Norconsult
Informasjonssystemer

Sluttrapport

Bachelorprosjekt i Anvendt Datateknologi 2021

Kontekstsensitive hjelp - Demo

OSLOMET

Gruppe 8

Jostein J. Hauge

Zakariya A. Ibrahim

Elin H. Hegsvold

Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo
Telefon: 22 45 32 00

PROSJEKT NR.
8

TILGJENGELIGHET
ÅPEN

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
Kontekstsensitiv hjelp - Demo	25.05.2021
	ANTALL SIDER / BILAG
	161
PROSJEKTDELTAKERE	INTERN VEILEDER
Jostein Jacobsen Hauge - s333943 Zakariya Abdinasir Ibrahim - s326058 Elin Holde Hegsvold - s333939	Andrea Arcuri

OPPDRAGSGIVER	KONTAKTPERSON
Norconsult Informasjonssystemer	Marita Ruud Gundersen

SAMMENDRAG

Bachelorprosjektet er gjennomført av tre studenter ved Oslomet - Storbyuniversitetet, på vegne av oppdragsgiver Norconsult informasjonssystemer.

Gruppen har utviklet en applikasjon med en kontekstsensitiv hjelp og et publiseringssystem. Hjelpen skal være enkel å finne, intuitiv å bruke og for å oppnå dette skal konteksten benyttes.

Rapporten inneholder prosess-, produkt-, og brukerdokumentasjon.

3 STIKKORD
Webapplikasjon, Vue, Tilgjengelighet

Forord

Denne rapporten er skrevet av tre studenter som en del av det avsluttende 3-årige bachelorstudiumet «Anvendt Datateknologi» ved OsloMet - Storbyuniversitet. Gruppen tok tidlig høsten 2020 kontakt med Norconsult Informasjonssystemer (NoIS) avd. Sandvika gjennom et åpent søknadsbrev. Til gruppens store glede ble brevet tatt godt imot og samarbeidet om å skape en bacheloroppgave alle kunne være stolte av begynte.

Utover høsten ble det gjennomført en rekke møter for å avklare oppgavens omfang. Vi fikk ganske frie tøyler under utformingen av oppgaven, noe som gav oss muligheten til å selv avgjøre hvor stort læringsutbytte oppgaven ville gi oss. Vi skal inn i en bransje som utvikler seg i rekordfart, og for å være best mulig forberedt ønsket vi å benytte muligheten til å lære oss mest mulig nytt. Juleferien ble benyttet til å lære nye teknologier vi skulle bruke i prosjektet. Med store ambisjoner og pågangsmot gikk vi i gang med oppgaven så fort kalenderen viste 2021, selv om vi ikke fikk kontakt med vår tildelte veileder fra skolen. Oppgaven bærer derfor noe preg av manglende veiledning fra skolens side i oppstartsfasen av prosjektet. Vi mener likevel at vi klarte å gjøre det beste ut av situasjonen og var raske til å omstille oss når situasjonen var blitt avklart.

I månedene som fulgte designet, utviklet og testet vi en webapplikasjon med en kontekstsensitiv hjelp og et publiseringssystem. Med unntak av én dag på kontoret til NoIS i Sandvika, har vi gjennomført hele prosjektet fra hjemmekontor, grunnet Covid-19. Det har gjort at vi har blitt godt kjent med verktøy som støtter denne måten å jobbe på, og kommunikasjonsevnen har fått god trening. Vi tror på ingen måte at det har hatt noen negativ konsekvens for prosjektet.

Vi vil benytte muligheten til å takke NoIS v/ Marita Ruud Gundersen og Cecilie Voss for muligheten til å gjennomføre bachelorprosjektet hos dere. Det har vært en veldig spennende oppgave som har gitt oss mange gode erfaringer vi skal ta med oss når vi etter sommeren møtes igjen.

Felles innholdsfortegnelse:

PRODUKTDOKUMENTASJON

BRUKERDOKUMENTASJON

PROSESSDOKUMENTASJON

Gruppe 8

OSLOMET - BACHELOR 2020 / 2021

Forord

Prosessdokumentasjonen har som formål å beskrive gjennomføringen av bachelorprosjektet. Først presenteres gruppens medlemmer, veiledere, samt oppdragsgiver. Her vil samtidig oppgavens fundament og problemstilling introduseres. Videre presenteres planleggingsfasen, kravspesifikasjonen, utviklingsprosessen og refleksjoner rundt utfordringer vi møtte underveis i prosessen. Rapporten er optimalisert for å leses på PC, og gir en god innsikt i hvordan applikasjonen har vært utviklet.

