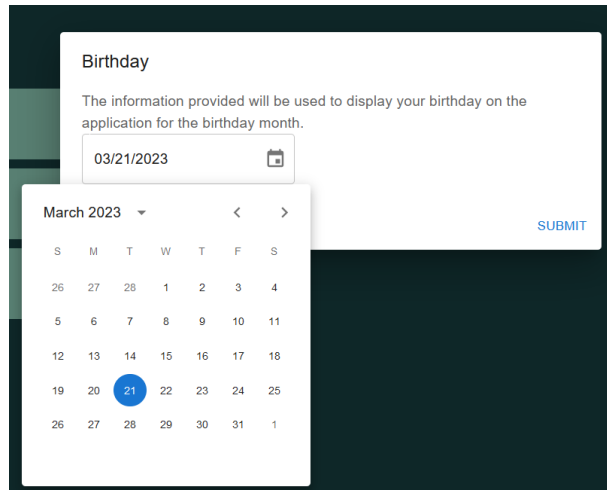
```

Når programmet skal opprette brukeren, henter det bruker-ID, fornavnet og etternavnet til brukeren fra informasjonskapselen og sender disse verdiene til backenden.

Bursdags-pop-up

Bursdagen til de ansatte hos Evidi ligger lagret i et annet system som vi ikke hadde tilgang til. Etter diskusjon med oppdragsgiver kom vi frem til at det enkleste ville være å lagre bursdagen som en attributt i databasen. Da kan Evidi i senere tid legge inn bursdag til tilhørende bruker her. I mellomtiden måtte vi finne en løsning for å legge til bursdag når brukeren er innlogget på webapplikasjonen. Det ble vurdert å legge til en knapp på navigasjonsmenyen der man kunne legge inn bursdag, men dette ville føre til at brukeren kunne legge til bursdagen sin flere ganger. For å forhindre dette, valgte vi å bruke et *pop-up*-vindu som bare ville synes første gang brukeren logget inn.

Figur 6.7 viser til dette *pop-up*-vinduet der brukeren må legge til bursdagen sin. På linje 4 i kodeliste 21 er det satt en sjekk som skal hindre brukeren i å klikke seg vekk fra *pop-up*-vinduet. Formålet med dette er å sikre at brukeren fullfører interaksjonen med *pop-up*-vinduet før de kan gå videre.



Figur 6.7: Velg bursdag.

Listing 21: Kode for visning av *pop-up*-vindu.

```

1  ...
2  <Dialog open={open} onClose={ (event, reason) => {
3    //Checks if background of dialog is clicked.
4    if (reason !== "backdropClick") {
5      handleClose()
6    }
7  }}>
8  <DialogTitle>Birthday</DialogTitle>
9  <DialogContent >
10   <DialogContentText>
11     The information provided will be used to display your birthday on
12     the application for the birthday month.
13   </DialogContentText>
14   <LocalizationProvider dateAdapter={AdapterDayjs}>
15     <DatePicker
16       ...
17     />
18   </LocalizationProvider>
19 </DialogContent>
20 <DialogActions>
21   <Button onClick={handleClose}>Submit</Button>
22 </DialogActions>
23 </Dialog>
24  ...

```

Som vist i kodeliste 21 er en *Dialog*-komponent brukt for å implementere *pop-up*-funksjonaliteten. Inni komponenten ligger *DialogContent* og *DialogActions*. *DialogContent* inneholder en imple-

mentert kalender fra *DatePicker*. Implementasjonen av *DatePicker* er beskrevet i seksjon 6.5.2. *DialogActions* inkluderer en knapp som bruker *handleClose*-funksjonen vist i kodeliste 22. Funksjonen sender bursdagen til backenden når brukeren har valgt en dato.

Neste steg var å sikre at vinduet bare vises første gang brukeren logger inn. Vi vurderte å kalle på backenden og sjekke hvilken verdi som lå lagret under bursdag. Hvis verdien var en standardverdi, kunne vinduet vises. Hvis ikke, trengtes ikke vinduet å dukke opp. Problemet med dette er at det blir mye unødvendig kalling på backenden som krever ressurser. Dermed tok vi heller i bruk *localStorage* for at denne sjekkinga skulle skje i frontend.

Listing 22: Kode for bruk av *localStorage*.

```
1  useEffect(() => {
2      // Fetches the value of the variable "shown".
3      const shown = localStorage.getItem("shown")
4      // Checks if "shown" is false.
5      if (!shown){
6          setOpen(true);
7      }
8  }, []);
9
10 const handleClose = async () => {'
11     // Sets the value of the variable "shown".
12     localStorage.setItem("shown", true);
13     setOpen(false);
14     if (idToken && birthday){
15         await updateBirthday(birthday, idToken);
16     }
17     await window.location.reload();
18 };
```

Det blir lagret en variabel i *localStorage* som blir satt til *true* når brukeren har angitt en bursdag og trykket på *submit*-knappen i *Dialog* vinduet, som vist på linje 12 i kodeliste 22. Linje 5 vil sjekke om variabelen er satt til *false*. Hvis den er *true* blir ikke brukeren oppfordret til å legge inn bursdagen sin på nytt.

Fordelen med *localStorage* er at det er enkelt å sette opp og ta i bruk. Ulempen er at den er lokal for hver nettleser, som vil føre til at *pop-up*-vinduet uansett vil vises om brukeren logger inn på en ny nettleser eller et privat vindu.

6.3.6 Logg ut

Applikasjonen inkluderer en utloggingsknapp for at brukeren har mulighet til å logge ut når de ønsker det. Utloggingsknappen ble satt på navigasjonsbaren slik at brukeren kan logge ut av applikasjonen uavhengig av hvilken side de befinner seg på. Kodeliste 23 viser hvordan funksjonaliteten er implementert i applikasjonen.

Listing 23: Kode for logg ut-funksjonalitet.

```

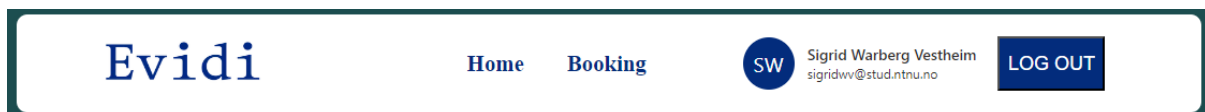
1  const myMsal = new PublicClientApplication(loginRequest);
2  const logoutRequest = {
3      account: accounts[0],
4      mainWindowRedirectUri: "/home",
5  };
6  await myMsal.logoutPopup(logoutRequest)

```

Logg ut-funksjonaliteten inkluderer en *logoutRequest*-konstant som inkluderer en *account*-variabel og en *mainWindowRedirectUri*-variabel. Variabelen *account* er brukeren som skal logge ut og *mainWindowRedirectUri* sender brukeren videre til innloggingssiden til applikasjonen [60].

6.3.7 Navigasjonsmeny

I navigasjonsmenyen finner vi informasjon om den innloggede brukeren, med navn og e-post. Se figur 6.8. Dette elementet er implementert med hjelp av Microsoft Graph Toolkit.



Figur 6.8: Navigasjonsmenyen, skjermbilde fra webapplikasjonen.

Microsoft Graph Toolkit - Person

Vi ønsker å hente ut informasjon fra den innloggede brukeren. Ved å kalle på Microsoft Graph sitt endepunkt <https://graph.microsoft.com/v1.0/me> og sende med tilgangstokenet til brukeren, vil man hente både navn og e-post fra brukeren. Kodeliste 24 viser hvordan programmet kaller på API-et.

Listing 24: `getInfoUser.js`: Hente data fra Microsoft Graph om brukeren.

```

1  export async function getInfoUser(token){
2      const options={
3          method:'GET',
4          headers:{
5              'Content-Type':'application/json',
6              'Origin':'',
7              'Authorization':"Bearer "+token
8          }
9      }
10     // GET_INFO_USER = https://graph.microsoft.com/v1.0/me
11     // Calling the url.

```

```
12     const res = await fetch(GET_INFO_USER, options);
13     if (!res.ok){
14         throw new Error("Failed to fetch data: " + res.statusText)
15     }
16     return res.json();
17 }
```

Noe av resultatet fra API-et er vist i kodeliste 25. I programmet lagres dette resultatet i en *struct* med variablene *displayName* og *mail*.

Listing 25: Noe av resultat fra Microsoft Graph.

```
1 {
2     ...
3     "displayName": "Sigrid Warberg Vestheim",
4     "mail": "sigridwv@stud.ntnu.no",
5     ...
6 }
```

Videre bruker vi komponenten *Person* fra Microsoft Graph Toolkit som er beskrevet i seksjon 4.2.4. For å bruke denne komponenten, må vi bruke *displayName* og *mail* fra kodelisting over. Disse ligger lagret i en *struct* og kodeliste 26 viser hvordan *Person* blir implementert ved hjelp av denne *structen*.

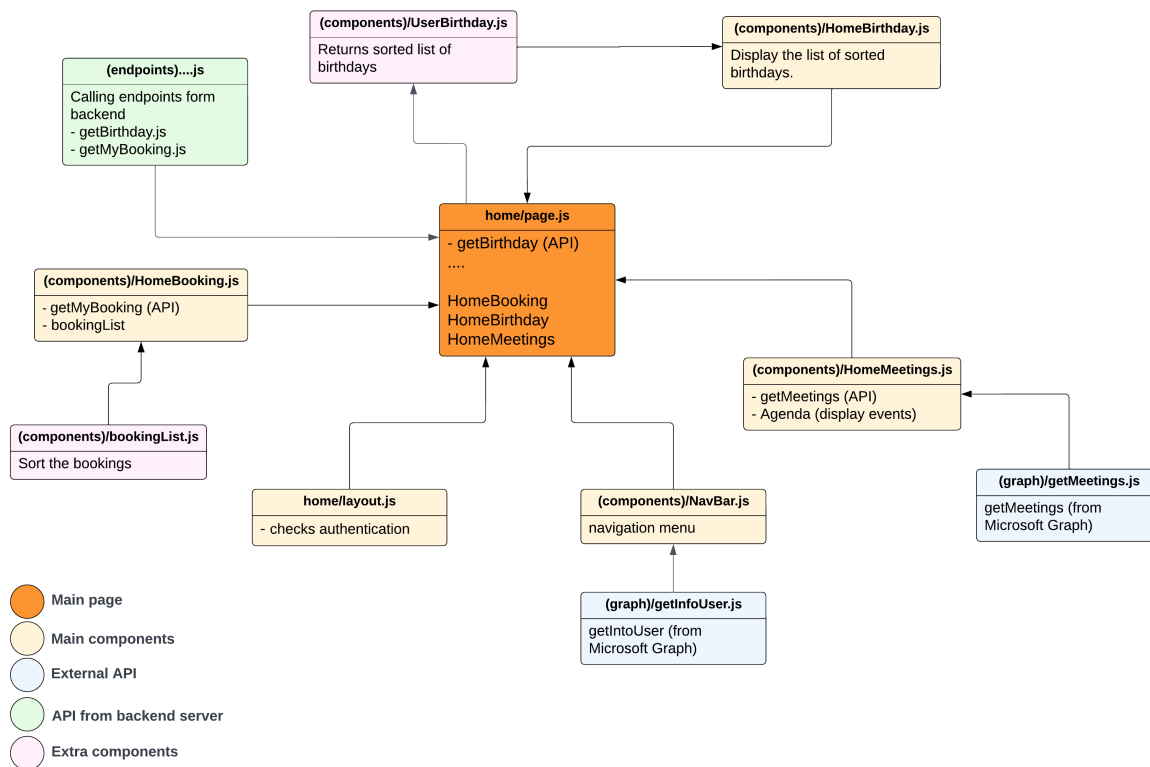
Listing 26: NavBar.js: Personkomponentet i navigasjonsmenyen.

```
1 <nav className={styles.Nav}>
2     ...
3     <Person
4         // Sets the background color to blue.
5         className={styles.Person}
6         // Initialize personDetails with userProfile data.
7         personDetails={userProfile}
8         // The view of the personDetails should be on two lines.
9         view={PersonViewType.twolines}
10    />
11     ...
12 </nav>
```

6.4 Hjemmeside

Figur 6.9 viser kodestrukturen over hjemmesiden til applikasjonen. Videre forklares det hvordan alt henger sammen.

- I mappen (endpoints) finner vi API-er som kaller på backenden. Disse endepunktene blir kalt fra hovedsiden fra filen *home/page.js*.
- Fra hovedsiden sendes resultatet av kallet på API-et *getBirthday* inn i *UserBirthday.js*. Her sorteres bursdagene før de blir presentert i hovedkomponenten *HomeBirthday*.
- I filen *getMeetings.js* kaller vi på Microsoft Graph og henter ut eventer fra kalenderen til brukeren. Resultatet av dette sendes videre inn i *HomeMeetings.js*.
- I filen *getInfoUser.js* kaller vi også på Microsoft Graph og henter informasjon om brukeren. Dette blir videre presentert i navigasjonsbaren.
- Filen *layout.js* passer på at brukeren hele tiden er autentisert gjennom Evidi.
- Fra hovedsiden sendes resultatet av kallet på API-et *getMyBooking* inn i *HomeBooking.js*. Disse blir videre sortert i filen *bookingList.js* før de igjen blir visuelt presentert i *HomeBooking.js*.



Figur 6.9: Kodestruktur hjemmeside (laget i lucid.app).

6.4.1 Møter / arrangement

Ved å bruke Microsoft Graph, beskrevet i seksjon 4.2.3, henter vi ut data fra Microsoft Outlook-kalenderen. Disse presenteres visuelt ved hjelp av komponenten Agenda fra Microsoft Graph Toolkit som vist i figur 6.10.



Figur 6.10: Møter / arrangement.

Microsoft Graph Toolkit - Agenda

Ved å bruke *Agenda*, får vi en oversiktlig visuell presentasjon av eventene som ligger i kalenderen til brukeren. Programmet kaller på <https://graph.microsoft.com/v1.0/me/calendarview> med en startdato og en sluttdato som utgjør et tidsintervall på to dager frem i tid, som vist i kodeliste 27. I tillegg sendes også tilgangstokenet sammen med brukeren for å lese av riktig kalender.

Listing 27: `getMeetings.js`: Henter data fra kalenderen ved hjelp av Microsoft Graph.

```

1  export async function getMeetings(token, startDate, endDate){
2      const options={
3          method:'GET',
4          headers:{
5              'Content-Type':'application/json',
6              'Origin':'',
7              'Authorization':"Bearer "+token
8          }
9      }
10     // Get events based on startDate and endDate.
11     const url=GET_MEETING_USER_START_DATE+startDate+GET_MEETING_USER_END_DATE+endDate
12     const res = await fetch(url, options);
13     // Check error.
14     if(!res.ok){
15         throw new Error("Failed to fetch meetings: " + res.statusText)
16     }else{
17         return res.json();
18     }
19 }

```

Videre må vi behandle dataene fra Microsoft Graph og lage en mal for hvordan vi vil presentere eventene. Kodeliste 28 viser hvordan vi lager en mal for hvordan vi vil at Agenda-komponenten skal se ut.

Listing 28: HomeMeetings.js: Lager mal for hvordan vi vil presentere dataen i Agenda-komponentet.

```
1 const renderMeeting = (props) => {
2   const meeting = props.dataContext;
3   return (
4     <div>
5       <h3>{meeting.subject}</h3>
6       <p>{meeting.start.dateTime}</p>
7       <p>{meeting.end.dateTime}</p>
8     </div>
9   );
10  };
```

Kodeliste 29 viser hvordan Agenda-komponenten presenterer dataen fra *meetingsData* ved å bruke malen *renderMeeting*.

Listing 29: HomeMeetings.js: Agenda-komponentet.

```
1 <div className = {styles.HomeInfoBoard}>
2   <h1>Meetings / Events</h1>
3   <div>
4     <Agenda
5       //Sets the color of the day to blue.
6       className={styles.Agenda}
7       //Groups the events by days.
8       groupByDay={true}
9       //Initialize events with meetingsData data.
10      events={meetingsData}
11      //Visualize the events by using the template renderMeeting.
12      eventTemplate={renderMeeting}
13    />
14  </div>
15 </div>
```

6.4.2 Bursdager denne måneden

Basert på måneden, henter vi ut bursdager fra databasen. Kodelisting 30 viser hvordan programmet kaller på API-et fra backend og returnerer et JSON-objekt. Videre blir resultatet fra dette sortert, der bursdagene står i stigende rekkefølge. Hvis noen har bursdag i dag, havner komponenten øverst og bytter farge. Se figur 6.11.



Figur 6.11: Bursdagskomponenter, skjermbilde tatt 20.april 2023.

Listing 30: getBirthday.js: viser hvordan vi kaller på API fra backend.

```

1  const options = {
2    method: 'GET',
3    headers: {
4      'Content-Type': 'application/json',
5      'Origin': '',
6      'x-functions-key': process.env.REACT_APP_FUNCTION_HOST_KEY
7    }
8  }
9  export async function getBirthday(month){
10   let url = GET_BIRTHDAY + month.toString()
11   const res = await fetch (url, options);
12
13   if(!res.ok){
14     throw new Error("Failed to fetch birthday: " + res.statusText)
15   }else{
16     return res.json();
17   }
18 }

```

6.4.3 Mine bookinger

Basert på ID-en til brukeren, henter vi ut alle bookinger for den innloggede brukeren fra databasen. *BookingList* er en funksjon som tar inn en liste med bookinger fra databasen som parameter, som vist i kodelisting 31. Funksjonen itererer gjennom listen og lager ny komponent *Booking* for hver iterasjon. *Booking*-komponenten har flere variabler som settes basert på attributtene fra databasen. Disse variablene er tilhørende dag, dato, etasje og tidspunkt for bookingen. Til slutt returneres en liste med leselige bookinger.

My Bookings

- Monday, 4/24/2023, Floor 16, The whole day
- Tuesday, 4/25/2023, Fellesareal, Before lunch
- Wednesday, 4/26/2023, Glassburet, After lunch

Figur 6.12: Liste med mine bookinger.

Listing 31: bookingList.js: lager en liste med bookinger som skal presenteres for brukeren.

```

1  export default function BookingList(bookings) {
2      ...
3      // Loop through my bookings.
4      for(let i = 0; i < bookings.length; i++ ) {
5          ...
6          // The date of the booking.
7          const date = bookings[i].Date.split(" ")[0];
8          // Finds the day number.
9          dayNumber = new Date(date).getDay()
10
11         // Find the correct time of the reservation.
12         if((bookings[i].Am) && (!bookings[i].Pm)){
13             time = "Before lunch"
14         } else if ((!bookings[i].Am) && (bookings[i].Pm)) {
15             time = "After lunch"
16         } else {
17             time = "The whole day"
18         }
19
20         // Finds the correct day based on the dayNumber.
21         switch(dayNumber){
22             case 1: day = 'Monday';    break;
23             ...
24             case 0: day = 'Sunday';    break;
25         }
26
27         // Switch name of the floor.
28         switch(bookings[i].Floor) {
29             case '16': floor = 'Floor 16';    break;

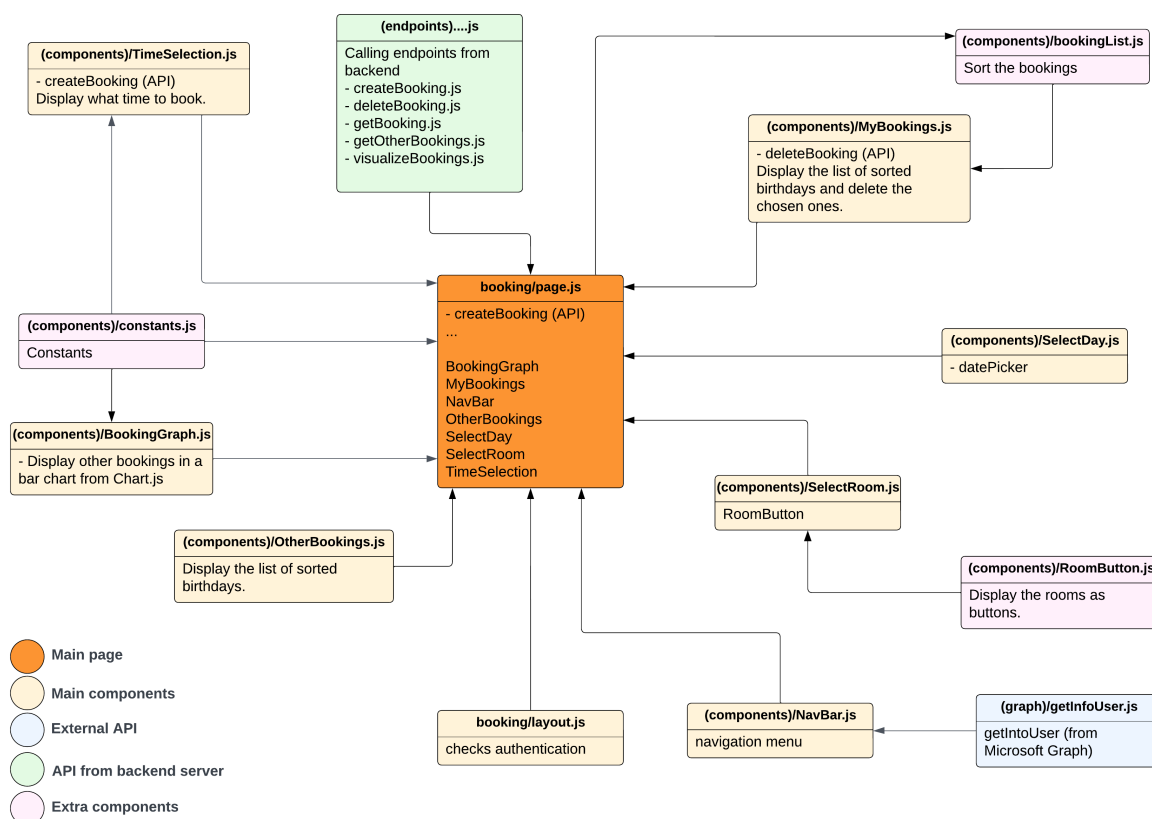
```

```
30         case '17': floor = 'Fellesareal'; break;
31         case '18': floor = 'Glassburet'; break;
32     }
33
34     Booking = {
35         Floor: floor,
36         Time: time,
37         Date: bookings[i].Date.split(" ")[0],
38         Day: day,
39         Id: bookings[i].Id
40     }
41     bookingsList.push(Booking)
42 }
43 return(bookingsList)
44 }
```

6.5 Bookingsiden

Figur 6.13 viser klassediagrammet til bookingsiden. Videre presenteres sammenhengen mellom alle filene.

- I mappen (endpoints) finner vi API-er som kaller på backenden. Disse endepunktene blir kalt fra hovedsiden fra filen *booking/page.js*.
- Fra hovedsiden sendes resultatet av kallet på API-et *getBooking* til bookingsList.js der de sorteres. Videre blir de presentert visuelt i filen MyBooking.js. I den samme filen, finner vi også muligheter for å slette reservasjoner. Da kalles API-et *deleteBooking*.
- SelectDay.js håndterer komponenten datovelger.
- RoomButton.js er en fil som presenterer etasjene som knapper. Disse blir videre brukt i SelectRoom.js som holder styr på hvilken etasje som er valgt.
- I filen getInfoUser.js kaller vi på Microsoft Graph og henter informasjon om brukeren. Dette blir videre presentert i navigasjonsbaren.
- Filen layout.js passer på at brukeren hele tiden er autentisert gjennom Evidi.
- Hovedsiden sender resultatet fra API-et *getOtherBookings* som filen OtherBookings.js presenterer visuelt.
- BookingsGraph.js lager en graf basert på resultatet fra API-et *visualizeBookings* over hvor mange som har reservert på en bestemt dato.
- I filen TimeSelection.js lages det nye reservasjoner og kaller på *createBooking* for å legge de til i databasen.
- constants.js er en fil som inneholder konstanter som blir brukt i programmet.



Figur 6.13: Klassediagram til bookingsiden (laget i lucid.app).

6.5.1 Slette egne bookinger

Denne komponenten er basert på komponentet Mine bookinger fra hjemmesiden og bruker også samme sorteringsfunksjon *BookingList* beskrevet i 6.4.3. Forskjellen på disse to komponentene, er at i dette tilfellet blir bookingene listet i en sjekkliste for å markere hvilke bookinger brukeren ønsker å slette. Se figur 6.14.

Knappen DELETE er deaktivert når ingen av bookingene i listen er sjekket av. Når en eller flere er sjekket av, blir den aktivert. Når knappen DELETE trykkes, blir en liste med ID-er fra de valgte bookingene sendt til funksjonen *deleteBooking*. Se kodelisting 32. Her kalles API-et som sletter bookinger fra databasen.



Figur 6.14: Slette bookinger.

Listing 32: deleteBooking.js: kaller på API-et for sletting av reserverasjoner.