Innhold

1. Innledning

1.1 Gruppemedlemmene

Vi er tre studenter fra OsloMet som er i ferd med å avslutte en bachelorgrad i Anvendt Datateknologi. Vi møttes under fadderuka og har jobbet tett sammen gjennom hele studiet. Dette gjør at vi har de beste forutsetninger for et godt samarbeid gjennom bachelorprosjektet.

Jostein J. Hauge

s333943@oslomet.no

+47 988 59 269

Zakariya A. Ibrahim

s326058@oslomet.no

+47 484 31 387

Elin H. Hegsvold

s333939@oslomet.no

+ 47 974 07 320

1.2 Oppdragsgiver

Norconsult Informasjonssystemer AS (NoIS)
post@nois.no
+47 675 71 500
Kjørboveien 16, 1337 Sandvika

«NoIS utvikler, markedsfører og leverer helhetlige IKT-løsninger for prosjektering, bygging og forvaltning av infrastruktur og eiendom. Selskapet har også betydelig virksomhet knyttet til leveranse av konsulenttjenester innen IT-prosjektledelse, IT-rådgivning, IT-arkitektur, systemutvikling, systemintegrasjon, RPA, maskinlæring og AI innen alle bransjer. I tillegg har selskapet dyptgående kompetanse innenfor informasjonssikkerhet og cyber security, med egne konsulenter som jobber dedikert innenfor fagområdet.» (Norconsult Informasjonssystemer, u.å.) NoIS har omkring 290 ansatte fordelt på kontorer flere steder i landet. Størst er kontoret i Trondheim, og på en god andreplass er kontoret i Sandvika.

1.2.1 Veileder fra NoIS

Marita Ruud Gundersen
marita.ruud.gundersen@norconsult.com
+47 454 01 553
Gruppeleder

Marita har lang bakgrunn som systemutvikler, og har erfaring med ulike roller innen systemutviklingsoppdrag. Hun har jobbet som utvikler, scrum master, prosjektleder og produkteier. Nå er hennes rolle i NoIS personalansvarlig gruppeleder for en gruppe ved navn «Systemutvikling og Integrasjon» i avdeling Fundator.

Marita har vært gruppens primære veileder og har blant annet hatt en naturlig rolle under utvikling og revidering av kravspesifikasjonen. Hun har også bistått med å sette oss i kontakt med ressurser hos NoIS når det har vært behov for faglig bistand.

1.2.2 Kontaktperson fra NoIS

Cecilie Voss
Cecilie.Voss@norconsult.com
+47 980 367 72
Seniorrådgiver

Cecilie har over 25 års erfaring med visuell kommunikasjon og design, med sterkt fokus på formidling og brukeropplevelse, både i digitale flater og i fysiske omgivelser. Hun er leder for faggruppen UX Design i NoIS som har som formål å fremme brukerfokus, designtenking og god brukskvalitet. I tillegg kontinuerlig bidra med å utvikle gode verktøy for systemutviklerne i NoIS.

Cecilie har vært en viktig ressurs i forbindelse med design og utforming av applikasjonen. Under forprosjektet bidro hun med sin innsikt i Oslo REG, og organiserte møter med kontaktpersonene der slik at vi fikk muligheten til å få innspill fra disse underveis i prosessen.

1.3 Veileder fra OsloMet

Andrea Arcuri
Andrea.Arcuri@oslomet.no
+47 67 23 71 90
Professor

Andrea har mer enn 14 års erfaring innen forskning på og utvikling av teknologier for testing av programvare. Av den grunn har han vært en viktig ressurs under planlegging og gjennomføring av testing av applikasjonen. Han har ellers stilt opp når vi har hatt behov for å diskutere oppgaven og løsninger på problemer som har dukket opp underveis.

1.4 Kort om prosjektet

I løpet av prosjektet utviklet gruppen en kontekstsensitiv hjelp (heretter CSH - *Context-sensitive Help*) på oppdrag fra Norconsult Informasjonssystemer (NoIS). CSH er en betegnelse på hjelp som er rettet mot en enkelt tilstand eller funksjon i en programvare. Denne skiller seg fra generell hjelp som er rettet mot programvaren i sin helhet. CSH kan tilbys i ulike former, og bør tilpasses den enkelte kontekst.