```

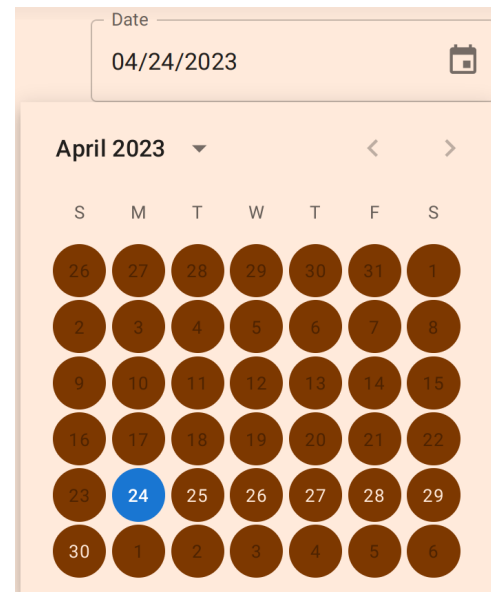
1  export async function deleteBooking(ids){
2      const options = {
3          revalidate: 0,
4          method: 'DELETE',
5          headers: {
6              'Accept': 'application/json',
7              'Content-Type': 'application/json',
8              'Origin': '',
9              'x-functions-key': process.env.REACT_APP_FUNCTION_HOST_KEY
10         },
11         body: JSON.parse(JSON.stringify(`[${ids}]`))
12     }
13
14     const res = await fetch(REMOVE_DESK_BY_ID, options);
15     if(!res.ok){
16         alert("Failed to delete booking");
17         throw new Error("Failed to delete booking: " + res.statusText)
18     }else{
19         return res.json();
20     }
21 }

```

6.5.2 Dato

Komponenten er bygd opp ved hjelp av *MUI DatePicker* som er beskrevet i seksjon 4.2.5. Datovelgeren gjør det mulig å velge en dato i nåværende uke, men det er ikke mulig å gå tilbake i tid, som vist i figur 6.15.

Videre ønsker vi å bruke datoen valgt fra datovelgeren og må derfor standardisere formatet. Grunnen til at vi standardiserer formatet, er at det skal være mulig å bytte datoformat på serveren som webapplikasjonen kjører på, uten at man trenger å endre koden. Dette gjøres ved hjelp av funksjonen *formatDate* med dato-parameter fra datovelgeren, som vist i kodelisting 33.



Figur 6.15: Datovelger.

Listing 33: SelectDay.js: formater datoen.

```

1 //The date from DatePicker
2 const [selectedDate, setSelectedDate] = useState(new Date());
3 const router = useRouter();
4
5 /**
6  * Format the date to MM/DD/YYYY with date as a parameter.
7  * @param date from the DatePicker
8  */
9 function formatDate(date) {
10   let month = date.getMonth() + 1;
11   let day = date.getDate();
12   let year = date.getFullYear();
13   // Format MM/DD/YYYY.
14   let newDay = `${month.toString().padStart(2, '0')}/`
15   `${day.toString().padStart(2, '0')}/`
16   `${year.toString()}`
17   // Set the selected date as a cookie.
18   cookieCutter.set('selectedDate', newDay)
19   router.refresh();
20 }
21
22 // When selectedDate changes,
23 // it calls the function formatDate(date) with selectedDate as the parameter.
24 useEffect(() => {
25   formatDate(selectedDate);
26 }, [selectedDate]);

```

Siste steg er å implementere datovelgeren i programmet, som vist i kodelisting 34. Ved å bruke *LocalizationProvider* sammen med *AdapterDayjs* forsikrer vi oss at datovelgeren støtter tids- og datoformatet til serveren som webapplikasjonen kjører på. I datovelgeren setter vi inputformatet til MM/DD/YYYY og passer på at *selectedDate* blir oppdatert når brukeren trykker

på en ny dato. For å forsikre oss om at brukeren ikke kan gå tilbake i tid, settes *disablePast* til sant, samtidig som maksdatoen settes fra funksjonen *findMaxDay*. Denne maksdatoen vil være søndagen i nåværende uke.

Listing 34: SelectDay.js: implementering av datovelgeren.

```

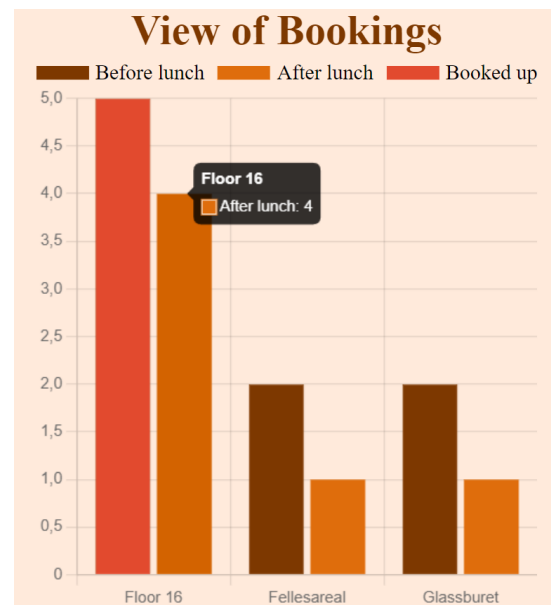
1 return(
2   <div className = {styles.BookingDay}>
3     <LocalizationProvider dateAdapter={AdapterDayjs}>
4       <DatePicker
5         label="Date"
6         inputFormat="DD/MM/YYYY"
7         value={selectedDate}
8         onChange={(newValue) => {
9           setSelectedDate(newValue.$d);
10        }}
11        // Handles input.
12        renderInput={(params) => <TextField {...params} />}
13        showDaysOutsideCurrentMonth={true}
14        //Min date today
15        disablePast={true}
16        maxDate={maxDate}
17        //Change layout
18        PopperProps={{
19          sx: popperSx
20        }}
21      />
22    </LocalizationProvider>
23  </div>
24 );

```

6.5.3 Visuell presentasjon av bookinger

I stolpediagrammet ønsker vi å hente ut alle reserverasjonene fra en bestemt dato. Som grafen i figur 6.16 viser, er det ikke mulig å booke «Før lunsj» i 16.etasje. Reserverasjonene hentes fra databasen og sendes videre inn i grafen. Reserverasjonene blir videre sortert på tidspunkt og etasje.

Kodelisting 35 viser hvordan det lages to datasett fra de sorterte listene; *Før lunsj* og *Etter lunsj* som skal brukes i stolpediagrammet. Hver sone med plasser har en maksverdi for antall sitteplasser. Ved å sammenligne lengden på listen med denne maksverdien, settes fargen til brun/oransje eller rød avhengig av lengden.



Figur 6.16: Visuell presentasjon av bookinger.

Listing 35: BookingGraph.js: lager datasett basert på de sorterte listene.

```
1  const data = {
2    // Name of the variables on x-axies for each bar.
3    labels: ["Floor 16", "Fellesareal", "Glassburet"],
4    datasets: [
5      {
6        label: "Before lunch",
7        data: [before16.length, before17.length, before18.length],
8        // Color of the bars.
9        backgroundColor: [
10         before16.length >= floor16MaxDesks ? "#e24a2e" : "#7d3800",
11         before17.length >= floor17MaxDesks ? "#e24a2e" : "#7d3800",
12         before18.length >= glassburetMaxDesks ? "#e24a2e" : "#7d3800",
13       ],
14       borderColor: ['#7d3800', '#7d3800', '#7d3800'],
15       borderWidth: 0.5,
16     },
17     {
18       label: "After lunch",
19       data: [after16.length, after17.length, after18.length],
20       // Color of the bars.
21       backgroundColor: [
22         after16.length >= floor16MaxDesks ? "#e24a2e" : "#df6d0c",
23       ...
```

Til slutt implementeres stolpediagrammet ved hjelp av biblioteket Chart.js som er beskrevet i seksjon 4.2.6. Datasettene presenteres i stolpediagrammet som vist i linje 4 i kodelisting 36.

Listing 36: BookingGraph.js: implementering av stolpediagrammet ved hjelp av Graph.js.

```

1  return (
2      ...
3      <ReactChartJs type="bar"
4          data={data}
5          options={{
6              // Auto scale to fit the data.
7              responsive: true,
8              // No defined size of the diagram.
9              maintainAspectRatio: false,
10             plugins: {
11                 legend: {
12                     display: false,
13                 },
14             },
15             scales: {
16                 y: {
17                     ticks: {
18                         beginAtZero: true,
19                     },
20                 },

```

6.5.4 Se andre sine bookinger

The screenshot shows a booking interface with three main sections:

- Book a Seat:** A vertical list of three buttons: "Floor 16", "17: Fellesareal", and "17: Glassburet".
- Time of the Day:** Three radio buttons: "Before lunch", "After lunch", and "The whole day". Below them is a "Book" button.
- Colleague Reservations:** A table showing reservations for "The whole day", "Before lunch", and "After lunch".

The whole day	Before lunch	After lunch
Sigrid Warberg Vestheim	Henriette Eltvik	Alf Magnus Nohr Jorgen Mobakken

Figur 6.17: Andre kollegaer sine reservasjoner.

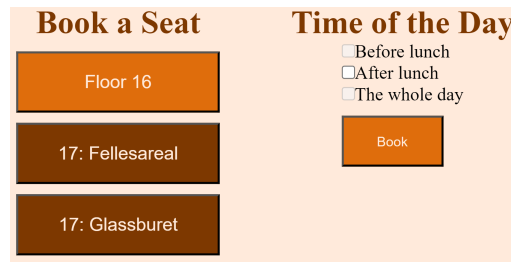
Fra databasen henter vi ut reservasjonene for en bestemt dato og etasje. Disse reservasjonene lagres i en liste og sendes inn i komponenten sammen med den valgte datoen og etasjen. Videre sorteres bookingene basert på tidspunkt og lagres i ulike lister. For å se komponenten, er man nødt til å velge en etasje.

6.5.5 Reserver en plass

Når brukeren ønsker å reservere en plass, må man først velge dato fra datovelgeren (beskrevet i seksjon 6.5.1), etasje og tidspunkt. Når man trykker på en av de tre knappene til venstre i figur 6.18, settes *selectedFloor* til ID-en til denne knappen.

Når en av knappene fra figur 6.18 er valgt, vil man få opp muligheten for å velge tidspunkt. *TimeSelection* er komponenten *Time of the Day* og inneholder flere funksjoner. Denne komponenten passer på at bare ett tidspunkt er valgt om gangen ved å bruke funksjonen *handleCheck*. En annen funksjon i komponenten, er *shouldDisableButton* som deaktiverer tidspunkt hvis det er fullt på dette tidspunktet.

For å opprette en reservasjon, må brukeren velge en etasje, et tidspunkt og en dato. Når knappen BOOK blir aktivert, sendes en POST-forespørsel til backend med informasjon om etasje, dato og tidspunkt. Dette vises i kodelisting 37.



Figur 6.18: Visuell presentasjon av booking.

Listing 37: createBooking.js: lager en reservasjon.

```

1  export async function createBooking(id, floor, date, am, pm){
2      const options = {
3          cache: 'no-store',
4          method: 'POST',
5          headers: {
6              'Accept': 'application/json',
7              'Content-Type': 'application/json',
8              'Origin': '',
9              'x-functions-key': process.env.REACT_APP_FUNCTION_HOST_KEY
10         },
11         body: JSON.stringify({
12             'u_id': id,
13             'floor': floor,
14             'date': date,
15             'am': am,
16             'pm': pm,
17         })
18     }
19
20     const res = await fetch(CREATE_BOOKING, options)
21     if(!res.ok){
22         // Check error type.
23         switch (res.status){
24             case 400: alert("You have already reserved a desk for today! "); break;
25         }
26         throw new Error("Failed to create booking: " + res.statusText)
27     }else{
28         return res.json();
29     }
30 }

```

6.6 Kommunikasjon mellom frontend og backend

Kommunikasjonen mellom frontend og API-et i backend skjer ved at frontenden sender HTTP-kall til API-ets endepunkter. Endepunktene har blitt navngitt basert på Microsoft sitt anbefalte API-design [61]. For vår del betyr det at endepunktene ble designet rundt de spesifikke ressursene *Employee*, *Booking* og *Birthday* som vist i figur 6.19.

Birthday	
GET	/birthday Get all of the birthdays for a period
Booking	
POST	/booking Create a new booking
DELETE	/booking Delete booking based on the id of the booking
GET	/booking Get all the bookings from others in the office
GET	/booking/visualized-booking Get others bookings in a visualize-enabled format
Employee	
GET	/employee/booking Get the employee's bookings
POST	/employee Create a new employee
PUT	/employee/birthday Update user birthday

Figur 6.19: API-endepunkter (fra swagger).

Som forklart i delkapittel 6.1.3, er responsen vi får tilbake fra API-et også i JSON-format. Det gjør at vi enkelt kan konvertere responsen til et JavaScript-objekt før informasjonen tas i bruk. Når det kommer til endepunktene som håndterer POST, PUT og DELETE, er det nødvendig å sende med informasjon i en *body*. Denne informasjonen må være i JSON, samt inneholde nødvendig data for å utføre kallet. Kodelisting 38 viser hva som er nødvendig for POST-kallet til booking.

Listing 38: Lag en ny booking JSON-*body*.

```

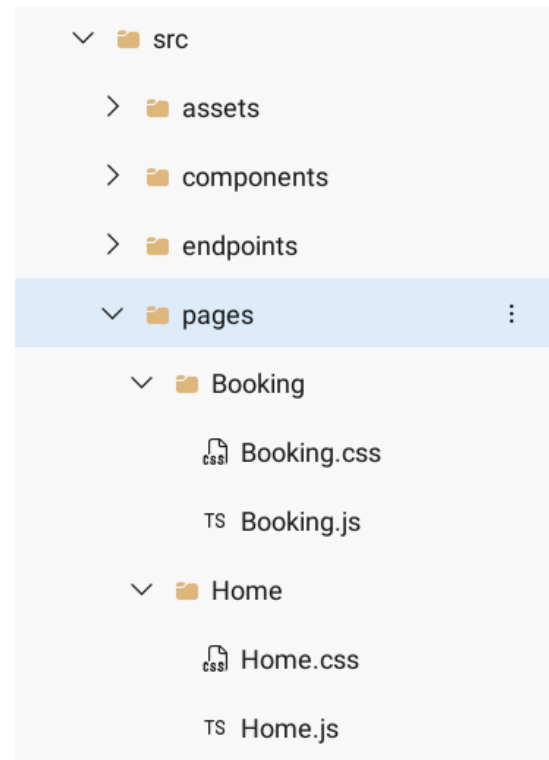
1 {
2   "u_id": "0b933fc9-3391...",
3   "floor": 16,
4   "date": "5/5/2023",
5   "am": true,
6   "pm": true
7 }
```

6.7 Overgang til Next.js

Med bruken av API-tokens som forklart i 4.6.5, dukket det opp utfordringer med React og henting av miljøvariabler. Vi diskuterer problemene i kapittel 10, men viser til de nødvendige endringene som ble gjort i denne seksjonen.

6.7.1 Refaktorering

Når vi først utviklet applikasjonen i React, måtte det brukes en *pages*-mappe for å definere de forskjellige hovedsidene til nettsiden. I motsetning til React, bruker Next.js en app-mappe og filen *page.js* for å definere hovedsidene, samt endepunktene til siden. Dette er forklart i 4.2.2. Dette medførte en større refaktorering av prosjektet der vi kvittet oss med *pages*-mappen og endte opp med filstrukturen som vist i 5.3.3.



Figur 6.20: Bruk av *pages*.

6.7.2 Bruk av miljøvariabel

Før overgangen til Next.js, var alle komponentene på klientsiden. Det gjorde at vi kalte på API-et direkte fra klient-komponentene som vist i kodeliste 39. Problemet med dette var at det ikke ville være mulig å inkludere miljøvariabelen med tilgangstokenet som vist på linje 5 i kodeliste 40, da denne lå på webserveren til applikasjonen. Det hadde imidlertid vært mulig å sette denne miljøvariabelen hos klienten, men da hadde poenget med å ikke eksponere tilgangstokenet vært borte.

Listing 39: MyBooking.js i React.

```

1 export default function BookingList() {
2   ...
3   const fetchData = async () => {
4     ...
5     // Call backend API.
6     const res = await getMyBooking(userId)
7   }
8   ...
9 }

```

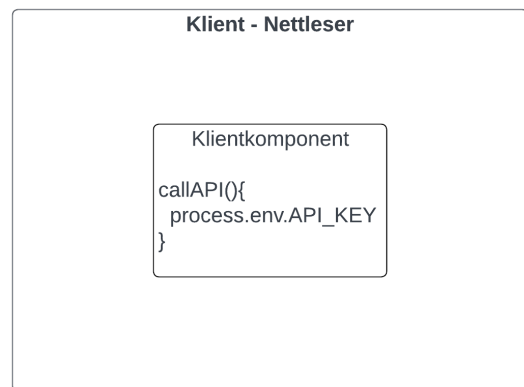
Listing 40: getMyBooking.

```

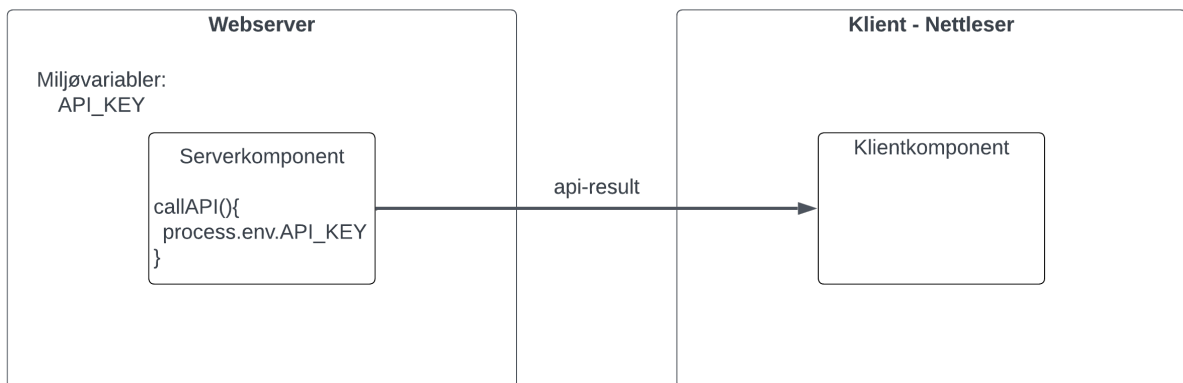
1  const options = {
2    ...
3    headers: {
4      ...
5      'x-functions-key': process.env.REACT_APP_FUNCTION_HOST_KEY
6    }
7  }
8  export async function getMyBooking(id){
9    ...
10   const res = await fetch(url, options);
11   ...
12  }

```

For å visualisere problemet for bruken av serverkomponenter, ser vi i figur 6.21 at alt blir håndtert hos klienten. Klientkomponenten kan ikke bruke miljøvariabelen `API_KEY` fra webserveren som visualisert i figur 6.22. Ved å ta i bruk serverkomponenter kan vi flytte funksjonene som håndterer API-kall slik at de kan hente miljøvariabler som ligger hos webserveren. På denne måten kan resultatet fra API-et sendes til klientkomponentet for videre bruk.



Figur 6.21: Før Next.js.



Figur 6.22: Etter Next.js.

Ved å bruke server-komponenter løste vi problemene med miljøvariablene, samt at vi fulgte Next.js sin anbefalte metode for henting og sending av data [62].

6.7.3 Server-komponent

Som forklart i seksjonen over, måtte vi flytte API-kall ut i fra klient-komponentene. Sammenligner vi eksemplet fra kodelisting 39 med kodelisting 41, ser vi at `getMyBooking` nå har blitt flyttet inn i hovedkomponenten `Home`. `Home` er forelder-komponenten til hjemmesiden og er definert som en server-komponent.

For å følge Next.js sin ytelses-anbefaling, plasserer vi klient-komponenter i bladene til komponent-treet[63]. Dette ser vi i kodelisting 41 hvor `HomeBooking` ligger under server-komponenten `Home`. Her kaller vi på API-et før resultatet videre blir sendt til klient-komponenten `HomeBooking` som vist på linje 9 i kodelisting 41.

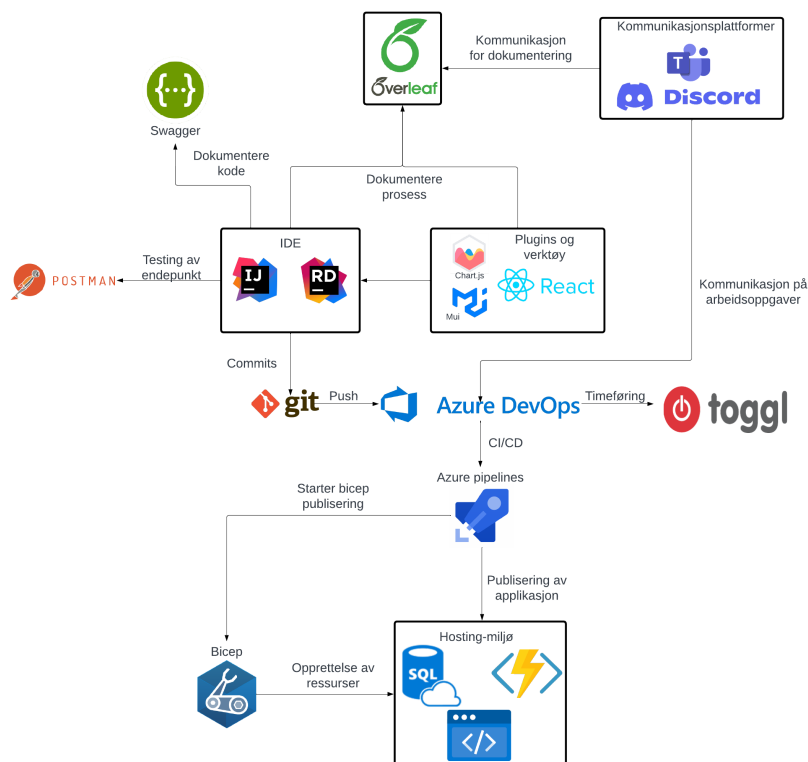
Listing 41: Home-serverkomponent.

```
1  const Home = async () => {
2      ...
3      // Get my bookings.
4      let myBookings = await getMyBooking(userID.value);
5      ...
6
7      return (
8          ...
9          <HomeBooking myBookings={myBookings}/>
10         ...
11     );
12 }
```

Kapittel 7

Utviklingsmiljøer og prosjektstyringsverktøy

I dette kapittelet skal vi se på hvilke utviklingsmiljøer og prosjektstyringsverktøy vi har brukt for å løse oppgaven. Siden vi hadde begrenset med tid, var det viktig å velge miljøer og verktøy som gjorde arbeidet så effektivt som mulig. Fra oppgavebeskrivelsen hadde Evidi som krav at vi skulle bruke Azure DevOps til planlegging, kildekodekontroll og automatisert bygg og release. I tillegg skulle vi lære å jobbe med ressurser i skytjenester ved hjelp av Microsoft Azure. Ved å bruke verktøyene fra Microsoft Azure, reduserte det antall verktøy som vi måtte forholde oss til. Figur 7.1 viser alle verktøyene vi brukte og hvor i prosessen de ble brukt. I kapittel 10 diskuteres det hvilke erfaringer dette ga oss.



Figur 7.1: Utviklingsmiljø og prosjektstyringsverktøy (laget i lucid.app).

7.1 Utviklingsmiljøer

7.1.1 Frontend

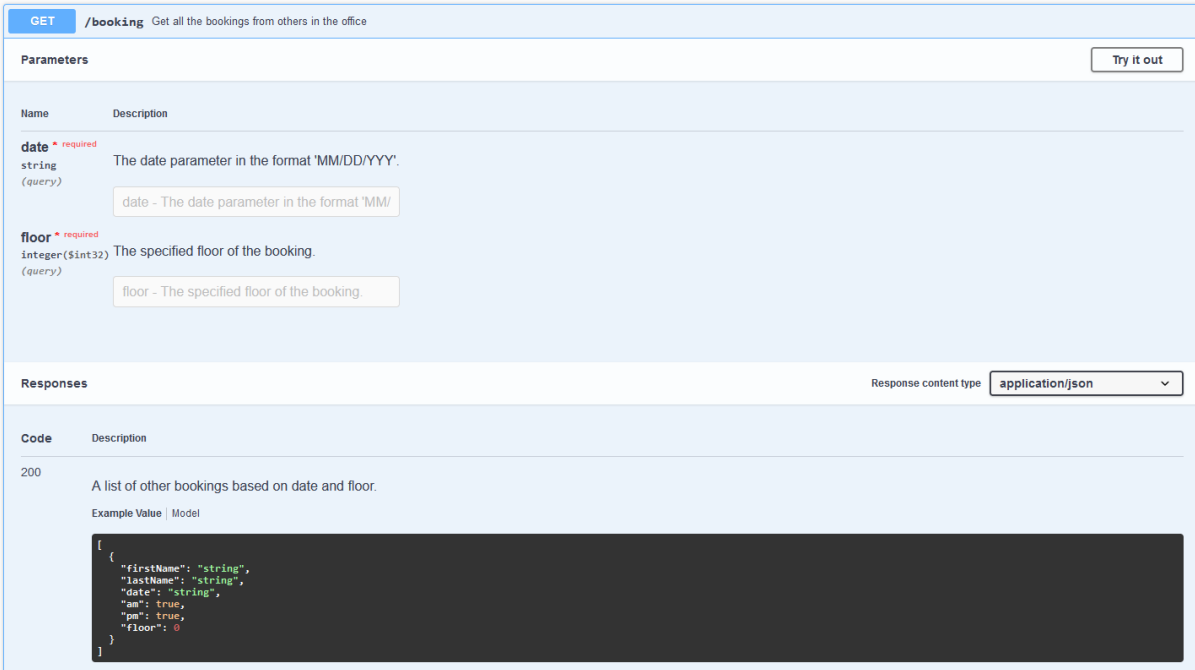
IntelliJ er et av JetBrains sine Integrated Development Environment (IDE) som er kompatibelt med ulike JVM-språk og forskjellige programmeringsspråk. For eksempel tilbyr IDE-en støtte til React og JavaScript. Dette var en av grunnene for at vi brukte IntelliJ som IDE. Andre grunner til at vi tok det i bruk, var at den kan kjøres på flere operativsystemer og den viser klare feilmeldinger [64]. Programmet inkluderer også en debuggingsfunksjonalitet som gjør det enkelt å finne problemer og fikse de før applikasjonen er publisert.

Forskjellige biblioteker er inkludert under utviklingsprosessen av frontend-applikasjonen. Noen av disse er *chart.js* og MUI. Bruken av bibliotekene resulterte i at vi kunne fokusere mer på den logiske kodingen av applikasjonen istedenfor designet. React er et JavaScript-bibliotek som gruppen aktivt har tatt i bruk for å effektivisere kodingen av programmet. Implementering av bibliotekene ligger under kapittel 6 og teorien ligger under kapittel 4.

7.1.2 Backend

Rider er en IDE som er kompatibelt sammen med .NET. Den er basert på IntelliJ-plattformen og ReSharper, som er en utvidelse av *visual studio code* [65]. Rider støtter mange .NET-prosjekttyper som ASP.NET- og ASP.NET Core-webapplikasjoner [66].

For at programmet skal være leselig for andre utviklere og det skal være enkelt å ta i bruk endepunktene, er Swagger brukt som dokumentasjon. Swagger gjør det enkelt å få en oversikt over implementerte endepunkt i backend og den inkluderer muligheten for å teste endepunktene gjennom en brukervennlig GUI [67]. Figur 7.2 viser hvordan «GetMyBooking»-endepunktet er vist gjennom Swagger.



The screenshot displays the Swagger UI for the endpoint `GET /booking`. The description is "Get all the bookings from others in the office".

Parameters:

Name	Description
date * required string (query)	The date parameter in the format 'MM/DD/YYYY'.
floor * required integer(\$int32) (query)	The specified floor of the booking.

Responses: Response content type: `application/json`

Code 200: A list of other bookings based on date and floor.

Example Value | Model

```
[
  {
    "firstName": "string",
    "lastName": "string",
    "date": "string",
    "am": true,
    "pm": true,
    "floor": 0
  }
]
```

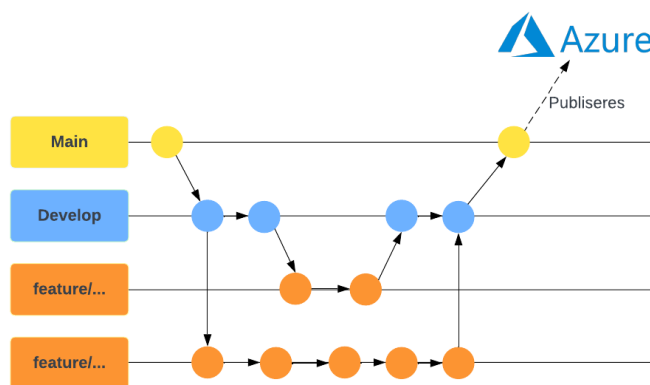
Figur 7.2: Endepunkter i Swagger.

7.1.3 Git

For å ha kontroll på utviklingen av funksjonalitet og passe på at ny funksjonalitet ikke ødela for gammel, brukte vi en grenstruktur som vist i figur 7.3. Gruppen brukte tre hovedgrener for utviklingen; *main*, *develop* og *feature*.

Develop-grenen inneholder koden som er under utvikling. Fra denne grenen lagde gruppelemmene *feature*-grener. Disse grenene ble brukt for å utvikle nye funksjonaliteter eller rette feil i programmet. Navnene til disse grenene skulle inneholde informasjon om hvilken funksjonalitet som ble utviklet.

Når ny funksjonalitet til slutt er ferdigstilt og det er klart for produksjon, blir det laget en *pull request* inn i *develop* før den videre blir *merget* til *main*.



Figur 7.3: Oppbevaringssted/repo grener (laget i lucid.app).

7.1.4 Azure DevOps

Azure DevOps gjør det enkelt for utviklere, prosjektledere og andre bidragsytere å utvikle et produkt sammen. Plattformen inneholder verktøy for å effektivisere utviklingsprosessen av produkter [68]. Azure DevOps har vært et nyttig hjelpemiddel under utviklingen og under er det en beskrivelse av verktøyene som er brukt.



Overview - inneholder en oversikt over prosjektet. I tillegg finner man wiki-siden. I prosjektet har vi brukt dette til å lage Git-standarder og brukerguide for oppsett av prosjektet.



Boards - viser en oversikt over backlog, sprint, retrospektiv og kanbanbrettet. Dette gjorde det enklere for gruppen å holde oversikt over hvor mange oppgaver som var fullført i løpet en sprint. Hvordan funksjonene er brukt, er beskrevet i kapittel 2.



Repos - her finner vi all informasjon om *repositoriet*. Denne informasjonen er blant annet hvilke grener som finnes, filer, *commits*, *pushes* og *pull request*. Dette gjorde det enkelt å få oversikt over repoet. For å sende kildekoden til repoet, brukte vi Git både for å sende og hente kode mellom eksternt og lokalt repo. En av de viktigste funksjonene vi brukte, var *pull request*, som er beskrevet i kapittel 2.



Pipelines - viser en oversikt og muligheten til å konfigurere pipelines for prosjektet. Ved bruk av pipelines kunne vi teste funksjonalitet fra skyen underveis i utviklingen ved å sette opp CI/CD pipelines for publisering av koden til Azure. Hvordan dette ble satt opp presenteres i kapittel 9.

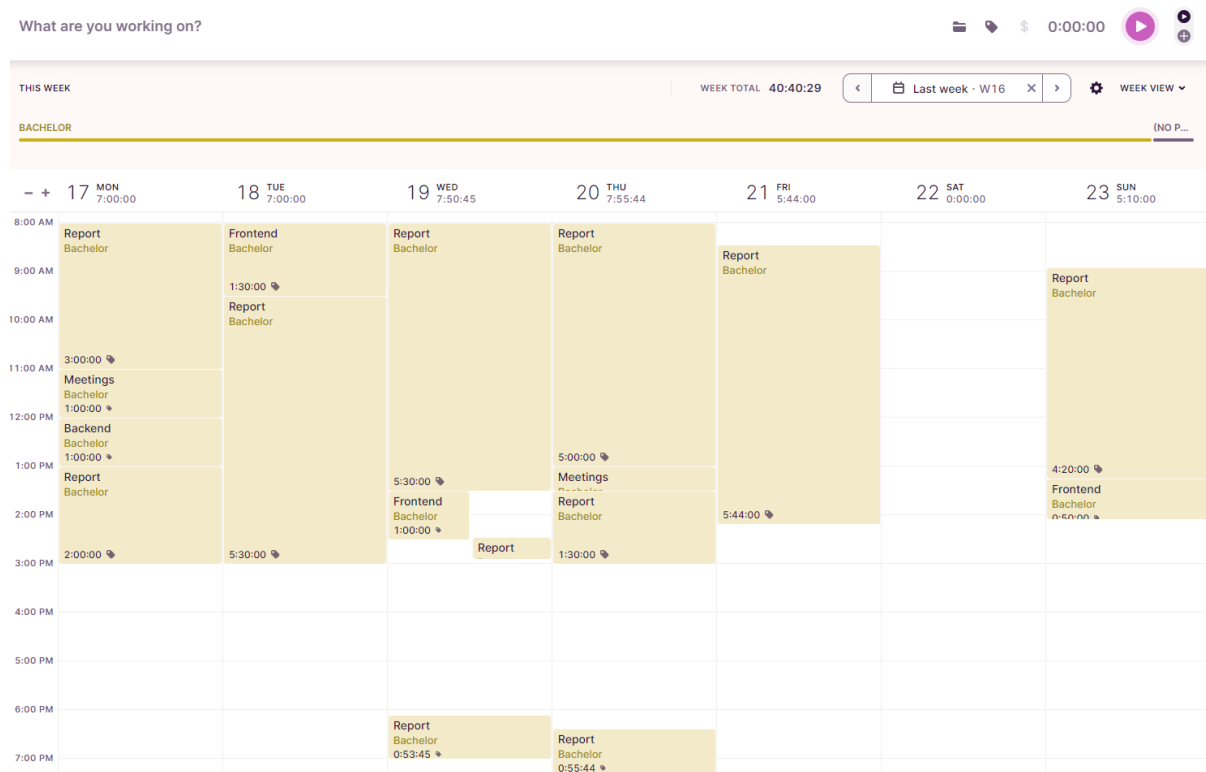
7.2 Prosjektstyringsverktøy

7.2.1 Timeføring

For å holde styr på hvor mange timer hvert enkelt medlem har brukt i løpet av prosessen, tok vi i bruk verktøyet *Toggl*. Dette ga oss mulighet til å loggføre alle timer. Etter hver arbeidsdag loggførte vi timene, slik at vi ikke skulle glemme hva vi hadde gjort. Vi laget forskjellige kategorier for å sortere timene. Disse kategoriene var:

- *frontend* - koding i frontend
- *backend* - koding i backend
- *meetings* - møte med veileder fra NTNU eller Evidi
- *report* - jobbe med rapporten
- *research* - lese seg opp på teknologier
- *infrastructure* - jobbe med infrastrukturen til prosjektet

Figur 7.4 viser et eksempel på hvordan ett medlem loggførte timer i uke 16. Vedlegg C presenterer loggførte timer for alle gruppe-medlemmene fra og med uke 2 til og med uke 20.



Figur 7.4: Skjerm bilde fra track.toggl.com.

7.2.2 Kommunikasjonsplattformer

Teams er brukt som en kommunikasjonskanal mellom gruppemedlemmene og Evidi. Plattformen ble også brukt til sending av teoretisk informasjon og nyttige lenker til utviklingen av prosjektet. Teams som en kommunikasjonskanal gjorde det spesielt enkelt å komme i kontakt med veilederne våre fra Evidi.

Discord ble brukt som hovedkommunikasjonsplattform mellom gruppemedlemmene. Kommunikasjonsplattformen ble brukt for å sende oppdateringer på oppgaver og for å komme raskt i kontakt med andre gruppemedlemmer. Det ble for det meste brukt når gruppemedlemmene ikke kunne arbeide på skolen. I Discord opprettet vi ulike kanaler for å sortere meldingene. Dette gjorde det oversiktlig og enkelt å finne frem til eldre meldinger. De opprettede kanalene var delt inn i meldinger som gjelder rapporten, meldinger som inkluderte viktige linker og generelle meldinger for å kontakte gruppemedlemmene. En kanal for tidsfrister som gjaldt tidsfrister for rapporten ble også opprettet.

Kapittel 8

Kvalitetssikring

Gjennom hele prosjektet har vi hatt fokus på kvalitet. For å sikre en god kvalitet på det ferdige produktet, har vi tatt i bruk ulike verktøy.

8.1 Kodekvalitet

For å forsikre at programmet er effektivt og vedlikeholdbart, har det vært nødvendig å følge visse standarder for hvordan vi ønsket å skrive kode. Vi valgte tidlig i utviklingsfasen å følge Microsoft sin *C# Coding Conventions* slik at koden ble konsistent skrevet, da vi som utviklere har forskjellige preferanser for å skrive kode [69]. Vi fulgte også standarder i React der vi blant annet skrev komponent-navnene med stor forbokstav og hjelpefiler med liten bokstav [70].

Det ble tatt hensyn til at koden måtte være modulær for å unngå duplikasjon, men også for å gjøre det lettere å feilsøke, samt bruke om igjen enkelte deler. God kodekvalitet hjalp også til med effektiviteten, da det sikret at koden var leselig, forståelig og vedlikeholdbar.

8.1.1 Dokumentasjon

For videre utvikling av programmet er det nødvendig at all kode er dokumentert. I tillegg til kommentering i selve koden, har vi benyttet oss av Swagger for dokumentering av API i backend. Hvordan dette ble implementert ser vi i kodelisting 42. De I frontend har vi brukt JSDoc-kommentarer som gjør det mulig å generere dokumentasjonssider for applikasjonen og dens funksjonalitet som vist i figur 8.1. Videre fra denne dokumentasjonen kan man finne en forhåndsvisning av koden som er dokumentert. Dette er vist i figur 8.2.

Listing 42: Swagger-dokumentering i Azure Function.

```
1 [ApiOperation(
2   operationId: "CreateBooking",
3   tags: new[] { "Booking" },
4   Summary = "Create a new booking"
5 )]
6 [ApiOperationRequestBody(
7   "application/json",
8   typeof(Reservation),
9   Description = "The request body for a new booking, containing userID," +
10                " floor, date and the time of the booking."
11 )]
12 [ApiResponseWithBody(
13   statusCode: HttpStatusCode.Created,
14   contentType: "application/json",
15   bodyType: typeof(Reservation),
16   Description = "The CREATED response"
17 )]
```

Methods

(*async*) HomeBooking() → {*JSX.Element*}

Displays the bookings of the user

Source: [HomeBooking.js, line 14](#)

Returns:

Returns the booking of the current user

Type

JSX.Element

Documentation generated by *JSDoc 3.6.11* on Thu May 18 2023 17:23:57 GMT+0200 (Central European Summer Time)

Figur 8.1: JSDoc HomeBooking.

Source: HomeBooking.js

```

1.  // 'use client';
2.
3.  import styles from "../page.module.css"
4.  import BookingList from "../../(components)/bookingList";
5.  import {getMyBooking} from "@app/(endpoints)/getBooking";
6.  import Login from "../../(components)/AuthenticateEvidi"
7.  import NoSSRWrapper from "@app/(components)/NoSSRWrapper";
8.
9.  /**
10.   * Displays the bookings of the user
11.   * @returns {JSX.Element} Returns the booking of the current user
12.   */
13.
14.  const HomeBooking = async ({myBookings}) => {
15.
16.    //List with all of my bookings
17.    const listBooking = BookingList(myBookings)
18.
19.    return (
20.      <div className={styles.HomeBooking}>
21.        <h1>My Bookings</h1>
22.        <ul style={{marginLeft: 20}}>
23.          {listBooking.map((item, index) => (
24.            <li key={index} style={{fontSize:18}}>
25.              {item.Day}, {item.Date}, {item.Floor}, {item.Time}
26.            </li>
27.          ))}
28.        </ul>
29.      </div>
30.    )
31.  }
32.
33.  export default HomeBooking

```

Documentation generated by *JSDoc 3.6.11* on Thu May 18 2023 17:23:57 GMT+0200 (Central European Summer Time)

Figur 8.2: JSDoc forhåndsvising kode.

8.2 Code Review

En av måtene vi sikret kvalitet på, var å se over koden som de andre gruppemedlemmene hadde utviklet. Når et av gruppemedlemmene var ferdig med utvikling i en sidegren, laget den en *Pull request*. Videre måtte de resterende medlemmene i gruppen se over koden. Her var det muligheter for å kommentere positive eller negative ting med koden. Figur 8.3 viser et eksempel på hvordan en *pull request* kan se ut og hvordan de andre gruppemedlemmene kan komme med kommentarer. Her er det lagt inn en kommentar om forbedringer før *requesten* godkjennes.

For å sikre kvalitet, var det viktig å se om standardene beskrevet i seksjon 8.1 var brukt. Ved å se over andre sine kodesnutter, kunne man se og lære hvordan standarden ble brukt i praksis. I tillegg til dette, oppstod det ofte *merge*-konflikter. Denne måtte løses før vi kunne *merge* sidegrenen inn i hovedgrenen.

The screenshot shows a GitHub pull request interface. At the top, the title is "test: add test for creating employee" with a blue "Approve" button and a "Complete" button. Below the title, it says "Active" and "12720" commits, and "Henriette Eltvik feature/test-employee-birthday into develop 0/1 comments resolved". The interface is divided into several sections: "Overview" (selected), "Files", "Updates", "Commits", and "Conflicts". A notification bar says "New changes were pushed." with a "View latest" button. A green checkmark indicates "No merge conflicts" (last checked 3m ago). The "Description" section contains the text "feat: add test for creating employee". The "Reviewers" section shows two optional reviewers: "Alf Magnus Nohr" (Approved) and "sigridwv" (Approved). The "Tags" section is empty. The "Work items" section shows a task "Task 11191: Backend: test create em..." (Updated Yesterday, Active). The main content area shows a list of activity: "Add a comment..." (input field), "sigridwv approved the pull request" (2m ago), "Henriette Eltvik pushed 1 commit" (3m ago) with commit details "doc: add some comments for create employee" (1b9b086, Today at 14:06), "sigridwv" (8m ago) with a comment "Add some comments." and a "Resolve" button, "Alf Magnus Nohr approved the pull request" (5h ago), and "Henriette Eltvik created the pull request" (Yesterday).

Figur 8.3: Code Review for *pull request*.

8.2.1 Parkoding

I prosjektet møtte vi på flere problemer som krevde mye ressurser. For å løse dette på en god måte, tok vi i bruk parkoding. Vi har benyttet oss av dette både i utviklingen av frontend og backend. Da kunne to gruppe-medlemmer sitte med det samme problemet og diskutere mulige løsninger. Dette har vært et godt hjelpemiddel for gruppen, der vi har reflektert og diskutert problemer og løsninger. Det har bidratt til en større forståelse for problemløsning og et bedre samarbeid innad i gruppen.

8.3 Testing

I starten var det tenkt at programmet skulle inkludere enhetstester. Etter at integrasjonen mellom databasen og backend var ferdig implementert, ble det mindre gunstig å inkludere enhetstestene. Grunnen var at de logiske kodesnuttene ble erstattet med klasser som inneholdt kode med SQL-uttrykk. Med andre ord så inneholder ikke det endelige programmet avanserte operasjoner, bare interaksjoner mellom databasen og backend.

Sammenlignet med enhetstester som bare tester individuelle logiske deler av programmet, vil integrasjonstestene teste hele programmet som en enhet. Det ble også enklere å teste selve funksjonene i programmet. Det trengs mer enn automatiske tester for å teste funksjonaliteten til applikasjonen som en helhet. Dermed ble det gjennomført en brukertesting nærmere slutten av prosjektet. Brukertesting ga viktig informasjon som kunne bli brukt av de som skulle ta over utviklingen av programmet senere.

8.3.1 Integrasjonstesting

Integrasjonstesting gjorde det enkelt å teste selve funksjonaliteten til backenden og kommunikasjonen mellom databasen og backenden. For å sjekke interaksjonen mellom databasen og backend, sjekker vi om databasen sender de riktige verdiene basert på funksjonaliteten vi tester. Kodelisting 43 viser hvordan vi har testet tilkoblingen mellom API-et og databasen. Vi tester også Azure Function ved å se om funksjonene returnerer riktig statuskode og respons når en bruker kaller på dem. Testene sjekker også om responsen er sendt tilbake på riktig format og med riktig innhold.

For å sjekke selve andelen av koden som vi tester, brukte vi testdekning. Rider inkluderer en funksjonalitet som viser andel kode testet i prosent. Funksjonaliteten viser også hva i programmet som er testet. I følge testdekningen er 65-70% av koden vår testet, som er litt under anbefalt testdekningsprosent i følge [71]. Vi har ikke inkludert testing for UpdateBirthday siden håndteringen av bursdager er en midlertidig løsning som oppdragsgiver kan endre når de overtar prosjektet. Dette fører til at prosenten ikke når over 70%. Ved å inkludere flere tester vil denne prosenten stige. Vi så ikke på dette som nødvendig på grunn av den midlertidige løsningen i prosjektet.

Listing 43: Kodelisting for testing av GetMyBooking funksjon.