Vår CSH er en egen modul som gir brukeren informasjon om den siden hen befinner seg på i en applikasjon. I tillegg tilbyr den blant annet en søkefunksjon og en FAQ dersom hjelpen i CSH'en ikke er tilstrekkelig. For at det skal være enklest mulig å skreddersy innholdet til CSH'en, samt oppdatere ved behov, ble det laget et publiseringssystem (heretter CMS - *Content Management System*). For å vise hvordan CSH'en og CMS'en kan fungere ble det i tillegg laget en dummy-applikasjon som komponentene ble bygget på.

Målet med prosjektet var å vise hvordan en kontekstsensitiv hjelp kan gjøre hjelpen mer tilgjengelig for brukere av et system. Hjelpen skal være enkel å finne, intuitiv å bruke og for å oppnå målsettingen skal konteksten benyttes. Samtidig var det potensielt store læringsutbyttet en viktig motivasjon som gruppens medlemmer brukte for å utfordre seg selv underveis. Les mer om prosjektets mål og rammebetingelser under kapittel 1.6.

1.5 Bakgrunnen for prosjektet

NoIS har sett behovet for en kontekstsensitiv hjelp i flere av systemene de har levert, men på grunn av lite motivasjon blant kundene har det aldri blitt implementert. NoIS så derfor på bachelorprosjektet som en mulighet for å utvikle en demoversjon av en CSH, for bruk i innsalgsprosessen av fremtidige systemer. På lengre sikt hadde det samtidig vært ønskelig om produktet kunne bygges videre på og bli en av NoIS sine hyllevarer.

For å gi demoen en mest mulig realistisk kontekst var det ønskelig å tilpasse denne til en kunde av NoIS. Høsten 2020 og vinteren 2021 gjennomførte NoIS et forprosjekt

for Renovasjons- og gjenvinningsetaten i Oslo (heretter Oslo REG) sitt nye system, og også her var CSH et tema. Det ble derfor bestemt at produktet skulle tilpasses Oslo REG sitt interne system *WebDEB*. Dagens system ble tatt i bruk i 2002, og er i liten grad oppdatert siden. Etaten har omtrent 720 ansatte med mange ulike bakgrunner og teknologiske forutsetninger, noe som gjør at behovet for veiledning i systemet er stort. Et nytt og moderne system vil naturligvis være enklere å bruke, men behovet for veiledning vil alltid være til stede.

Brukerundersøkelsene NoIS gjennomførte i forbindelse med forprosjektet avdekket at brukerne i liten grad benytter seg av de hjelpefunksjonene som er tilgjengelige i systemet. Brukerne har tilgang til en tradisjonell form for brukermanual i PDF-format, som de synes er vanskelig å navigere seg frem i. I tillegg har systemet en form for kontekstsensitiv hjelp, som er både godt skjult og i liten grad oppdatert. Sistnevnte ble ikke avdekket under brukerundersøkelsene, men under første møte gruppen hadde med Oslo REG. Det stilles spørsmålstegn ved hvorvidt brukerne av systemet er klar over at hjelpefunksjonen eksisterer eller om den ikke tilbyr brukerne den hjelpen de trenger. Brukerundersøkelsene avdekket at brukerne som regel spør en kollega når de trenger hjelp, da det som oftest er en raskere løsning på problemet deres.

1.6 Mål og rammebetingelser

Hovedmålsettingen med prosjektet er å vise hvordan en CSH kan gjøre hjelpen mer tilgjengelig for brukere av et system. Hjelpen skal ikke bare være enkel å finne, den skal også være intuitiv å bruke. I dette legger vi at brukerne umiddelbart skal kunne forstå hva de har fremfor seg, uavhengig av tidligere kunnskaper og erfaringer.

1.6.1 Effektmål

NoIS ønsker seg en demo de kan bruke under innsalg av CSH til fremtidige prosjekter. Demoen må kunne demonstrere nytteverdien av en CSH, hvordan den kan tilby variert innhold og enkelt tilpasses de ulike behovene til forskjellige systemer. Det er samtidig et mål at demoen kan brukes som grunnmur når en CSH skal implementeres i et nytt system. På den måten kan det redusere tiden og kostnaden knyttet til implementering, noe som igjen kan gjøre terskelen mindre for kunden. Til syvende og sist er målet at

demoen kan bidra til at flere ser behovet for en CSH og at NoIS dermed lykkes i å selge flere slike løsninger.