```
1 [Fact]
2 public async Task TestGetMyBookingWithoutQueryValue()
3 {
4     var request = new DefaultHttpContext().Request;
5     // Get result of the Azure Function.
6     var result = await Company.Function.GetMyBooking.Run(request, null);
7
```

```
8     // Expected.
9     const string expectedValue = "Unspecified id for booking";
10    const int expectedStatusCode = 400;
11
12    // Actual.
13    var actual = Assert.IsType<BadRequestObjectResult>(result);
14
15    // Asserts.
16    Assert.Equal(expectedValue, actual.Value);
17    Assert.Equal(expectedStatusCode, actual.StatusCode);
18 }
```

8.3.2 Brukertesting

For å teste funksjonaliteten til programmet, ønsker vi å benytte oss av brukertesting. Ved å teste programmet fra start til slutt, kan vi verifisere om programmet svarer på problemstillingen. Brukertesten går ut på å logge seg inn på applikasjonen, se informasjon som ligger der og legge til reserveringer før brukeren logger seg ut igjen.

Da vi skulle gjennomføre brukertesting, lagde vi et spørreskjema. Se vedlegg D. Her blir testpersonen møtt med ulike oppgaver som den skal igjennom, før den må svare på noen påstander om oppgaven i etterkant. Et eksempel er: «Reserver to plasser på kontoret. Skaff deg oversikt over hvilke andre som har reservert seg en plass på de samme datoene. Prøv deretter å slette den ene reserveringen din.» Brukerne skal ikke hjelpes under testen. Dette er for at testpersonen selv skal bruke designet til å forstå funksjonalitetene til programmet. Siden denne applikasjonen bare skal brukes internt, har vi kun utført brukertest hos de ansatte.

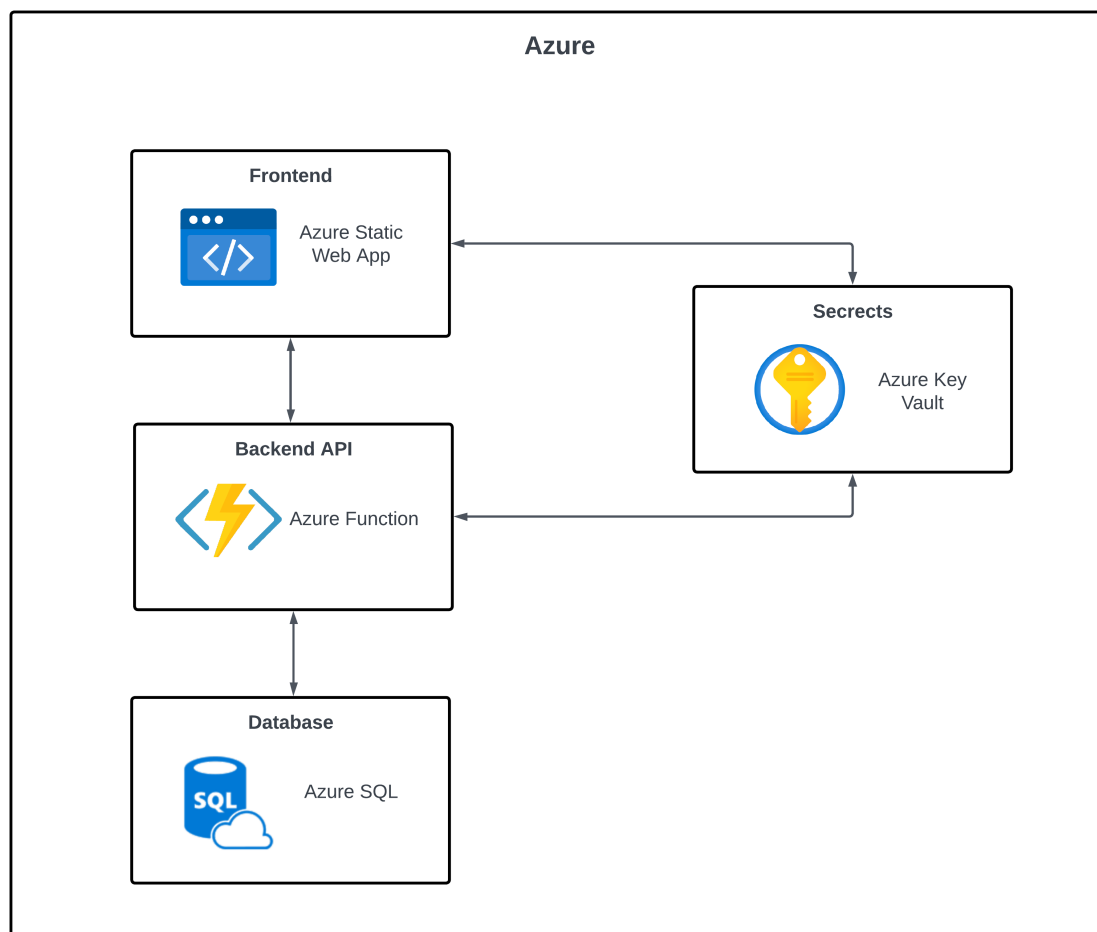
Resultatene fra brukertesten forteller at brukerne er fornøyde med funksjonalitetene, men at selve designet på noen komponenter kunne vært litt mer tydelig. Se vedlegg E. Brukerne er godt fornøyde med hjemmesiden og synes at det er enkelt å booke seg en plass. De synes også det er enkelt å logge ut og inn. De opplever også applikasjonen som brukervennlig. Vi fikk noen tilbakemeldinger om hva de likte minst. Disse var; «Kunne vært litt tydeligere at det var fullbooket.» og «Skulle gjerne sett en litt mer klar oversikt på hvem som har booka og eksakt når de har booka det.».

Kapittel 9

Produksjonssetting

Under utviklingsprosessen ble det satt opp et utviklingsmiljø i Microsoft Azure som gjorde det mulig å kjøre og produksjonssette applikasjonen. For å gjøre dette måtte det settes opp et abonnement i Azure som håndterer betaling, utgifter og budsjett for tjenesten.

Når abonnementet var på plass, fikk vi tildelt en ressursgruppe for å opprette ressurser. En ressursgruppe er hovedsakelig en container som Microsoft anbefaler å bruke til å gruppere tilhørende ressurser [72]. I løsningen tok vi i bruk Azure Static Web App for frontenden, Azure Function for API-et og Azure SQL for lagring av data. Key Vault ble også brukt for håndtering av *secrets*. Hvordan disse ressursene snakker sammen vises i figur 9.1.



Figur 9.1: Ressurser i Azure (laget i lucid.app).

9.1 Bicep

For opprettelse av ressursene, tok vi i bruk Bicep som er Microsoft sitt domenespesifikke språk for opprettelse av resurser og infrastruktur i Microsoft Azure [5].

Ved å ta i bruk Bicep kunne vi følge prinsipper fra boken *Infrascture As Code* som oppfordrer å skrive infrastrukturen som kode[73]. Fordelene som forkart i boken er at vi alltid vil være sikre på at serverene blir satt opp likt hver gang, samt at andre personer kan gå inn i koden for å se hvordan serveren er satt opp. Det vil også være mulighet for å ta i bruk koden i tilsvarende prosjekt i fremtiden.

I kodeliste 44 ser vi et forenklet eksempel på hvordan *Azure Functions* kan bli satt opp med hjelp av Bicep.

Listing 44: Bicep Functionapp.

```
1 resource functionApp 'Microsoft.Web/sites@2022-03-01' = {
2   name: functionName
3   location: location
4   ...
5   kind: 'functionapp'
6   properties: {
7     ...
8     siteConfig:{
9       netFrameworkVersion: 'v6.0'
10      ...
11    }
12  }
13 }
```

9.1.1 Azure Static Web App

Som forklart i seksjon 4.1.1 gjør Azure Static Web App det enklere å publisere applikasjonen for frontenden. Det er ikke nødvendig å spesifisere infrastrukturen i samme detalj som for Azure Function i kodelisting 44. Det eneste som er nødvendig å spesifisere er hvilken branch CI/CD pipelinen skal trigge på, samt lokasjonen på koden. Vi slipper altså å tenke på servere, samt hvilke pakker og språk som må installeres. Utenom det fungerer byggingen og publiseringen av koden på samme måte som fortalt videre.

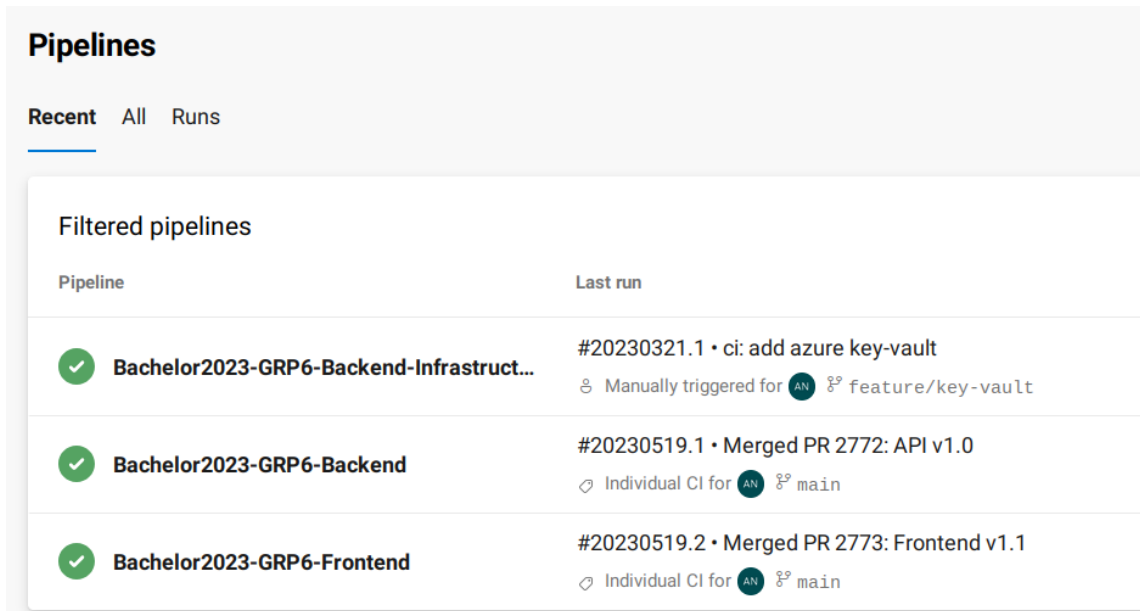
9.2 CI/CD

Under utviklingen ble det tatt i bruk kontinuerlig integrasjon(CI) og kontinuerlig leveranse(CD). Kontinuerlig integrasjon er praksisen for å automatisk teste og bygge applikasjonen for å oppdage feil raskest mulig. Kontinuerlig leveranse(CD) tar dette et steg videre og automatiserer publiseringen av koden om integrasjonen var vellykket.

Det ble ikke satt opp separate miljøer for produksjon og utvikling. Det vil si at hver *pull request* som ble merget inn i develop-branchen startet byggingen og publiseringen av applikasjonen. Mot slutten av utviklingsfasen gikk vi bort i fra publisering fra develop-branchen og gjorde at publiserings-pipelinen kjørte på endring i main-branchen i stedet. Dette ble gjort da all hovedfunksjonalitet var ferdig og nye endringer kunne testes lokalt fra develop før endringene ble publisert.

9.2.1 Pipelines

For å ta i bruk fordelene med CI/CD, samt muligheten til å teste funksjonaliteten fra skyen underveis i utviklingen, ble det satt opp pipelines for bygging og publisering av både frontend og backend. Det ble også satt opp en pipeline for backend-infrastrukturen for å gjøre det enklere å sette opp serveren på nytt hvis ønskelig. Denne blir kun kjørt manuelt. For å sette opp pipelineene brukte vi Azure Pipelines som er en del av Azure DevOps.



Figur 9.2: Pipelines i Azure DevOps.

I figur 9.2 ser vi pipelineene for backend, frontend og infrastrukturen til backenden. Hver av disse blir laget ved hjelp av YAML der vi spesifiserer hvilken branch den skal følge med på, samt hvilke steg den skal utføre. Det vil si at pipelineen kun kjører ved manuell start eller når kodeendringer skjer etter fullført merge. Det spesifiseres også variabler som navn og lokasjon til bruk for byggingen og publiseringen som vist i kodelisting 45.

Listing 45: Pipeline i YAML.

```

1 trigger:
2 - main
3
4 variables:
5   appName: 'func-bachelor2023-gr6-dev'
6   azureServiceConnection: 'SC-BachelorGr6'
7   resourceGroupName: 'bachelor-gruppe6-dev-rg'
8   location: resourceGroup().location #'West Europe'
9   projectFileLocation: 'src/evidi-office-portal.csproj'
10
11 stages:
12 - template: templates/ci.yaml
13   parameters:
14     projectFileLocation: '$(projectFileLocation)'
15
16 - template: templates/cd.yaml
17   parameters:
18     appName: '$(appName)'
19     azureServiceConnection: '$(azureServiceConnection)'

```

Går vi inn på en spesifikk pipeline-kjøring som i figur 9.4, kan vi se en oversikt over kjørte steg, tidsbruk og status om noe gikk galt.

Trykker vi oss videre inn på stegene, får vi opp en detaljert liste med kjørende/fullførte hendelser for den gitte jobben. Her kan vi se kommandoene som ble kjørt og eventuelle feilmeldinger som skulle oppstå. Når det første steget som bygger applikasjonen fullføres, genereres det en *artifact* som er koden som er bygget. Denne vil i publiserings-steget tas i bruk for publiseringen til Azure. Se figur 9.3

Jobs in run #20230505.2		Build function app	
Bachelor2023-GR6-Backend		<ul style="list-style-type: none"> 1 Pool: Azure Pipelines 2 Image: ubuntu-latest 3 Agent: Azure Pipelines 2 4 Started: May 5 at 12:32 PM 5 Duration: 25s 6 7 ▶ Job preparation parameters 8 📦 1 artifact produced 	
BuildFunctionApp			
▼	Build function app	25s	
	Initialize job	1s	
	Checkout Bachelor2023-GR6-Bac...	1s	
	CmdLine	12s	
	DotNetCoreCLI	5s	
	Archive files	2s	
	PublishBuildArtifacts	2s	
	Post-job: Checkout Bachelor202...	<1s	
	Finalize Job	<1s	
Deploy to DEV environment			
▼	Deploy backend to DEV	1m 3s	
	Initialize job	4s	
	Checkout Bachelor2023-GR6-Bac...	3s	
	Download Build Artifacts	2s	
	deploy backend app to DEV	1s	
	AzureFunctionApp	50s	
	Post-job: Checkout Bachelor202...	<1s	
	Finalize Job	<1s	

Figur 9.3: Jobber i pipeline.

✓ #20230505.2 • Merged PR 2728: fix: parametrized user input
 Bachelor2023-GR6-Backend

🔗 This run is being retained as one of 3 recent runs by pipeline.

Summary

Triggered by Henriette Eltvik

Repository and version Bachelor2023-GR6-Backend develop a29ba4e6	Time started and elapsed May 5 at 12:32 PM 1m 54s	Related 1 work items 1 published
--	---	--

Stages Jobs

✓ BuildFunctionApp

1 job completed 29s

📦 1 artifact

✓ Deploy to DEV enviro...

1 job completed 1m 7s

Figur 9.4: Oversikt over pipeline.

Kapittel 10

Diskusjon

10.1 Utviklingsprosess

Som nevnt i kapittel 2, er en kombinasjon av Scrum og Kanban tatt i bruk under utviklingen av prosjektet. Sprintlengden på to uker skulle gi oss mer tid til å forandre på design og funksjonalitet fra tilbakemeldingene vi fikk fra oppdragsgiver. Derimot ville en mindre sprintlengde ha gjort det mulig å ha hyppigere møter med oppdragsgiver, som kunne ha gitt oss flere tilbakemeldinger. Ulempen ville ha vært at oppgavene ikke hadde blitt ferdige i løpet av sprinten, da vi trengte tid til å teste og utvikle funksjonalitet før demonstrasjonen. Vi mener det var viktig å fullføre funksjonalitetene i hver sprint i stedet for å gjøre de halvferdige på en kortere sprintlengde. Dermed passet en sprintlengde på to uker godt for prosjektet, for å implementere ferdige funksjonaliteter i hver sprint.

I løpet av prosjektutviklingen endret vi arbeidsmengden på *issue*ne betraktelig. Vi lagde mindre og mer spesifikke *issues* for å fordele arbeidsmengden mer rettferdig, da det i starten var *issues* som krevde mer jobb enn andre. Dette førte til at det ble enklere å fordele oppgaver, samt se de nødvendige stegene i prosessen. *Pull requestene* ble også mindre på grunn av dette og det ble enklere å se konkret hva gruppemedlemmene arbeidet med og hva som hadde blitt implementert. Vi valgte å ha sprint-review og sprint-planlegging i samme møte med oppdragsgiver. I mange tilfeller ville det vært mer naturlig å dele dette i to ulike møter for å separere formålet. Fordelen med dette er at et møte bare tar for seg ett formål der man kun fokuserer på formålet. Ulempen er at man får to møter med et kort tidsintervall mellom dem. I prosjektet vårt har vi hatt lite med tid, som har ført til at vi har integrert både sprint-review og -planlegging i samme møte. Dette har vært for å utnytte tiden med å ha ett møte i stedet for to. I tillegg har vi også hatt muligheten til å legge til *issues* i løpet av sprinten, som førte til at planleggingsfasen ble kortere. Hadde vi hatt et større prosjekt med flere arbeidsoppgaver som krever nøyere review, hadde det vært mer ideelt med to separate møter.

Med disse erfaringene er vi fornøyde med valget av utviklingsmodell. Muligheten for å legge til *issues* i løpet av sprinten har vært avgjørende for å ferdigstille produktet.

10.2 Møter

Møtene med veileder ble godt utnyttet gjennom hele prosjektet. Rune har mye erfaring med å veilede tidligere bachelorstudenter og hvordan man skal følge en utviklingsprosess. I starten fikk vi mange tips om hvordan vi skulle få en god flyt på utviklingsprosessen. Når vi var kommet godt i gang med verktøyene i Microsoft Azure og utviklingen av programmet, ble gruppen mer selvstendige og behovet for veiledning ble mindre. Likevel hadde vi en arena for å dele tanker, vise progresjon og diskutere både problemstillinger og løsninger. Mot slutten av prosjektet ble veileder benyttet aktivt for å forbedre både struktur og språk i rapporten for å sikre en god flyt.

Møtene med oppdragsgiver ble brukt til å vise implementert funksjonalitet og diskutere nye løsninger. Disse møtene ble holdt på starten av hver sprint, som gjorde at gruppen fikk en tidsfrist for å gjøre programmet klart for presentasjon før hvert møte. Vi fikk tilbakemeldinger på løsningen vår og diskuterte hva som måtte gjøres i kommende sprint. Gruppen sendte uformelle møteagendaer i Teams-kanalen, noe som skapte en uformell setting i møtet. Ved å skape en uformell setting ble det enklere å få en god dialog mellom partene. Dette ga en arena for diskusjon og ulike synspunkter mellom gruppen og oppdragsgiver.

Når vi ser tilbake på hvilke erfaringer vi sitter igjen med rundt kommunikasjon med oppdragsgiver, dro vi stor nytte av å ha en Teams-kanal. Her kunne vi raskt komme i kontakt med oppdragsgiver. I etterkant ser vi også at vi kunne ha vært flinkere til å spørre om ting vi stod fast på. Dette ville ha løst problemet raskere, noe som kunne spart både tid og ressurser.

10.3 Manglende funksjonalitet

Fra prosjektplanen i vedlegg A, planla vi å lage en informasjonstavle som vist i skissen 5.4. Her tenkte vi å visualisere viktige meldinger og hendelser, men dette ble i senere tid nedprioritert da vi fokuserte på hovedfunksjonalitetene først. Det var også en idé å lage en adminside for administrasjon av slike meldinger, men det ble ikke noe behov da tavlen ikke ble implementert. I skissen 5.5 ser vi også en knapp med tekst «ENDRE» som gjør det mulig å endre en reservasjon. Denne funksjonen ble utelatt da det er like enkelt å slette reservasjonen og lage en ny med riktig tidspunkt.

10.4 Brukergrensesnitt

Designet i oppgaven var åpent og gruppen kunne fritt bestemme hvordan brukergrensesnittet skulle se ut. Dette førte til at vi brukte mye tid og ressurser på å lage selve designet på nettsiden. Fordelen med dette var at vi selv kunne bestemme hvordan alt skulle se ut og hvilke komponenter vi ønsket å ha med. Ulempen var at funksjonalitetene til programmet var for åpne, noe som førte til at det ble misforståelser på hvilken funksjonalitet oppdragsgiver egentlig ønsket seg. Dette kan sees på skissene fra seksjon 5.3.1 vs. det endelige designet i seksjon 5.3.2. Disse misforståelsene ble fanget opp under møtene med oppdragsgiver. I ettertid har begge parter innsett at det hadde vært bedre å ha fastsatt mer konkrete funksjonaliteter for programmet og en mal vi kunne brukt for designet.

10.5 Utviklingsmiljøer

I løpet av prosjektet, har vi fått flere erfaringer med de ulike miljøene beskrevet i kapittel 7. Vi har fått mer erfaring med bruk av Git og sett hvor viktig det er å bruke ulike grener og ha gode *commit*-meldinger. Gruppen har brukt reglene presentert i seksjon 2.3.1, men til tider har det sklidd litt ut. Bruken av *feat* og *style* har blitt brukt om hverandre. Grunnen til dette er at vi ikke hadde definert klar nok bruk av disse to. Hyppigheten av *commitene* har også variert. På starten av prosjektet der vi utviklet mye kode, fulgte vi standarden der en *commit* skulle omfatte en funksjonalitet. I senere tid når det ble mer refaktorering og endring av koding, endte *commit*-meldingene til å omfatte større kodedeler. Dette kunne vært unngått om man hadde små og veldefinerte *issues* som omfattet refaktorering og endring av kode.

Vi har også fått god erfaring med bruk av *pull request* og *code review*. I løpet av prosjektet har størrelsen på *pull requestene* variert. Det var enklere å gi konstruktive og gode tilbakemeldinger når det ikke var for mange filer og *commit*-meldinger å se gjennom. I ettertid ser vi også at vi kunne ha brukt mer tid på å se gjennom og teste koden som lå under *pull requesten*. Dette ville ha fanget feil tidligere og optimalisert koden ytterligere.

Ingen av gruppemedlemmene hadde erfaring med Microsoft Azure og ressursene som den tilbyr. Vi brukte dermed en del tid i starten av prosjektet på å sette oss inn i Microsoft Azure, noe som følte litt overveldende på grunn av alle tjenestene som fantes. For å kunne velge teknologiene vi skulle bruke i prosjektet, måtte vi først lese om flere ulike teknologier før vi bestemte oss. Dette gjorde at vi ble sikre på det endelige valget vi tok og vi fikk en bred forståelse rundt teknologiene. Microsoft Azure har mye dokumentasjon som medførte at vi fikk satt opp ressursene på en effektiv måte.

I prosjektet har vi brukt Discord som kommunikasjonsplattform internt i gruppen. Siden gruppen som regel har møttes fysisk på campus hver dag, har vi for det meste brukt Discord som en felles notatblokk. Her har vi lagret nyttige lenker og tidsfrister. Vi har også opprettet egne kanaler for deler i rapporten, der vi har skrevet nyttige tips til hverandre. Dette har fungert godt for hele gruppen da vi brukte plattformen daglig. Ved siden av dette har vi også benyttet Discord til å snakke sammen via videosamtaler de dagene vi satt hver for oss. Til tider kunne plattformen føles litt uoversiktlig når det ble mye tekst i de ulike kanalene. Dette løste vi fort ved å lage flere kanaler med spesifikke tema.

10.6 ChatGPT

I løpet av prosjektet har vi benyttet oss av verktøyet ChatGPT på ulike måter. Vi har brukt verktøyet som en søkemotor og problemløser. Se vedlegg F. Når vi møtte på et problem, var det veldig effektivt å spørre ChatGPT der man også kunne stille oppfølgings spørsmål.

Siden ChatGPT er en chatrobot som baserer seg på kunstig intelligens, har det vært viktig for gruppa å være kritiske til svarene. Når vi brukte roboten som en søkemotor var det viktig at vi sammenlignet svarene fra roboten med flere kilder. Dette gjaldt for eksempel når vi spurte om *localStorage*. Først spurte vi ChatGPT om mulige løsninger på problemet vårt. Deretter søkte vi opp potensielle andre løsninger og bedre definisjoner på informasjonen som ChatGPT kom med. Et annet argument for at vi var kritiske ved bruken av ChatGPT, var at den er en ganske ny tjeneste som har ikke tilgang til informasjon som er publisert etter 2021. I tillegg har ikke tjenesten direkte kilder til informasjonen den viser som gjør at vi ble skeptiske på det vi fikk tilbake.

10.7 Løsning

Hensikten med oppgaven var å gi Evidi en ansattportal for å gjøre det mulig å reservere kontor plass, samt få oversikt over informasjon for kommende møter og bursdager. Tidlig i utviklingsfasen tok vi i bruk React for frontenden som i senere tid kom med sine problemer med bruk av miljøvariabler. Dette medførte at vi valgte å gå over til Next.js for å kunne bruke serverkomponenter som nevnt i 6.7. Ulempen med denne overgangen var at det var relativt tidskrevende mot slutten av utviklingsfasen, som igjen forskjøv brukertesting lenger enn nødvending. Det ble sett på alternative metoder hvor vi kunne hente *secrets* fra Key Vault direkte fra React-applikasjonen. Ulempen med dette var at *secrets* måtte hentes via autoriserte brukere som ville ha gjort koden mer avansert enn ved bruk av serverkomponenter. Vi kunne også latt være å gå over til bruk av serverkomponenter ved å skru av tilgangshåndteringen til API-et, men dette var ikke aktuelt.

Som det vises i seksjon 5 er frontend beskrevet som en “relativ” tynn klient. At den er “relativ” vil si at frontenden inkluderer for mange logiske operasjoner for at det kan bli kalt en fullstendig tynn klient. Senere ser vi at innloggingsfunksjonaliteten og henting av data fra Microsoft Graph kunne ha vært implementert i backend. Resultatet hadde vært at frontend bare hadde vist designet og håndtert interaksjonen mellom brukeren og webapplikasjonen. Dermed hadde backend håndtert alt av forretningslogikken som inkluderer manipulering av data i databasen, integrering av innloggingen og henting av data fra Microsoft Graph. En fordel med en slik arkitektur hadde gjort det enklere for videreutvikling. Det er også mer sikkert å inkludere selve behandlingen av *tokens* i backend siden det reduserer sjansene for typiske webangrep som CSRF (Cross Site Request Forgery)-angrep og autorisasjonskode-injeksjonsangrep.

Brukertesting ble utført mot slutten av prosjektet på den publiserte versjonen. Fra brukertestene i seksjon 8.3.2 kan vi se at brukerne opplevde grensesnittet som intuitivt med en god informasjonsstruktur. De savnet derimot en mer tydelig funksjonalitet som viste når det var fullt og en litt mer klar oversikt over hvem andre som har booka. I ettertid innser vi at det hadde vært lurt om vi hadde gjennomført brukertest på et tidligere stadium. Da hadde vi hatt mulighet for å endre programmet basert på tilbakemeldingene vi fikk, før vi hadde tatt en ny runde med brukertesting. Dette ville ført til en bedre kommunikasjon mellom brukerne og gruppen som utviklere, noe som igjen ville ført til en mer brukervennlig applikasjon.

Implementeringen av pop-up-vinduet for å legge til brukerens bursdag er en midlertidig løsning. Vi har tatt i bruk *localStorage* som både har sine fordeler og ulemper. Ulempen er at *localStorage* lagrer en variabel lokalt i nettleseren. Hvis en ansatt ved Evidi besøker webapplikasjonen på en annen nettleser, eksisterer ikke variabelen på denne nettleseren og man må derfor legge til bursdagen på nytt. Fordelen med implementeringen er at *localStorage* er enkel å ta i bruk og endre, som gjør det enklere for Evidi å videreutvikle programvaren. Valget med å bruke *localStorage* i den midlertidige løsningen passet godt da vi ikke trengte å bruke mye tid på å implementere det.

Under utviklingen valgte vi som nevnt i kapittel 9 å kun bruke ett produksjonsmiljø. I senere tid ser vi fordelene med å ta i bruk to eller flere produksjonsmiljøer. Problemet med kun ett produksjonsmiljø er at løsningen må kjøres og testes i samme miljø. Det vil si at applikasjonen som er i bruk kan få problemer ved videreutvikling, da det ikke er mulig å teste løsningen i skyen før den blir tilgjengeliggjort for brukerne. Hadde vi satt opp et testing-miljø med egne servere, kunne vi ha kvalitetssikret og publisert videreutviklet kode før den blir lansert hos produksjonsmiljøet.

10.8 Evaluering av gruppens arbeid

I løpet av prosjektet har gruppen hatt en god arbeidsfordeling der hvert gruppelem har brukt omtrent like mye tid på prosjektet. Se vedlegg G for oversikt over timer brukt på prosjektet. Dette har bidratt til jevn fremdrift. Siden vi er en liten gruppe, var det nødvendig å dele opp i ansvarsområder. Vi ønsket tidlig å koble alle delene av systemet sammen slik at en funksjonalitet ble fullført på alle deler. Imidlertid har det vært en ulempe med ansvarsfordelingen mellom frontend, backend og infrastruktur som fordelt i seksjon 1.7. Dette førte til at medlemmene fikk en mindre helhetsforståelse av systemet fra starten og enkelte valg ble gjort på eget initiativ selv om vi var finke til å sparre. Utover prosjektet ble vi flinkere til å ta på oss oppgaver på andre ansvarsområder.

For å forbedre gruppearbeidet i fremtiden, kan det være nødvendig å sikre en mer balansert fordeling av oppgavene på tvers av ansvarsområdene underveis i utviklingen. Det hadde også vært en fordel å holde kræsje-kurs på enkelte deler av løsningen som enkelte gruppelemmer ikke har vært en del av.

Utenom dette har samarbeidet i gruppen vært bra. Vi har hjulpet hverandre ved å utføre parkoding på komplekse og kritiske oppgaver, samt vært motiverende støttespillere. Vi har også sittet svært mye sammen som har gjort at vi har blitt bedre kjent og lært mye om hverandre.

Kapittel 11

Konklusjon

11.1 Måloppnåelse

Evidi ønsket en informasjonsplattform som presenterer møter og bursdager. I tillegg skulle plattformen inneholde funksjonalitet for å reservere kontorplasser.

Løsningen vår er en fullstack-løsning bestående av backend-server som kommuniserer med en database og en frontend-applikasjon som samhandler med brukeren. Løsningen er integrert med Microsoft Graph og Microsoft Identity Platform. Ved hjelp av Microsoft Azure har vi satt opp pipelines for å integrere automatisk bygg og publisering.

Produktet vårt inneholder kode med høy kvalitet, modularitet og god dokumentasjon. Dette gjør det enkelt å videreutvikle programmet. Løsningen vår tar også hensyn for å skalere programmet etter behov.

Alle funksjonalitetene som er oppgitt i oppgavebeskrivelsen 1.2 er oppnådd noe som oppfyller resultatmålene. Oppdragsgiver er fornøyd med sluttproduktet, noe som kom frem under det siste møte med oppdragsgiver da brukertesten var gjennomført.

Løsningen vår presenterer en programvare som gjør det mulig å utnytte kapasiteten på kontoret samtidig som man beholder kulturen om et åpent kontorlandskap med frie sitteplasser. Et sosialt og inkluderende arbeidsmiljø vil føre til sterkere relasjoner mellom de ansatte. Dette vil bidra til et sterkere samhold i bedriften og en følelse av fellesskap. Alle effektmålene er også oppfylt, der løsningen gjør det mer effektivt for brukeren å bestille plasser.

Gjennom prosjektet har vi lært å jobbe i et større prosjekt som baserer seg på en smidig utviklingsmetodikk. Vi har skaffet oss gode erfaringer og kunnskap ved å utvikle en webapplikasjon med tilhørende backend der vi har fått mer erfaring i testing og kvalitetskontroll av programvaren. I tillegg har vi også lært mye om hele utviklingsfasen: fra idé til publisering av kode.

Til slutt konkluderer vi med at løsningen svarer godt på problemstillingen der den oppfyller alle funksjonalitetene fra oppgavebeskrivelsen på en god måte i tillegg til at alle målene er oppnådd.

11.2 Videre arbeid

Brukerguide i vedlegg H er utviklet for å hjelpe utviklere for videre arbeid. Denne finnes også i README-fila i repo for både backend og frontend. Det første gruppen ville ha jobbet videre med, er å flytte en del forretningslogikk fra frontend til backend for å få en tynnere klient. Videre hadde dette gjort det enklere for Evidi å videreutvikle programmet når all logikken ligger i backend, mens frontend bare håndterer design.

En annen ting som kan forbedres er håndteringen av bruker-ID i databasen. Problemet er at nåværende løsning håndterer to forskjellige ID-er som begge er unike. Her kunne vi tatt i bruk ID-en vi får etter innlogging via Microsoft for å forenkle databasen.

Fra de spesifikke tilbakemeldingene vi fikk fra resultatene fra brukertestingene, ser vi at noe av designet til noen av komponentene på bookingsiden kan endres. Det vil være en idé å endre stolpediagrammet til et sirkeldiagram. Da er det enklere å se makskapasiteten på kontoret. For videreutvikling av designet vil det være hensiktsmessig å følge retningslinjene til WCAG 2.1 for å tilrettelegge for blant annet svaksynte og fargeblinde [74].

En mobilapplikasjon ville ha utvidet løsningen betydelig. Man hadde fått flere brukere og muligheten til å sende push-varslinger. Dette ville ha gjort løsningen lettere tilgjengelig for brukerne, der man får muligheten til å booke seg en plass uten å bruke datamaskinen.

Det vil også være nyttig å opprette et nytt produksjonsmiljø for testing og videreutvikling av ny funksjonalitet. Dette er fordi nåværende løsning kun publiserer funksjonalitet som er ferdigutviklet og merget inn i main. Det vil si at det ikke er mulig å teste applikasjonen i skyen før den blir tilgjengeliggjort for alminnelig bruk.

Vi mener vi har lagt til rette for videreutvikling av løsningen, der koden er dokumentert både i selve koden og i rapporten. Etterhvert som Evidi jobber med videreutviklingen og endrer eller legger til funksjonaliteter, vil denne rapporten være en viktig dokumentasjon på løsningen.

Bibliografi

- [1] «Azure DevOps.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops> (sjekket 18.04.2023).
- [2] «Azure Functions.» (), adresse: <https://azure.microsoft.com/en-us/products/functions> (sjekket 18.05.2023).
- [3] «Static Web Apps.» (), adresse: <https://azure.microsoft.com/en-us/products/app-service/static> (sjekket 18.05.2023).
- [4] M. Granevang. «backend.» (2020), adresse: <https://snl.no/backend> (sjekket 17.04.2023).
- [5] «What is Bicep?» (), adresse: <https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep> (sjekket 21.04.2023).
- [6] «A tour of the C# language.» (2023), adresse: <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/> (sjekket 19.04.2023).
- [7] T. H. Nätt. «CSS.» (2020), adresse: <https://snl.no/CSS> (sjekket 17.04.2023).
- [8] M. Granevang. «frontend.» (2020), adresse: <https://snl.no/frontend> (sjekket 17.04.2023).
- [9] T. H. Nätt. «HTML.» (2021), adresse: <https://snl.no/HTML> (sjekket 17.04.2023).
- [10] T. H. Nätt. «JavaScript.» (2020), adresse: <https://snl.no/JavaScript> (sjekket 17.04.2023).
- [11] «Get to know Azure.» (2023), adresse: <https://azure.microsoft.com/en-us/explore/> (sjekket 18.04.2023).
- [12] L. Gupta. «What is REST.» (2022), adresse: <https://restfulapi.net/> (sjekket 19.04.2023).
- [13] «Access Tokens.» (2023), adresse: <https://auth0.com/docs/secure/tokens/access-tokens> (sjekket 19.04.2023).
- [14] . foundation. «About xUnit.net.» (2023), adresse: <https://xunit.net/> (sjekket 17.04.2023).
- [15] «YAML: YAML Ain't Markup Language™.» (), adresse: <https://yaml.org/> (sjekket 15.05.2023).
- [16] «Use best practices when implementing Kanban in Azure Boards.» (), adresse: <https://learn.microsoft.com/en-us/azure/devops/boards/boards/best-practices-kanban?view=azure-devops> (sjekket 11.05.2023).
- [17] «Git –fast-version-control.» (), adresse: <https://git-scm.com/about> (sjekket 18.05.2023).
- [18] M. Tsitoara, «Git Best Practices,» s. 79–81, 2019. doi: https://doi.org/10.1007/978-1-4842-5313-7_6.

- [19] «Commit Message Guidelines.» (2018), adresse: <https://github.com/angular/angular/blob/22b96b9/CONTRIBUTING.md#commit-message-guidelines> (sjekket 08.05.2023).
- [20] «About pull requests.» (), adresse: <https://learn.microsoft.com/en-us/azure/devops/repos/git/about-pull-requests?view=azure-devops> (sjekket 11.05.2023).
- [21] «Why Azure.» (), adresse: <https://azure.microsoft.com/en-us/explore/#why-azure> (sjekket 21.04.2023).
- [22] «Hva er egentlig Microsoft Azure?» (), adresse: <https://www.evelon.no/artikler/hva-er-egentlig-microsoft-azure> (sjekket 21.04.2023).
- [23] «React Community.» (2023), adresse: <https://react.dev/community> (sjekket 17.04.2023).
- [24] «React.» (2023), adresse: <https://react.dev/> (sjekket 17.04.2023).
- [25] «The React Framework for the Web.» (2023), adresse: <https://nextjs.org> (sjekket 20.04.2023).
- [26] «Overview of Microsoft Graph.» (2023), adresse: <https://learn.microsoft.com/en-us/graph/overview> (sjekket 17.04.2023).
- [27] «Microsoft Graph Toolkit overview.» (2022), adresse: <https://learn.microsoft.com/en-us/graph/toolkit/overview> (sjekket 18.04.2023).
- [28] «Person component in Microsoft Graph Toolkit.» (), adresse: <https://learn.microsoft.com/en-us/graph/toolkit/components/person> (sjekket 23.04.2023).
- [29] «Agenda component in Microsoft Graph Toolkit.» (), adresse: <https://learn.microsoft.com/en-us/graph/toolkit/components/agenda> (sjekket 23.04.2023).
- [30] «Material UI - Overview.» (2023), adresse: <https://mui.com/material-ui/getting-started/overview/> (sjekket 18.04.2023).
- [31] «Date Picker.» (2023), adresse: <https://mui.com/x/react-date-pickers/date-picker/> (sjekket 18.04.2023).
- [32] «Date and Time Pickers - Localization.» (), adresse: <https://mui.com/x/react-date-pickers/localization/> (sjekket 24.04.2023).
- [33] «Date and Time Pickers - Gettings Started.» (), adresse: <https://mui.com/x/react-date-pickers/getting-started/> (sjekket 24.04.2023).
- [34] «Dialog.» (), adresse: <https://mui.com/material-ui/react-dialog/> (sjekket 21.04.2023).
- [35] «Chart.js.» (2023), adresse: https://www.w3schools.com/ai/ai_chartjs.asp (sjekket 18.04.2023).
- [36] «Chart.js.» (2023), adresse: <https://www.chartjs.org/docs/latest/> (sjekket 18.04.2023).
- [37] «cookie-cutter.» (), adresse: <https://www.npmjs.com/package/cookie-cutter> (sjekket 24.04.2023).
- [38] «Timer trigger for Azure Functions.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-timer?tabs=python-v2%5C%2Cin-process&pivots=programming-language-csharp> (sjekket 20.04.2023).
- [39] «What is Azure SQL?» (2022), adresse: <https://learn.microsoft.com/en-us/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview?view=azuresql> (sjekket 21.04.2023).
- [40] «Getting Started with xUnit.net.» (), adresse: <https://xunit.net/docs/getting-started/netfx/jetbrains-rider> (sjekket 26.04.2023).

- [41] «What is Azure Active Directory?» (), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis> (sjekket 16.05.2023).
- [42] «What is the Microsoft identity platform?» (2023), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-overview> (sjekket 11.05.2023).
- [43] «Overview of the Microsoft Authentication Library (MSAL).» (2022), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/develop/msal-overview> (sjekket 19.04.2023).
- [44] N. A. Microsoft Security. «What is the Microsoft Authentication Library (MSAL)?» (Jun. 2021), adresse: <https://youtu.be/zufQ0QRUHuk> (sjekket 21.05.2023).
- [45] «Microsoft Authentication Library for React (msal-react).» (2023), adresse: <https://github.com/AzureAD/microsoft-authentication-library-for-js/tree/dev/lib/msal-react/#about.md> (sjekket 20.04.2023).
- [46] «Key Vault.» (2023), adresse: <https://azure.microsoft.com/en-us/products/key-vault/> (sjekket 03.05.2023).
- [47] «Working with JSON.» (2023), adresse: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON> (sjekket 20.04.2023).
- [48] «Azure Functions HTTP triggers and bindings overview.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-http-webhook?tabs=in-process%2Cfunctionsv2&pivots=programming-language-csharp> (sjekket 09.05.2023).
- [49] «Evidi.» (2022), adresse: <https://www.evidi.com/> (sjekket 02.05.2023).
- [50] «Application types for the Microsoft identity platform.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-app-types> (sjekket 10.05.2023).
- [51] «Microsoft identity platform and OAuth 2.0 authorization code flow.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-auth-code-flow#redeem-a-code-for-an-access-token> (sjekket 09.05.2023).
- [52] «Quickstart: Register an application with the Microsoft identity platform.» (), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app> (sjekket 19.05.2023).
- [53] «Configuration Options.» (), adresse: <https://github.com/AzureAD/microsoft-authentication-library-for-js/blob/dev/lib/msal-browser/docs/configuration.md> (sjekket 24.04.2023).
- [54] «Module @azure/msal-react.» (), adresse: https://azuread.github.io/microsoft-authentication-library-for-js/ref/modules/_azure_msal_react.html#usemsal (sjekket 23.04.2023).
- [55] «Module @azure/msal-browser.» (), adresse: https://azuread.github.io/microsoft-authentication-library-for-js/ref/modules/_azure_msal_browser.html#publicclientapplication (sjekket 23.04.2023).
- [56] «PublicClientApplication Class.» (), adresse: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.identity.client.publicclientapplication?view=msal-dotnet-latest> (sjekket 23.04.2023).
- [57] «Hooks - useMsal hook.» (), adresse: <https://github.com/AzureAD/microsoft-authentication-library-for-js/blob/dev/lib/msal-react/docs/hooks.md> (sjekket 21.05.2023).

- [58] «azure/msal-react package.» (), adresse: <https://learn.microsoft.com/en-us/javascript/api/@azure/msal-react/#@azure-msal-react-getchildrenorfunction> (sjekket 11.05.2023).
- [59] «Module @azure/msal-react - microsoft-authentication-libraries-for-js.» (), adresse: https://azuread.github.io/microsoft-authentication-library-for-js/ref/modules/_azure_msal_react.html#authenticatedtemplate (sjekket 14.05.2023).
- [60] «Single-page application: Sign-in and Sign-out.» (), adresse: <https://learn.microsoft.com/en-us/azure/active-directory/develop/scenario-spa-sign-in?tabs=javascript2> (sjekket 21.05.2023).
- [61] «RESTful web API design.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design> (sjekket 05.05.2023).
- [62] «Data Fetching.» (2023), adresse: <https://nextjs.org/docs/app/building-your-application/data-fetching> (sjekket 11.05.2023).
- [63] «Moving Client Components to the Leaves.» (2023), adresse: <https://nextjs.org/docs/getting-started/react-essentials#moving-client-components-to-the-leaves> (sjekket 13.05.2023).
- [64] «IntelliJ IDEA overview.» (2023), adresse: <https://www.jetbrains.com/help/idea/discover-intellij-idea.html> (sjekket 09.05.2023).
- [65] «ReSharper - The Visual Studio Extension for .NET Developers.» (), adresse: <https://www.jetbrains.com/dotnet/guide/products/resharper/> (sjekket 23.04.2023).
- [66] «Fast powerful cross-platform .NET IDE.» (), adresse: <https://www.jetbrains.com/rider/> (sjekket 27.04.2023).
- [67] «What Is Swagger?» (), adresse: <https://swagger.io/docs/specification/2-0/what-is-swagger/> (sjekket 23.04.2023).
- [68] «What is Azure DevOps?» (2022), adresse: <https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops> (sjekket 01.05.2023).
- [69] «C Coding Conventions.» (), adresse: <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions> (sjekket 12.05.2023).
- [70] «React coding Standards and Practices.» (), adresse: <https://medium.com/@navitasinghal77/react-coding-standards-and-practices-3b133bcaea8> (sjekket 12.05.2023).
- [71] S. Pittet. «What is code coverage?» (), adresse: <https://www.atlassian.com/continuous-delivery/software-testing/code-coverage> (sjekket 19.05.2023).
- [72] «Manage Azure resource groups by using the Azure portal.» (2023), adresse: <https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal> (sjekket 10.05.2023).
- [73] K. Morris, «Infrastructure as Code,» s. 10–11, 2021. doi: 978-1-098-11467-1.
- [74] «Web Content Accessibility Guidelines (WCAG) 2.1.» (), adresse: <https://www.w3.org/TR/WCAG21/> (sjekket 16.05.2023).

Vedlegg A

Prosjektplan



Prosjektplan

Henriette Eltvik
Alf Magnus Nohr
Sigrid Warberg Vestheim
Gjøvik
31.januar 2023

Innhold

1	Mål og rammer	2
1.1	Bakgrunn	2
1.2	Prosjekt mål	2
1.3	Rammer	3
2	Omfang	4
2.1	Fagområde	4
2.2	Avgrensning	4
2.3	Oppgavebeskrivelse	4
3	Prosjektorganisering	5
3.1	Ansvarsforhold og roller	5
3.2	Rutiner og grupperregler	6
3.2.1	Grupperregler	6
3.2.2	Rutiner	6
4	Planlegging, oppfølging og rapportering	7
4.1	Hovedinndeling av prosjektet	7
4.2	Utviklingsmodell	7
4.3	Plan for statusmøter og beslutningspunkter i perioden	7
5	Organisering av kvalitetssikring	8
5.1	Dokumentasjon	8
5.2	Standarder	8
5.3	Utviklingsrutiner	8
5.4	Verktøy	9
5.5	Plan for inspeksjoner og testing	9
5.6	Risikoanalyse på prosjektnivå	10
6	Plan for gjennomføring	11
6.1	Gantt diagram	11
6.2	Milepæler	12

1 Mål og rammer

1.1 Bakgrunn

Evidi er et konsultentselskap som leverer digitale løsninger for kundene sine. De har spesialisert seg i Microsoft sine teknologier og mener at styrken ligger i helheten. De er opptatte av å skape en god fremtid og satser dermed på bærekraft. Evidi har hovedkontoret sitt i Oslo, men man kan også finne kontorer i Halden, Sarpsborg, Asker, Tønsberg, Bergen og Danmark.

Evidi ønsker å utvikle en ansattportal der man har mulighet til å se ledig kapasitet på kontorene og booke seg en plass. I tillegg til dette, ønsker de muligheten til å presentere felles informasjon som bursdager, møter og sosiale arrangement på en god måte for de ansatte.

1.2 Prosjektmål

Målene for prosjektet har vi valgt å dele inn i tre kategorier: resultatmål, effektmål og læringsmål. Resultatmål sier noe om hva prosjektet konkret skal oppnå og er dermed knyttet til prosjektets resultater og leveranser. Effektmål sier noe om målsettingen til prosjektet og grunnen til at vi jobber med denne oppgaven. Læringsmål beskriver hva vi som studenter forventer å lære av prosessen og oppgaven. Resultatet av oppgaven skal være en løsning som gjør det enklere for de ansatte å få med seg felles informasjon og å være sikker på at de får en plass på kontoret.

Resultatmål

I løpet av prosjektet skal det produseres en løsning som gjør det mulig for ansatte å booke en plass på kontoret samtidig som de skal få med seg felles informasjon. Målet er å lage et enkelt og intuitivt brukergrensesnitt som vil kommunisere med et RESTful API i backend.

- Integre en innloggingside med eksisterende Microsoft-konto, og Microsoft Authenticator.
- Applikasjonen skal ta i bruk eksterne API'er, som Teams og Outlook.
- Applikasjonen skal inneholde de spesifiserte funksjonalitetene nevnt i oppgavebeskrivelse 2.3.
- Lage et intuitivt brukergrensesnitt.

Effektmål

Målet er at det skal være enklere for de ansatte å kunne se om det er ledig plass på kontoret og å kunne få med seg felles informasjon, som møter og bursdager.

- Skal gjøre det mer effektivt for brukeren å bestille plasser.
- Det sosiale samværet mellom de ansatte skal øke.
- Det skal være enklere å få beskjeder fra kollegaer.

Læringsmål

Gjennom prosjektet ønsker vi å lære mest mulig om hvordan det er å arbeide smidig som team, og å lære å bruke Microsoft Azure sine verktøy for utvikling av nye systemer. Vi vil fokusere på:

- Ta i bruk SCRUM som smidig utviklingsmetodikk.
- Bruke Azure og dens teknologier.
- Utvikle webapplikasjon med tilhørende backend.
- Lage intuitive brukergrensesnitt.
- Lære oss å bruke testing og kvalitetskontroll.
- Automatisert bygg og release(CI/CD).

1.3 Rammer

Tidsramme

Prosjektet har en tidsramme fra 12.januar 2023 til 22.mai 2023. Da skal både prosjektet og rapporten være klare til levering. Gruppen har planlagt ferdigstilling av selve produktet til rundt 10.april. Etter produktet er komplett, så bruker gruppen resten av tiden på å ferdigstille selve rapporten.

Teknologiramme

- Webgrensesnittet skal fungere på nettleserne FireFox 109.0, Google Chrome 109.0.5414.76, samt nyere versjoner.
- Løsningen skal ha et enkelt og intuitivt brukergensnitt.
- Tjenesten skal kunne deployes i Azure.

2 Omfang

2.1 Fagområde

Evidi ønsker en løsningen som gjør det enklere for de ansatte å kunne booke seg en plass på kontoret. Dette vil føre til at flere kan møtes og bidra til et sterkere sosial miljø. I tillegg ønsker de å gjøre det enklere for de ansatte å få med seg felles informasjon, som bursdager og møter.

I dag har de ingen løsning. All kommunikasjon foregår gjennom Teams, Slack eller muntlig fra person til person. Når det blir mange kanaler å forholde seg til, er det lett å miste viktig informasjon. Vi ønsker å løse dette problemet ved å lage en effektiv webapplikasjon som presenterer viktige meldinger.

2.2 Avgrensning

- Oppgaven avgrenser seg til å jobbe med kontorstrukturen i Oslo.
- I starten av prosjektet vil vi kun se nærmere på to eksterne API; Microsoft Outlook og Teams. Der vi kun vil bruke informasjon som gjelder bursdager og arrangement / møter som gjelder for hver enkelt ansatt.
- Språket for applikasjonen vil kun være på engelsk.

2.3 Oppgavebeskrivelse

Oppgaven vår er å levere en webapplikasjon som de ansatte i Evidi skal bruke. Denne webapplikasjonen skal ha følgende funksjonalitet:

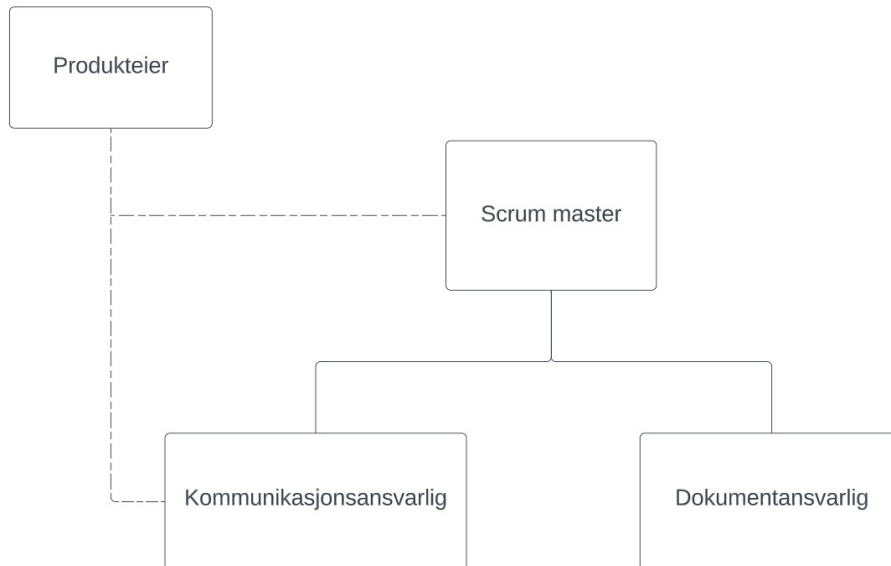
- Brukeren må logge inn med eksisterende Microsoft-konto og Microsoft Authenticator.
- Brukeren skal kunne booke seg en plass på kontoret på en bestemt dato. Bookingen for en ny uke åpner søndag før den aktuelle uken begynner.
 - Ledige plasser og bookinger skal presenteres visuelt.
 - Det må være mulig å slette bookingen.
 - Det må også være mulig å endre både tidspunkt og dato på bookingen.
- Det skal være et informasjonsfelt på siden der man finner informasjon om Dagens bursdag, «Dagens møter» / kurs / arrangement. Denne informasjonen blir oppdatert automatisk fra API til Outlook og Teams.
- En informasjonstavle skal være tilgjengelig med relevante møter og felles beskjeder.

I tillegg vil det være en admin som har flere rettigheter enn de vanlige brukerne. Denne adminbrukeren vil kunne legge til / fjerne kontorplasser, legge til / fjerne ansatte og kontrollere bookinger.

3 Prosjektorganisering

Gruppen består av to dataingeniørstudenter og en programmeringsstudent. For å kunne effektivisere arbeidet i gruppen, har vi definert gitte roller og ansvarsområder som definert i 3.1.

3.1 Ansvarsforhold og roller



Figur 1: Organisasjonskart.

- **Produkteier** - Jørgen Mobakken
- **Scrum master** - Alf Magnus Nohr
 - Ansvarlig for progresjon og at utviklingen utføres i samhold med forutsatte krav.
 - Håndtere konflikter og iverksette tiltak hvis nødvending.
 - Innkalle og lede Scrum-relaterte møter som sprint planlegging, review og retrospective.

Siden vi er en liten gruppe, kan noen av oppgavene til scrum master sees på som typiske prosjektlederopp-gaver.

- **Dokumentansvarlig** - Sigrid Warberg Vestheim
 - Følge opp dokumenter og dokumentere møter.
 - Organisere utvikling av brukerguide og følge opp dokumentasjon av systemet.
- **Kommunikasjonsansvarlig med veiledere** - Henriette Eltvik
 - Sende inn møteinnkallelser og agenda for møtene.

I tillegg til internrollene som vist i figur 1, vil alle gruppemedlemmene ha roller som utviklere og bidra med utviklingen ut fra utviklingsmodellen beskrevet i 4.2. For å sikre fremgang i utviklingen, har vi ulike ansvarsområder. Henriette vil ta hovedansvar for testing, Alf tar hovedansvar for backend og Sigrid tar hovedansvar for frontend. Det er forventet at alle skal jobbe med alle de tre ulike delene, men ved å fordele ansvar sikrer man fremgang i prosjektet.

3.2 Rutiner og gruppe­regler

Under finner man en beskrivelse på bestemte rutiner og gruppe­regler som gruppen har kommet med. Rutinene som er valgt, skal hjelpe gruppen i å organisere og prestere godt gjennom utviklingen av prosjektet. Hvis noen i gruppen ikke følger rutiner og gruppe­regler, får de en skriftlig advarsel. Etter to advarsler må de andre på gruppa sette inn ekstra tiltak. Hvis ting ikke blir bedre etter dette, må dette tas opp med veileder for diskusjon om veien videre.

3.2.1 Gruppe­regler

- Diskuter uenigheter i gruppen.
- Diskuter dårlig oppførsel i gruppen.
- Ikke vær redd for å spør om forklaring på visse begreper.
- Vær åpen om tema man ikke forstår.
- Møt opp til avtalte møter.
- Ved gjentakende distraksjoner fra gruppearbeid, så må prosjektleder ta styringen og få gruppen på riktig spor igjen.
- Oppgaver med frist skal leveres innen avtalt tidspunkt hvis det ikke er noen god grunn for forsinkelse.
- Det er forventet at alle gruppe­medlemmene skal være involverte i diskusjoner og møter.
- Oppgaver med tidsfrist skal leveres inn med god kvalitet.
- Minimum arbeidstid per medlem er 30 timer per veke, hvor gruppen skal sammen arbeide 20 timer.

3.2.2 Rutiner

Møter	Medlemmer som ikke kan delta på avtalt møte, må gi beskjed 24 timer i forkant. Vi har faste møter torsdag med veileder og hver fredag med intern gruppe. Medlemmene har daglige møter klokka 09:00 som varer 15 minutter, for å gå gjennom dagens oppgaver og eventuelle problemer.
Dokumenter	Kildekode og andre dokumenter vil bli delt over en felles plattform; Microsoft Azure DevOps. Evidi har ansvaret for å gi tilgang til gruppe­medlemmene.
Oppgavefordeling	Oppgaver vil bli publisert på et kanbanboard, som skal holde oversikt over ukentlige sprinter og oppgaver.
Innsending av dokumenter	Dokumentansvarlig skal sende inn dokumenter før tidsfrist.
Kommunikasjonsplattform	Møteinnkallelser og andre relevante gruppehendelser blir sendt over <i>Messenger</i> . Innkallelse til møtet skal skje 24 timer før avtalt møte.
Timenotering	Gruppe­medlemmene skal føre timer etter endt arbeidsdag.
Testing	Testing skjer fortløpende ved ny funksjonalitet.
Oppgavene	Fordelingen av oppgaver mellom gruppe­medlemmene står under «Ansvarsforhold og roller» (3.1).

4 Planlegging, oppfølging og rapportering

4.1 Hovedinndeling av prosjektet

Prosjektet vil bestå av to hoveddeler som vi deler inn i backend og frontend. Backend vil være et RESTful API og frontenden vil bli presentert som en webapplikasjon.

4.2 Utviklingsmodell

Da det ikke ble gitt noen konkret kravspesifikasjon fra Evidi var valget for en smidig utviklingsmodell en enkel avgjørelse. Vi er avhengig av å kunne legge til oppgaver fortløpende når nye ønsker kommer på banen, samt pushe til produksjon regelmessig.

Vi så derfor på Kanban som fokuserer på kontinuerlig arbeid uten fastsatte tidslinjer. Fordelen med Kanban er at man kan legge til issues og utføre oppgaver fortløpende etter prioritert. Det settes en WIP(work in progress)-limit slik at arbeidsmengden per person ikke blir for stor i forhold til kapasitet. Ulempen med en slik modell er at det er lite strukturert da det er fri flyt for å ta på seg oppgaver. Det er heller ingen fastsatte tidsrammer da man leverer funksjonalitet kontinuerlig. Noe som i seg selv kan være positivt, men ettersom vi ønsker en mer strukturert arbeidsflyt med fastsatte mål, falt valget mot Scrum.

Her blir oppgavene fastsatt for hver sprint slik at det blir lettere å definere og jobbe frem mot et felles mål. Ulempen er at tidslinjen kan bli stram, men det holdes regelmessige møter som kan hjelpe og løse problemer på en rask og effektiv måte. For å få det beste ut i fra begge, tenker vi å bruke en kombinasjon av Scrum og Kanban der vi tar i bruk det typiske Kanbanboardet med user stories.

4.3 Plan for statusmøter og beslutningspunkter i perioden

Hver sprint har en fastsatt lengde på to uker som går fra mandag til mandag. Før hver sprint holdes det et sprint-planleggingsmøte hvor vi kartlegger hvilke arbeidsoppgaver som legges i backloggen for den kommende sprinten.

Etter hver sprint holdes det et sprint-vurderingsmøte sammen med produkteier fra Evidi. Dette møtet vil være annenhver mandag 10:00-10:30. Her går vi over det som har blitt gjort i den gjeldene sprinten, samt hvilke ting som kan forbedres og legges til i produkt-backloggen.

Hver torsdag kl 13:00, utfører vi et møte sammen med veileder Rune, her går vi igjennom hva som har blitt gjort siden sist og får tilbakemeldinger. Annenhver torsdag kjører vi det retrospective møte samtidig der vi går over hva som fungerte bra i den gjeldene sprinten, samt hva som kan forbedres.

5 Organsering av kvalitetssikring

5.1 Dokumentasjon

Gjennom prosjektet, skal vi ha mye fokus på god dokumentasjon og lagring av systemet og ha en profesjonell systemutvikling. For å klare dette, skal vi følge relevante standarder for programmeringsspråkene og utviklingsverktøyene som inngår i prosjektet.

Dokumenter og lagring

- All prosjektkode ligger i sine respektive repositories.
- Hovedrapporten skrives i L^AT_EX via online-programmet Overleaf.
- Møtereferat skrives også i L^AT_EX via online-programmet Overleaf.
- Andre dokumenter som kontrakter ligger lagret på OneDrive.

5.2 Standarder

Prosjektet skal preges av gode utviklingsrutiner der det skal følges gitte standarder og konvensjoner for de relevante programmeringsspråkene.

- For backend skal Microsoft sin C# coding og naming convention følges¹ for å forsørge at all kode blir konsist skrevet.
- REST api'et skal følge OpenAPI spesifikasjonen for enklere automatisering av dokumentasjon².

5.3 Utviklingsrutiner

- Alle arbeidsoppgaver skal legges til som work-item i Azure DevOps.
- Hver work-item skal være så atomisk som mulig.
- All kode skal kommenteres før hver commit.
- Alle commit-meldinger skal være beskrivende og inneholde smart-tags (fix, fixes eller fixed) etterfulgt av #work-item.
- Atomisk committing av kode. Spesielt etter hver funksjon og klasse.
- Det skal jobbes branch-basert der vi har en main branch for produksjonsklar kode og en dev branch som vi lager nye feature-branches ut i fra. På denne måten har vi alltid fungerende kode i main, mens testing og nyutvikling skjer ut ifra dev.
- Når nye features er klare merges de tilbake til dev og om alt fungerer som det skal, lages det en *merge request* til main, som en eller flere av gruppelemene må godkjenne etter gjennomgang av endringene.

¹<https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>

²<https://oai.github.io/Documentation/start-here.html>

5.4 Verktøy

Tabell 1 inneholder en oversikt over utviklingsverktøyene gruppen skal bruke ved prosjektet.

Tabell 1: Utviklingsverktøy.

Verktøy	Beskrivelse	Bruksområde
Azure	Skyplattform for hosting av applikasjonen	Deployment
Azure DevOps	Samarbeidsverktøy og repo til kildekode	Prosjektstyring
Toggl	Verktøy for oversikt og føring av timer	Timeregistrering
Overleaf	Verktøy for rapport-skriving	Rapport
Teams	Kommunikasjonskanal mellom gruppemedlemmer, Evidi og veileder	Kommunikasjon
Lucidchart	Online verktøy for å tegne diagramer	Visualisering
Google sheet	Verktøy for å lage et Gantt diagram	Rapport
Visual Studio	IDE for utvikling	Utvikling
Visual Studio Code	Teksteditor for redigering av enkle filer, samt Bicep-support	Utvikling
Jetbrains Rider	IDE for utvikling	Utvikling

5.5 Plan for inspeksjoner og testing

For å fordele ressurser, har vi valgt å prioritere testene ved å teste de delene i systemet som har krevende logikk. Det er ikke vits å bruke ressurser på triviell logikk. Disse delene skal Unit-testes der testene blir skrevet fortløpende ved ny funksjonalitet. Det skal også kjøres integrasjonstester for å sjekke at API'et kommuniserer som ønsket med andre komponenter i systemet.

Når ny eller oppdatert funksjonalitet blir implementert, skal det gjennomføres en akseptens-test som tester den nye funksjonaliteten. Dette er for å forsikre at funksjonaliteten oppfører seg som forventet.

- Testverktøy for C# er integrert i .NET
- Testframeworket Jest³ vil bli brukt for JS/React

³<https://jestjs.io/>

5.6 Risikoanalyse på prosjektnivå

Figur 3 viser en oversikt over hvilke risikoer som kan oppstå under prosjektet. Den viser hvor stor sannsynlighet det er for at en hendelse oppstår og hvilke typer påvirkninger en slik hendelse vil ha for prosjektet. Figur 2 viser et overordnet blick over den *totale risikoen* en hendelse har på utviklingen til prosjektet. Denne figuren blir brukt sammen med figur 3 for å gi et enklere overordnet blick over hvilke av risikoene i figur 3 som burde ha størst prioritet.

Konsekvens/sannsynlighet	Meget liten sannsynligh	Liten sannsynlighet	Sannsynlig	Stor sannsynlighet	Ekstremt sannsynlig
Katastrofe					
Svært stor					
Middels					
Svært liten					
Liten					

Figur 2: Enkel risikotabell.

No.	Hovedkategorier	Risiko	Årsak	Sannsynlighet	Konsekvens	Risiko	Løsning
1	Bedrift	Gruppemedlemm er ikke istand til å utføre tildelt ukentlig oppgave	Sjukmeldt/skadet	3	2	Middels	Gruppemedlemmene skal ha ha samme kunnskap om alle delene av prosjektet, sånn at de "friske" medlemmene kan fortsette på oppgaven.
2	Produkt	Ikke fungerende kode blir pushet til produksjon	Kode blir direkte pushet til main	3	3	Middels	Arbeidet på prosjektet skal bli fordelt på branches, og hver merge request skal bli inspisert av gruppen. Ingen medlemmer skal få lov til å pushe direkte til main branchen.
3	Bedrift	Kildekode og prosjekt blir slettet	Data og informasjon blir slettet	2	3	Middels	Da all kildekode blir håndtert igjennom git vil hver gruppemedlem alltid ha en lokal versjon tilgjengelig på egen pc
4	Produkt	Misforstått oppgaven	Tolket oppgaven på en annen måte enn det klienten hadde tenkt for seg	2	3	Middels	Diskutere med eksterne veiledere tidlig i prosjektet, for å bedre overblikk over hva som er forventet fra oppgaven.
5	Bedrift	Prosjektet blir ikke ferdigstilt	Fokus på irrelevante funksjonaliteter, og	2	4	Middels	Lage en plan på hvilke funksjonaliteter har høyest prioritet, og hvem som er unødvendige å ha med.
6	Produkt	Kildekoden er ustrukturert og ubrukelig	Dårlig kommunikasjon mellom gruppemedlemmene	1	4	Liten	Planlegge, og ha jevnlig kommunikasjon og inspeksjon av kildekoden.
7	Bedrift	Mister kommunikasjonen med et av gruppemedlemmene, som øker arbeidsmengden	Medlemmet ignorerer alle former for kontakt	1	4	Liten	Se konsekvens på dette under "Rutiner og grupperegler"
9	Bedrift	Bryter taushetsplikts avatalen	Deler kildekode, og sensitiv informasjon over åpne plattformer	2	5	Middels	Bruke tildelte programmer for lagring av kildekode og sensitiv informasjon

Figur 3: Oversikt over risikoer og løsninger.

6 Plan for gjennomføring

6.1 Gantt diagram

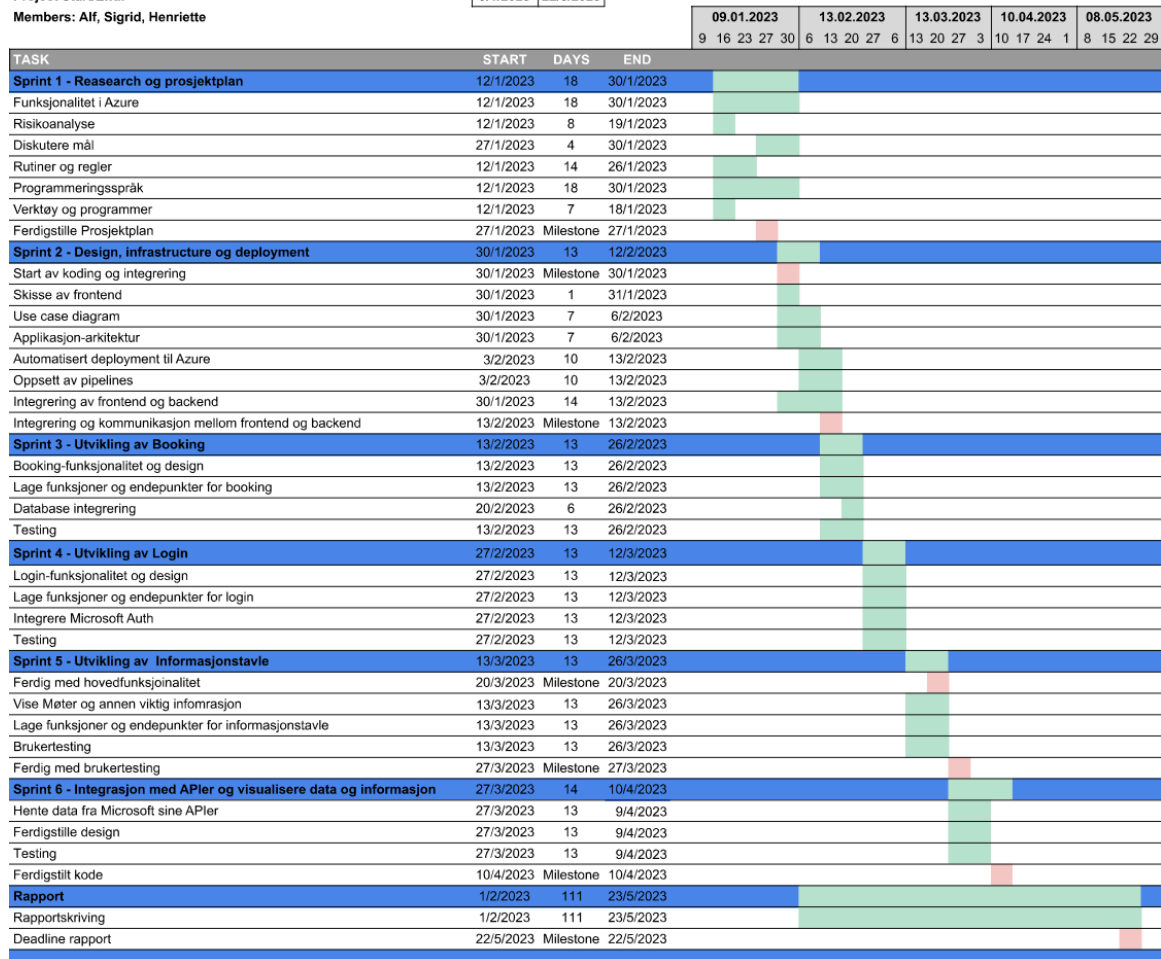
Starttider, deadlines, oppgaver og milepæler er noe gruppen må ha bestemt og ha kontroll på, for å kunne få en god arbeidsflyt gjennom prosjektfasen. Gantt diagrammet er brukt for at gruppen skal ha kontroll over valgte tidsfrister og holde oversikt over hvor langt hvert grupped medlem har kommet i prosjektprosessen.

Informasjonsplattform for ansatte

Project Start/End:

9/1/2023 22/5/2023

Members: Alf, Sigrid, Henriette



Figur 4: Gantt diagram.

6.2 Milepæler

Utover prosjektet, har vi definert en rekke milepæler. Dette gir oss en pekepinn på når vi ønsker å være ferdig med enkelt oppgaver. I løpet av prosjektet, kan vi sikre oss at vi følger planen.

- **Milepæl 1, 27.januar:** Prosjektplan ferdig og bestemme teknologier
 - Prosjektplanen er ferdig og oppsett.
 - Etter grundig research har vi bestemt oss for hvilke teknologier vi ønsker å bruke.
- **Milepæl 2, 30.januar:** Oppstart av koding
 - Begynne med den første funksjonaliteten.
- **Milepæl 3, 13.februar:** Integrasjon av backend og frontend, samt oppsett av CI/CD
 - Alle komponenter av prosjektet skal fungere sammen.
 - Oppsett av pipelines for testing og deployment til Azure.
- **Milepæl 4, 20.mars:** Ferdig med hovedfunksjonalitet
 - Fortsette med koding og finpusse funksjoner.
- **Milepæl 5, 27.mars:** Brukertesting ferdig
 - Brukertesting av systemet skal være ferdig.
- **Milepæl 6, 10.april:** Koding skal være ferdig og rapporten godt påbegynt
- **Milepæl 7, 22.mai:** Deadline rapport

Vedlegg B

Møtereferat

11.01.2023 - Lynkurs i prosjektstyring

Deltakere

- Tom Røise
- Alle gruppemedlemmer
- Andre bachelorstudenter

Mål

Komme i gang med bacheloroppgaven og prosjektplan.

Agenda

1. Karakteristika ved Bacheloroppgavearbeidet som vil prege arbeidet
2. Prosjektplanen - gjennomgang av mal
3. Valg av prosessrammeverk / systemutviklingsmodell
4. Råd og vink til prosjektstyring i Bacheloroppgaven

Oppgaver

Bli ferdig med prosjektplanen som skal levers før 1.februar.

12.01.2023 - Oppstartsmøte med Evidi i Oslo

Deltakere

- Fra Evidi:
 - Alle veilederne og daglig leder Jan Sverre Larsen
 - Våre veiledere:
 - Jørgen Mobakken
 - Felix Leenborg Anthonessen
 - Abdirahman Hassen
- Alle gruppemedlemmene
- Grupper fra andre skoler

Agenda

1. Lynkurs i Azure, Bicep, Resource Group, Function Apps
2. Kontrakt mellom oppdragsgiver, gruppemedlemmer og NTNU

3. Fastsette oppgaven og eventuelle spørsmål
4. Avtale nytt møtetidspunkt

Kontrakt

Arbeidsperioden er kontraktfestet til tidsrommet 12.01.2023 – 22.05.2023. Evidi får eierskap og bruksrett til alt vi produserer, mens rapporten kan være offentlig.

Nytt møtetidspunkt

Jørgen kaller inn til nytt møte der vi planlegger å sette opp sprinter for den første perioden.

13.01.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle deltakere

Agenda

- Spørsmål rundt prosjektplanen
- Avklare ukentlig møtetid

Spørsmål rundt prosjektplanen

Vi har valgt å skrive rapporten på norsk, så prosjektplanen blir også skrevet på norsk. Håper på å bli ferdig med rapporten i løpet av uke 4. Vi gir beskjed til Rune om spesifikke avsnitt han kan lese gjennom.

Avklare ukentlig møtetid

Møtetid blir torsdager kl 13:00 på Rune sitt kontor. Kan også møtes på Teams hvis nødvendig.

18.01.2023 - Sprintplanlegging med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Alle gruppe medlemmene

Agenda

- Sette opp sprinter for den første perioden
- Bestemme fast møtetidspunkt

Sette opp sprinter

En sprint går over to uke fra mandag til mandag. Den første sprinten inneholdet opp oppgaver som skal være ferdig til 30.01. Det meste er research og å prøve ut Azure sine teknologier.

Bestemme fast møtetidspunkt

Møtene blir ved slutten av hver sprint, altså annenhver mandag kl 09:30.

19.01.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle deltakere

Agenda

- Sprintplanning
- Prosjektplanen

Sprintplanning

Vi har fått hjelp av Evidi med å sette opp oppgaver for den først sprinten. Hver sprint går over to uker.

Prosjektplan

Vi har kommet godt i gang med prosjektplanen. Rune kommenterer i teksten etter hvert som han leser. Plan å levere i slutten av uke 4.

26.01.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle deltakere

Agenda

- Prosjektplanen

Prosjektplan

Vi er snart ferdige med prosjektplanen. Rune hadde noen kommentarer i teksten som vi skal se nærmere på før vi leverer.

30.01.2023 - Sprint review / planlegging med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppelemmene

Agenda

- Hvordan gikk sprint 1?
- Planlegge sprint 2

Hvordan gikk sprint 1?

Alle oppgavene var gjort, bortsett fra å sette opp eget repo siden vi ikke hadde tilgang til dette. Evidi skal se videre på dette, slik at vi kan få gjort dette så fort som mulig.

Planlegge sprint 2

I sprint 2 skal vi starte å se på design i frontend, se på infrastruktur og backend. Vi vil og få alt til å fungere sammen, slik at den første funksjonaliteten kan implementeres.

02.02.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle deltakere

Agenda

- Status på prosjektet

Status på prosjektet

Etter møtet med Evidi har vi satt opp oppgaver for neste sprint. Vi skal få alle deler til å være oppe og gå og begynne med den første funksjonaliteten.

06.02.2023 - Opplæring i Pipelines

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppemedlemmene

Agenda

- Gjennomgang av Pipelines

Gjennomgang av Pipelines

Vi fikk et kræsjskurs i pipelines som skulle hjelpe oss å komme i gang med infrastrukturen.

13.02.2023 - Retrospective / Sprint planning med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppemedlemmene

Agenda

- Retrospective
- Demo av prosjektet så langt
- Sprint Planning

Retrospective

I sprint 2 ble vi ferdige med alle oppgavene som var tildelt. Vi ble ferdige før sprinten var over, så vi startet på oppgaver som tilhørte sprint 3.

Demo av prosjektet så langt

Så langt har vi jobbet med med frontend, så det var dette vi viste fram. De synes at det så veldig bra ut.

Sprint Planning

I sprint 3 skal vi starte på funksjonalitetene på bookingsiden. Her skal både backend og frontend integreres. Det er ikke nødvendig å velge mellom frie tidspunkt, men man får valget mellom «Før lunsj», «Etter lunsj» og «Hele dagen».

16.02.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold

- Alle deltakere

Agenda

- Status på prosjektet
- Demo av prosjektet så langt

Status på prosjektet

Vi har hatt møte med Evidi og vist dem demoen. Rune ga oss et tips om å utnytte disse møtene enda mer ved å få dem til å tenke på hvilke funksjonaliteter de ser for seg.

Demo av prosjektet

Rune synes vi hadde kommet godt i gang med prosjektet. Neste del blir å få på plass funksjonaliteter.

23.02.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle deltakere

Agenda

- Status på prosjektet

Status på prosjektet

Vi har fått frontend og backend til å snakke med hverandre. Vi har fått til både GET- og POST-request. I neste sprint skal vi starte med databasen.

27.02.2023 - Retrospective / Sprint planning med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Alle gruppelemmene

Agenda

- Retrospective
- Demo av prosjektet så langt

Retrospective

I sprint 3 ble vi ferdige med alle oppgavene som var tildelt. Det var ganske bra beregna med antall oppgaver, så vi traff bra på fristen. Vi er fornøyde med hva vi har fått til. Videre kan vi være flinkere til å ha daglige møter.

Demo av prosjektet så langt

Vi har lagd nye funksjonaliteter i frontend på bookingsiden, etter inspirasjon fra Evidi.

02.03.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold

- Alf Magnus Nohr
- Henriette Eltvik

Agenda

- Status på prosjektet

Status på prosjektet

Vi har fått frontend og backend til å snakke med hverandre. Vi har fått til både GET- og POST-request. I neste sprint skal vi starte med databasen.

Diskusjon

- Skrive hvorfor vi har gjort det vi har gjort så langt
- Kan skrive tester som en form for debugging
- Få startet med testing, og finne ut hva som skal testes
- Viktigts å teste backend. Brukersnittet er gange simpelt så ikke så viktig å teste ennå
- Hvorfor valgte vi nosql i førsteomgang?
- Evt logge SQL for å se bruksområder
- script: sette opp skjema, sette inn metadata applikasjonen trenger, demo database (separere i egne .sql filer)
- Kan estimere user stories i stedet for enkle task
- Høre med Evidi om når ting skal oppdateres på frontend. (må være naturlig)
- Legge til chatGPT chat som vedlegg

28.02.23 - Lynkurs 2

Deltakere

- Frode Haug
- Sigrid
- Henriette
- Andre bachelorstudenter

Mål

Komme i gang med rapporten, og hvordan skrive en bra rapport.

Agenda

1. Gjennomgang av hoveddelene på rapporten.
2. Djupere gjennomgang av noen av underdelene som skal være i rapporten.
3. Diskusjon om plagiat; viktig å ta med alt som ikke er eget utviklet!

Oppgaver

Begynn tidlig på rapportskrivningen, og ta vare på kilder som er brukt i utviklingen av produktet.

13.03.2023 - Retrospective / Sprint planning med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen

- Alle gruppe-medlemmene

Agenda

- Retrospective
- Demo av prosjektet så langt

Retrospective

I sprint 4 har vi jobba mye med andre fag for å øve til eksamen, men vi er snart i mål med bookingsiden på prosjektet. Vi har sett på login, men ikke implementert det enda. I starten av neste uke skal vi ha et ekstra møte med Evidi for å komme i gang med login.

Demo av prosjektet så langt

Mulighet for å se egne bookinger og slette de.

16.03.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alf Nohr
- Sigrid Vestheim

Agenda

- Status på prosjektet
- Prosjektplanen

Status på prosjektet

Snart ferdig med bookingsiden. Mangler testing og grafisk fremstilling av ledige plasser.

Diskusjon

- Se på deadlines og produktive timer. Viktig å utnytte tiden for å rekke deadlines
- Begynn å se på testing
- Testing burde ha egne databaser og være separerte fra hverandre

27.03.2023 - Retrospective / Sprintplanlegging med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppe-medlemmene

Agenda

- Retrospective
- React og key vault

Retrospective

Så godt som ferdig med bookingsiden. Nå mangler vi å hente ut data fra API til teams.

React og key vault

Gikk gjennom problemene om hvordan man skal referere til key vault i React. Konkluderte med å se på serverkomponenter i Next.js

30.03.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle gruppemedlemmene

Agenda

- Status på prosjektet
- Prosjektplanen

Status på prosjektet

Demo av prosjektet. Testing ferdig. Hovedfunksjonalitet ferdig, unntatt bursdager. Starte på rapporten etter påska.

Diskusjon

- Siden alle prosjekter er forskjellige, må vi bruke forskjellige maler når vi skriver.
- Vi skal sifra til Rune om det er noe han skal lese over, gjerne små avsnitt.

11.04.2023 - Retrospective / Sprint planning med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppemedlemmene

Agenda

- Retrospective
- Demo av prosjektet så langt

Retrospective

Ferdig med bursdager og møter. Bytta over til Next.js. Refaktorering av prosjekt.

Demo av prosjektet så langt

De synes funksjonaliteten ser bra ut. Fint design. Mangler logg ut knapp. Får ikke tilgang til møter.

13.04.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle gruppemedlemmene

Agenda

- Status på prosjektet

- Rapport

Status på prosjektet

Mesteparten av tiden fremover skal brukes på rapport. Er godt i gang med kap 1, 2 og 3. Trenger tilbakemelding. Litt problemer med next.js. Litt problemer med pop-up-vindu.

Diskusjon

- Skal mål være med i rapporten? Ja
- Må fordele arbeid og sette opp tidsfrister for å ha en god flyt.
- Alle prosjekter er forskjellige. Viktig at rapporten gjenspeiler prosjektet.

17.04.2023 - Brukertesting med Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppelemmene

Agenda

- Gikk gjennom deployet applikasjon
- Svarte på brukertesting skjema.
- Observerte brukertesting på bruker.
- Fikk fikset tilgangstoken, møte funksjonalitet fungerer som forventet.

Status på prosjektet

Fokusert på rapportskrivning, og debugget kode.

24.04.2023 - Retrospective / Sprint planning med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppelemmene

Agenda

- Retrospective
- Access token - meetings

Retrospective

Fått deploya alt. Snart klart for brukertesting.

Access token - meetings

Får ikke henta ut riktig access token for møter. Tror at vi ikke har tilgang. Evidi skal se på det.

20.04.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle gruppemedlemmene

Agenda

- Status på prosjektet
- Rapport

Status på prosjektet

Har bare jobbet med rapport. Vi skal bruke overtidstimer på resterende koding.

Diskusjon

- Diskusjon rundt implementering
- Diskusjon rundt ingeniørfaglig innsikt

27.04.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle gruppemedlemmene

Agenda

- Status på prosjektet
- Rapport

Status på prosjektet

Har for det meste jobbet med rapport skriving.

Diskusjon

- Kommentarene på rapporten skal være *resolved* når de er ferdige.
- Noe av teksten på implementasjon kan gå over til design hvis det inneholder beskrivelse av selve designet.
- Mulighet for å skille mellom css og selve javascript koden. Det som kontrollerer det visuelle skal ikke være inkludert i javascript koden.
- Diskusjon angående bruk av konseptuell eller domene modell.
- Beskrive valg av arkitektur. Hvorfor har vi valgt å programmere på denne måten?

04.05.2023 - Veiledningsmøte med Rune

Deltakere

- Rune Hjelsvold
- Alle gruppemedlemmene

Agenda

- Rapport

Status på prosjektet

Har for det meste jobbet med rapport skriving.

Diskusjon

- Kapittel 7 og 8 er nesten klar for gjennomlesing.
- Eksterne folk skal lese over rapporten, for å sjekke helheten av den.
- Skal fjerne kodesnutter i rapporten og heller lage modeller. Begynt med avgrensing på rapport.
- Vedlegg skal bindest inn i rapporten.
- I diskusjon skal vi diskutere det vi har vist.
- NB! Ta med det viktigste som vi har arbeidet på.

08.05.2023 - Retrospective / Sprint planning med veiledere hos Evidi

Deltakere

- Jørgen Mobakken
- Abdirahman Hassen
- Felix Leenborg Anthonessen
- Alle gruppelemmene

Agenda

- Retrospective
- Sluttproduktet

Retrospective

Evidi har tatt brukertesting. De er godt fornøyde med produktet og innsatsen. Dette er siste møte, men om det er noe kan vi ta kontakt med de gjennom Teams.

Sluttproduktet

Evidi er godt fornøyde med sluttproduktet og kan godt tenke seg å videreutvikle dette.

Vedlegg C

Ukelogg

NAME	MON 01/09	TUE 01/10	WED 01/11	THU 01/12	FRI 01/13	SAT 01/14	SUN 01/15	TOTAL
Alf Magnus Nohr	—	—	6:00:00	4:00:00	2:07:00	—	—	12:07:00
Henriette Eltvik	—	—	2:00:00	4:00:00	0:37:00	—	1:00:00	7:37:00
Sigrid W Vestheim	—	3:30:00	6:00:00	4:00:00	2:07:00	—	—	15:37:00
TOTAL	—	3:30:00	14:00:00	12:00:00	4:51:00	—	1:00:00	35:21:00

Figur C.1: Timelogg uke 2.

NAME	MON 01/16	TUE 01/17	WED 01/18	THU 01/19	FRI 01/20	SAT 01/21	SUN 01/22	TOTAL
Alf Magnus Nohr	—	—	5:12:42	6:52:35	2:00:00	—	—	14:05:17
Henriette Eltvik	—	—	7:55:05	6:36:43	9:10:00	—	1:08:35	24:50:23
Sigrid W Vestheim	—	—	7:00:00	7:00:00	4:11:00	—	—	18:11:00
TOTAL	—	—	20:07:47	20:29:18	15:21:00	—	1:08:35	57:06:40

Figur C.2: Timelogg uke 3.

NAME	MON 01/23	TUE 01/24	WED 01/25	THU 01/26	FRI 01/27	SAT 01/28	SUN 01/29	TOTAL
Alf Magnus Nohr	2:00:00	4:00:00	4:00:00	6:00:00	2:00:00	—	—	18:00:00
Henriette Eltvik	8:00:00	8:00:00	5:30:00	4:00:00	—	—	2:00:00	27:30:00
Sigrid W Vestheim	2:05:00	1:00:00	6:32:25	8:00:00	2:37:00	—	3:30:00	23:44:25
TOTAL	12:05:00	13:00:00	16:02:25	18:00:00	4:37:00	—	5:30:00	69:14:25

Figur C.3: Timelogg uke 4.

NAME	MON 01/30	TUE 01/31	WED 02/01	THU 02/02	FRI 02/03	SAT 02/04	SUN 02/05	TOTAL
Alf Magnus Nohr	7:30:00	3:00:00	9:00:00	8:30:00	—	—	—	28:00:00
Henriette Eltvik	6:40:00	5:30:00	6:00:00	6:50:00	4:00:00	1:22:00	—	30:22:00
Sigrid W Vestheim	5:30:00	3:00:00	7:16:31	7:00:00	4:45:00	—	—	27:31:31
TOTAL	19:40:00	11:30:00	22:16:31	22:20:00	8:45:00	1:22:00	—	85:53:31

Figur C.4: Timelogg uke 5.

NAME	MON 02/06	TUE 02/07	WED 02/08	THU 02/09	FRI 02/10	SAT 02/11	SUN 02/12	TOTAL
Alf Magnus Nohr	7:15:16	9:00:00	5:58:00	8:15:00	3:33:00	—	—	34:01:16
Henriette Eltvik	6:15:00	4:12:00	6:27:00	7:00:00	3:25:00	—	—	27:19:00
Sigrid W Vestheim	4:45:00	4:00:00	7:02:00	5:45:00	1:00:00	—	—	22:32:00
TOTAL	18:15:16	17:12:00	19:27:00	21:00:00	7:58:00	—	—	83:52:16

Figur C.5: Timelogg uke 6.

NAME	MON 02/13	TUE 02/14	WED 02/15	THU 02/16	FRI 02/17	SAT 02/18	SUN 02/19	TOTAL
Alf Magnus Nohr	5:54:00	7:15:00	6:09:40	5:30:00	6:00:00	—	—	30:48:40
Henriette Eltvik	6:05:21	1:37:00	6:22:00	7:37:00	6:00:00	—	—	27:41:21
Sigrid W Vestheim	5:00:00	1:38:00	6:55:00	7:00:00	3:30:00	1:30:00	—	25:33:00
TOTAL	16:59:21	10:30:00	19:26:40	20:07:00	15:30:00	1:30:00	—	84:03:01

Figur C.6: Timelogg uke 7.

NAME	MON 02/20	TUE 02/21	WED 02/22	THU 02/23	FRI 02/24	SAT 02/25	SUN 02/26	TOTAL
Alf Magnus Nohr	6:39:00	6:45:00	5:29:08	4:25:00	9:09:00	—	—	32:27:08
Henriette Eltvik	2:30:00	4:00:00	6:30:00	4:30:00	5:00:00	2:00:00	—	24:30:00
Sigrid W Vestheim	5:00:00	—	8:00:00	7:00:00	6:00:00	—	1:45:00	27:45:00
TOTAL	14:09:00	10:45:00	19:59:08	15:55:00	20:09:00	2:00:00	1:45:00	84:42:08

Figur C.7: Timelogg uke 8.

NAME	MON 02/27	TUE 02/28	WED 03/01	THU 03/02	FRI 03/03	SAT 03/04	SUN 03/05	TOTAL
Alf Magnus Nohr	6:22:00	7:29:00	5:00:00	2:30:00	6:00:00	—	—	27:21:00
Henriette Eltvik	6:00:00	3:00:00	6:43:00	—	3:15:00	—	—	18:58:00
Sigrid W Vestheim	7:00:00	4:00:00	5:45:00	5:30:00	—	—	—	22:15:00
TOTAL	19:22:00	14:29:00	17:28:00	8:00:00	9:15:00	—	—	68:34:00

Figur C.8: Timelogg uke 9.

NAME	MON 03/06	TUE 03/07	WED 03/08	THU 03/09	FRI 03/10	SAT 03/11	SUN 03/12	TOTAL
Alf Magnus Nohr	—	—	—	4:55:00	6:30:00	—	—	11:25:00
Henriette Eltvik	—	—	—	3:00:00	4:36:00	3:00:00	6:46:00	17:22:00
Sigrid W Vestheim	—	—	—	7:00:00	7:00:00	—	1:15:00	15:15:00
TOTAL	—	—	—	14:55:00	18:06:00	3:00:00	8:01:00	44:02:00

Figur C.9: Timelogg uke 10.

NAME	MON 03/13	TUE 03/14	WED 03/15	THU 03/16	FRI 03/17	SAT 03/18	SUN 03/19	TOTAL
Alf Magnus Nohr	6:00:00	2:00:00	—	4:00:00	—	—	—	12:00:00
Henriette Eltvik	5:30:00	3:45:00	2:00:00	—	—	—	—	11:15:00
Sigrid W Vestheim	8:00:00	3:15:00	—	6:55:00	2:00:00	—	—	20:10:00
TOTAL	19:30:00	9:00:00	2:00:00	10:55:00	2:00:00	—	—	43:25:00

Figur C.10: Timelogg uke 11.

NAME	MON 03/20	TUE 03/21	WED 03/22	THU 03/23	FRI 03/24	SAT 03/25	SUN 03/26	TOTAL
Alf Magnus Nohr	—	5:44:13	7:00:00	5:45:00	7:00:00	—	—	25:29:13
Henriette Eltvik	4:24:00	7:15:00	7:47:07	7:03:00	2:45:00	—	—	29:14:07
Sigrid W Vestheim	—	7:00:00	7:00:00	8:00:00	5:00:00	—	2:45:00	29:45:00
TOTAL	4:24:00	19:59:13	21:47:07	20:48:00	14:45:00	—	2:45:00	84:28:20

Figur C.11: Timelogg uke 12.

NAME	MON 03/27	TUE 03/28	WED 03/29	THU 03/30	FRI 03/31	SAT 04/01	SUN 04/02	TOTAL
Alf Magnus Nohr	7:00:00	10:00:00	7:00:00	7:00:00	6:45:00	—	—	37:45:00
Henriette Eltvik	7:00:00	8:00:00	8:00:00	6:45:00	6:45:00	—	—	36:30:00
Sigrid W Vestheim	7:00:00	7:00:00	7:50:45	7:00:00	5:30:00	—	—	34:20:45
TOTAL	21:00:00	25:00:00	22:50:45	20:45:00	19:00:00	—	—	108:35:45

Figur C.12: Timelogg uke 13.

NAME	MON 04/03	TUE 04/04	WED 04/05	THU 04/06	FRI 04/07	SAT 04/08	SUN 04/09	TOTAL
Henriette Eltvik	3:27:00	4:00:00	2:45:00	—	—	—	—	10:12:00
Sigrid W Vestheim	—	2:00:00	—	—	—	—	—	2:00:00
TOTAL	3:27:00	6:00:00	2:45:00	—	—	—	—	12:12:00

Figur C.13: Timelogg uke 14.

NAME	MON 04/10	TUE 04/11	WED 04/12	THU 04/13	FRI 04/14	SAT 04/15	SUN 04/16	TOTAL
Alf Magnus Nohr	—	7:00:00	7:45:00	8:45:00	—	5:00:00	4:59:59	33:29:59
Henriette Eltvik	1:30:00	5:38:09	6:00:00	3:58:00	6:00:00	—	—	23:06:09
Sigrid W Vestheim	2:00:00	7:00:00	8:13:26	5:00:00	6:00:00	2:15:00	—	30:28:26
TOTAL	3:30:00	19:38:09	21:58:26	17:43:00	12:00:00	7:15:00	4:59:59	87:04:34

Figur C.14: Timelogg uke 15.

NAME	MON 04/17	TUE 04/18	WED 04/19	THU 04/20	FRI 04/21	SAT 04/22	SUN 04/23	TOTAL
Alf Magnus Nohr	10:00:00	10:30:00	9:00:00	8:59:59	9:00:00	—	—	47:29:59
Henriette Eltvik	7:00:00	5:30:00	8:00:00	7:00:00	5:14:00	—	4:05:00	36:49:00
Sigrid W Vestheim	7:00:00	7:00:00	7:50:45	7:55:44	5:44:00	—	5:10:00	40:40:29
TOTAL	24:00:00	23:00:00	24:50:45	23:55:43	19:58:00	—	9:15:00	124:59:28

Figur C.15: Timelogg uke 16.

NAME	MON 04/24	TUE 04/25	WED 04/26	THU 04/27	FRI 04/28	SAT 04/29	SUN 04/30	TOTAL
Alf Magnus Nohr	9:00:00	7:00:00	8:00:00	6:30:00	9:00:00	—	—	39:30:00
Henriette Eltvik	7:00:00	8:30:00	8:35:42	8:00:00	6:00:00	—	—	38:05:42
Sigrid W Vestheim	8:41:20	7:00:00	7:30:00	7:00:00	6:00:00	1:30:00	2:21:00	40:02:20
TOTAL	24:41:20	22:30:00	24:05:42	21:30:00	21:00:00	1:30:00	2:21:00	117:38:02

Figur C.16: Timelogg uke 17.

NAME	MON 05/01	TUE 05/02	WED 05/03	THU 05/04	FRI 05/05	SAT 05/06	SUN 05/07	TOTAL
Alf Magnus Nohr	9:15:00	8:15:00	8:00:00	8:00:00	6:00:00	—	—	39:30:00
Henriette Eltvik	9:10:00	8:00:00	8:00:00	8:00:00	7:00:00	3:00:00	6:30:00	49:40:00
Sigrid W Vestheim	7:22:00	7:00:00	7:45:00	8:00:00	7:30:00	—	—	37:37:00
TOTAL	25:47:00	23:15:00	23:45:00	24:00:00	20:30:00	3:00:00	6:30:00	126:47:00

Figur C.17: Timelogg uke 18.

NAME	MON 05/08	TUE 05/09	WED 05/10	THU 05/11	FRI 05/12	SAT 05/13	SUN 05/14	TOTAL
Alf Magnus Nohr	1:15:00	6:45:00	6:00:00	8:00:00	7:00:00	6:15:00	5:00:00	40:15:00
Henriette Eltvik	7:00:00	7:00:00	6:45:00	7:00:00	6:45:00	2:00:00	5:00:00	41:30:00
Sigrid W Vestheim	9:12:00	7:15:00	8:00:00	8:15:00	6:45:00	3:00:00	3:30:00	45:57:00
TOTAL	17:27:00	21:00:00	20:45:00	23:15:00	20:30:00	11:15:00	13:30:00	127:42:00

Figur C.18: Timelogg uke 19.

NAME	MON 05/15	TUE 05/16	WED 05/17	THU 05/18	FRI 05/19	SAT 05/20	SUN 05/21	TOTAL
Alf Magnus Nohr	7:45:00	7:30:00	2:00:00	10:45:00	12:00:00	7:00:00	—	47:00:00
Henriette Eltvik	7:00:00	5:15:00	2:00:00	10:00:00	12:40:00	6:25:00	—	43:20:00
Sigrid W Vestheim	8:00:00	7:45:00	2:08:12	11:00:00	12:00:00	7:00:00	—	47:53:12
TOTAL	22:45:00	20:30:00	6:08:12	31:45:00	36:40:00	20:25:00	—	138:13:12

Figur C.19: Timelogg uke 20.

Vedlegg D

Brukertest skjema

Ansattportal

Dette er et spørreskjema som skal ta for seg brukertesting av programmet "Ansattportal". Vi har lagt inn flere oppgaver. Når en oppgave er gjennomført, trykker du neste og så svarer du på spørsmålet til denne oppgaven.

Link til applikasjonen:

<https://happy-tree-0b3277603.3.azurestaticapps.net>

* indikerer at spørsmålet er obligatorisk

Jobbtittel

1. Hva er din jobbtittel?

Oppgave 1: Logg inn

2. Oppgave 1: Logg inn *

Markér bare én oval per rad

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Å logge inn var intuitivt	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 2: Legg til bursdag

Legg til bursdagen din.

3. Oppgave 2: Legg til bursdag

Markér bare én oval per rad

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å legge til bursdagen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 3: Hjemmesiden

Se på hjemmesiden og skaff deg oversikt.

4. Oppgave 3: Hjemmesiden *

Markér bare én oval per rad

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å se hvem som hadde bursdag i dag.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å få oversikt over møter / arrangement i dag.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 4: Booking

Book deg to plasser på kontoret, på to forskjellige datoer. Prøv å slette den ene. Prøv også å finne ut hvordan man kan se hvilke andre som har booket seg plass på kontoret for en gitt dato.

5. Oppgave 4: Booking *

Markér bare én oval per rad

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Å booke seg en plass var intuitivt.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var intuitivt å slette en booking.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Å se andre bookinger på en bestemt dato var intuitivt.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Å booke seg plasser på forskjellige datoer var intuitivt.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var en god oversikt over egne bookinger.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å se at det var fullt på kontoret.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 5: Logg ut

Logg deg ut av applikasjonen.

6. Oppgave 5: Logg ut

Markér bare én oval per rad

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å logge seg ut.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 6: Oppsummering

7. Oppgave 6: Oppsummering *

Markér bare én oval per rad

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å navigere seg gjennom nettsiden	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å finne informasjonen man trengte	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg kommer til å bruke denne daglig	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg synes dette er et godt hjelpemiddel	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg kommer til å reise mer på kontoret	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg synes designet var intuitivt	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

8. Hva likte du best med applikasjonen? *

9. Hva likte du minst med applikasjonen? *

10. Har du andre kommentarer til nettsiden?

Dette innholdet er ikke laget eller godkjent av Google.

Google Skjemaer

Vedlegg E

Brukertest resultat

Ansattportal

Dette er et spørreskjema som skal ta for seg brukertesting av programmet "Ansattportal". Vi har lagt inn flere oppgaver. Når en oppgave er gjennomført, trykker du neste og så svarer du på spørsmålet til denne oppgaven.

Link til applikasjonen:

<https://happy-tree-0b3277603.3.azurestaticapps.net>

Jobbtittel

Hva er din jobbtittel?

Utvikler

Oppgave 1: Logg inn

Oppgave 1: Logg inn *

Helt enig

Delvis enig

Nøytral

Delvis uenig

Helt uenig

Å logge inn var intuitivt



Oppgave 2: Legg til bursdag

Legg til bursdagen din.

Oppgave 2: Legg til bursdag

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å legge til bursdagen	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 3: Hjemmesiden

Se på hjemmesiden og skaff deg oversikt.

Oppgave 3: Hjemmesiden *

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å se hvem som hadde bursdag i dag.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å få oversikt over møter / arrangement i dag.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 4: Booking

Book deg to plasser på kontoret, på to forskjellige datoer. Prøv å slette den ene. Prøv også å finne ut hvordan man kan se hvilke andre som har booket seg plass på kontoret for en gitt dato.

Oppgave 4: Booking *

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Å booke seg en plass var intuitivt.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var intuitivt å slette en booking.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Å se andre bookinger på en bestemt dato var intuitivt.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Å booke seg plasser på forskjellige datoer var intuitivt.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var en god oversikt over egne bookinger.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å se at det var fullt på kontoret.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 5: Logg ut

Logg deg ut av applikasjonen.

Oppgave 5: Logg ut

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å logge seg ut.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 6: Oppsummering

Oppgave 6: Oppsummering *

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å navigere seg gjennom nettsiden	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å finne informasjonen man trengte	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg kommer til å bruke denne daglig	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg synes dette er et godt hjelpemiddel	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg kommer til å reise mer på kontoret	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg synes designet var intuitivt	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Hva likte du best med applikasjonen? *

Rent og ryddig design

Hva likte du minst med applikasjonen? *

Kunne vært litt tydeligere at det var fullbooket

Har du andre kommentarer til nettsiden?

Dette innholdet er ikke laget eller godkjent av Google.

Google Skjemaer

Ansattportal

Dette er et spørreskjema som skal ta for seg brukertesting av programmet "Ansattportal". Vi har lagt inn flere oppgaver. Når en oppgave er gjennomført, trykker du neste og så svarer du på spørsmålet til denne oppgaven.

Link til applikasjonen:

<https://happy-tree-0b3277603.3.azurestaticapps.net>

Jobbtittel

Hva er din jobbtittel?

Utvikler

Oppgave 1: Logg inn

Oppgave 1: Logg inn *

Helt enig

Delvis enig

Nøytral

Delvis uenig

Helt uenig

Å logge inn var intuitivt



Oppgave 2: Legg til bursdag

Legg til bursdagen din.

Oppgave 2: Legg til bursdag

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å legge til bursdagen	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 3: Hjemmesiden

Se på hjemmesiden og skaff deg oversikt.

Oppgave 3: Hjemmesiden *

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å se hvem som hadde bursdag i dag.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å få oversikt over møter / arrangement i dag.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 4: Booking

Book deg to plasser på kontoret, på to forskjellige datoer. Prøv å slette den ene. Prøv også å finne ut hvordan man kan se hvilke andre som har booket seg plass på kontoret for en gitt dato.

Oppgave 4: Booking *

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Å booke seg en plass var intuitivt.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var intuitivt å slette en booking.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Å se andre bookinger på en bestemt dato var intuitivt.	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Å booke seg plasser på forskjellige datoer var intuitivt.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var en god oversikt over egne bookinger.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å se at det var fullt på kontoret.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 5: Logg ut

Logg deg ut av applikasjonen.

Oppgave 5: Logg ut

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å logge seg ut.	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Oppgave 6: Oppsummering

Oppgave 6: Oppsummering *

	Helt enig	Delvis enig	Nøytral	Delvis uenig	Helt uenig
Det var enkelt å navigere seg gjennom nettsiden	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Det var enkelt å finne informasjonen man trengte	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg kommer til å bruke denne daglig	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg synes dette er et godt hjelpemiddel	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg kommer til å reise mer på kontoret	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jeg synes designet var intuitivt	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Hva likte du best med applikasjonen? *

Alt i alt en intuitiv nettside med noen kule features. Liker oversikten over møter, bursdager og eventuelle egne bookinger.

Hva likte du minst med applikasjonen? *

Litt forvirrende oversikt over bookinger for alle. Skulle gjerne sett en litt mer klar oversikt på hvem som har booka og eksakt når de har booka det.

Har du andre kommentarer til nettsiden?

Dette innholdet er ikke laget eller godkjent av Google.

Google Skjemaer

Vedlegg F

ChatGPT



How to make a popup happen once, and after submit it never appears again (even when reloading the page) in react

Figur F.1: Spørsmål localStorage



One way to achieve this behavior in a React application is to use the browser's local storage to store a flag indicating whether the popup has been submitted or not.

Here's an example implementation:

1. Create a state variable in your component to keep track of whether the popup should be displayed or not:

scss

Copy code

```
const [showPopup, setShowPopup] = useState(true);
```

1. In the component's `useEffect` hook, check if the flag is set in local storage, and update the state accordingly:

scss

Copy code

```
useEffect(() => {  
  const popupShown = localStorage.getItem('popupShown');  
  if (popupShown) {  
    setShowPopup(false);  
  }  
}, []);
```

1. When the user submits the popup, set the flag in local storage to indicate that it has been shown:

javascript

Copy code

```
const handleSubmit = () => {  
  localStorage.setItem('popupShown', true);  
  setShowPopup(false);  
};
```

Figur F.2: Chatgpt

1. Finally, conditionally render the popup based on the `showPopup` state variable:

```
arduino Copy code  
  
{showPopup && (  
  // Render the popup component here  
)}
```

With this implementation, the popup will only be shown the first time the user visits the page. After they submit the popup, it will never be shown again, even if they reload the page or visit it again later.

Figur F.3: Chatgpt

SI

in javascript, i want to split three word "Per Olsen Hansen" where Hansen in the lastNAme and Per Olsen is the first name



You can split the full name into an array of first, middle, and last names using the `split()` method in JavaScript. Here's an example:

```
javascript Copy code  
  
const fullName = "Per Olsen Hansen";  
const nameArray = fullName.split(" ");  
const firstName = nameArray[0] + " " + nameArray[1]; // "Per Olsen"  
const lastName = nameArray[2]; // "Hansen"
```

In this code, we first define the full name as a string, "Per Olsen Hansen". We then use the `split()` method to split the string into an array using the space character (" ") as the separator. This gives us an array `nameArray` with three elements: "Per", "Olsen", and "Hansen".

We then extract the first and last names from the `nameArray` using array indexing. The first name is the first two elements of the array concatenated with a space in between, while the last name is the third element of the array.

Figur F.4: Split name

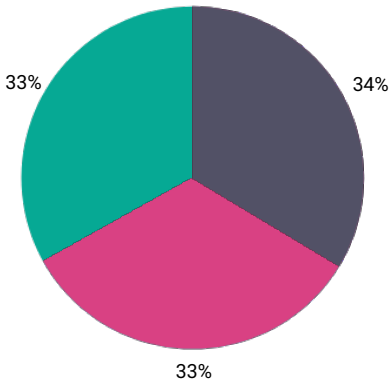
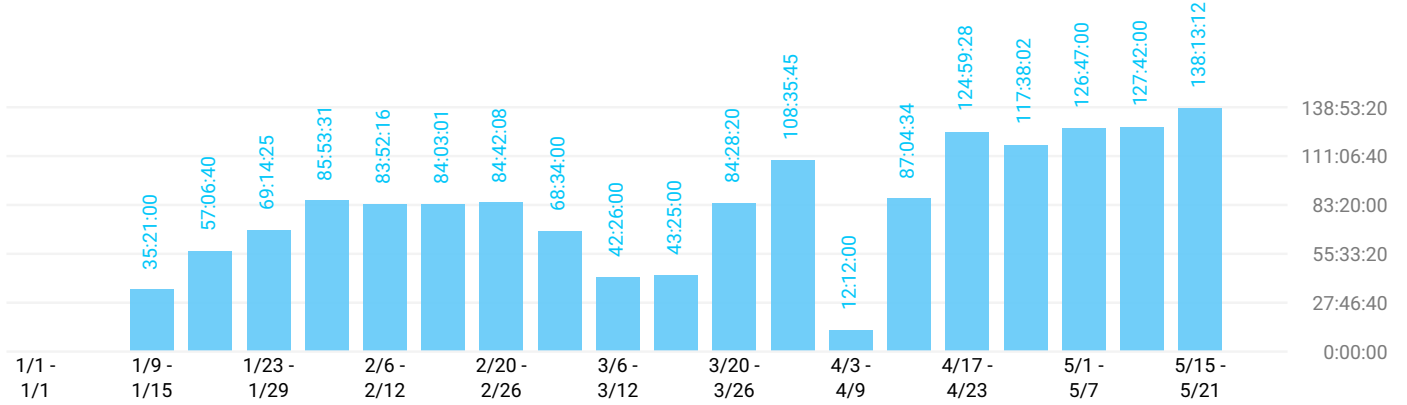
Vedlegg G

Timeføring

Summary Report

01/01/2023 – 05/22/2023

TOTAL HOURS: 1582:18:22

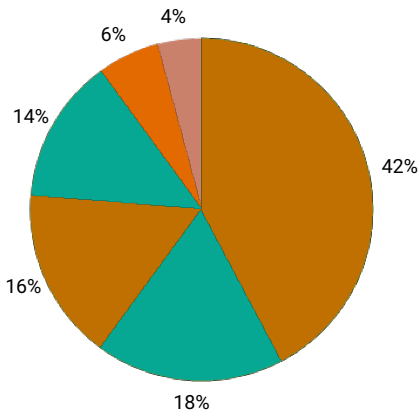


USER

- AM Alf Magnus Nohr
- SW Sigrid W Vestheim
- HE Henriette Eltvik

DURATION

- 530:44:32
- 527:18:08
- 524:15:42



TIME ENTRY

- Report
- Frontend
- Backend
- Research
- Infrastructure
- Meetings

DURATION

- 668:10:20
- 281:17:57
- 258:06:49
- 216:39:18
- 90:57:58
- 67:06:00

USER - TIME ENTRY	DURATION	PERCENTAGE
AM Alf Magnus Nohr	530:44:32	33.54%
Backend	78:25:08	4.96%
Frontend	86:58:15	5.5%
Infrastructure	90:57:58	5.75%
Meetings	22:17:00	1.41%
Report	182:56:11	11.56%
Research	69:10:00	4.37%
HE Henriette Eltvik	524:15:42	33.13%
Backend	147:31:41	9.32%
Frontend	28:40:00	1.81%
Meetings	20:52:00	1.32%
Report	241:53:08	15.29%
Research	85:18:53	5.39%
SW Sigrid W Vestheim	527:18:08	33.32%
Backend	32:10:00	2.03%
Frontend	165:39:42	10.47%
Meetings	23:57:00	1.51%

USER - TIME ENTRY	DURATION	PERCENTAGE
Report	243:21:01	15.38%
Research	62:10:25	3.93%

Vedlegg H

Brukerguide

How to run

To run the application you can choose if you want to run the backend and frontend locally.

1. Run backend:

- Clone the backend repository and enter the project folder
- Install Azure Functions Core Tools with `npm install -g azure-functions-core-tools`
- Start the functions with `func start`

2. Run frontend:

- Clone the frontend repository and enter the project folder
- Run `npm install` and then `npm run dev`
- You can also create a production build with `npm run build` and start it with `npm run start`

Further development and deployment

- New functionality should be developed under the develop branch.
- When a new functionality is ready to be deployed, you should create pull requests into the main branch to make the CI/CD pipeline handle the build and deployment.
- For suggestions on recommended improvements, read the project report.

Figur H.1: Brukerguide.