1.6.2 Resultatmål

Resultatmålene er i stor grad knyttet til kravene i kravspesifikasjonen (se kap. 2.4). Et viktig punkt her vil likevel være at CSH'en skal bruke konteksten i størst mulig grad. Kravet «CSH'en skal gi brukeren informasjon om innholdet i siden hen befinner seg» er et eksempel der konteksten blir brukt. Andre bruksområder kan for eksempel være å bruke informasjonen om brukeren (gjennom profilen) og siden hen befinner seg på til å forhåndsutfylle feilrapporteringer o.l. På den måten vil man gjøre det enklere å rapportere feil samtidig som man kvalitetssikrer informasjonen.

1.6.3 Læringsmål

Gruppemedlemmenes motivasjon med prosjektet var læringsutbyttet knyttet til å jobbe på et virkelighetsnært prosjekt for oppdragsgiver. Målet var å få erfaringer med hvordan utviklingsprosessen fungerer i praksis, og på den måten bli bedre forberedt på arbeidslivet. For oppnåelse var det essensielt at vi ikke var redde for å utfordre seg selv og lære nye teknologier.

1.6.4 Rammebetingelser

For å muliggjøre videre bruk av produktet utover demonstrasjonsformål, var det viktig at vi så etter løsninger som gjør det enklest mulig for NoIS å videreutvikle og gjøre tilpasninger til andre applikasjoner. Det innebar blant annet at vi brukte de samme verktøy og programmeringsspråk som NoIS bruker i dag. Det var også viktig at koden var godt strukturert og dokumentert, slik at det blir enklest mulig for en annen utvikler å sette seg inn i det arbeidet vi har gjort.

For å oppnå prosjektets hovedmålsetting innenfor den tiden som var til rådighet, ble det bestemt at fokuset skulle være på klientsiden av applikasjonen. Dersom produktet i fremtiden skal brukes utover demonstrasjonsformål, vil det være naturlig å benytte serversiden til systemet CSH'en skal integreres i.

Selv om CSH'en skal fungere som en generell demonstrasjon, vil den i denne oppgaven tilpasses Oslo REG. Siden WebDEB (Oslo REG sitt nåværende system) er utdatert og nytt system ikke er på plass, vil CSH'en implementeres i en dummy-applikasjon. Når det kommer til design, må vi forholde oss til Oslo kommune sine retningslinjer for bruk av font, farger og logo (Oslo Kommune, u.å.).

2. Planlegging og metode

2.1 Forberedelser

I løpet av høsten 2020 ble det bestemt at vi skulle bruke JavaScript biblioteket Vue.js (se kap. 2.6.1). Dette bygget på at NoIS brukte Vue.js og at vi ønsket å lære oss noe nytt i forbindelse med bachelorprosjektet. Vi hadde tidligere jobbet noe med JavaScript og biblioteket AngularJS, som blir noe av det samme. Vue.js har dog en helt annen struktur og mange nyttige verktøy som vi måtte sette oss inn i før utviklingsprosessen startet. Arbeidet startet allerede desember 2020, da vi fant et omtrent 30 timers komplett online kurs på plattformen Udemy (Schwarzmuller, 2020).

2.2 Forprosjekt

4. Januar 2021 startet forprosjektet, som innebar undersøkelser omkring CSH, utforming, ønsket funksjonalitet og implementering. Det ble gjennomført en rekke møter med NoIS for å få innsikt i WebDEB og hvordan Oslo REG bruker systemet, samt oversikt over ulike behov knyttet til CSH'en. På bakgrunn av dette ble det utarbeidet en kravspesifikasjon for funksjonelle og ikke-funksjonelle krav, samt fastsatt enkelte rammebetingelser. Det ble bestemt hvilke teknologier og verktøy som skulle brukes for å oppnå de fastsatte målene, og hvilken arbeidsmetodikk som skulle følges gjennom utviklingens gang. For å kunne måle fremdriften til prosjektet ble det utarbeidet en fremdriftsplan.

Ettersom produktet skulle tilpasses Oslo REG hadde det vært ideelt å gjennomføre brukerobservasjoner og intervjuer. Grunnet Covid-19 og det at Oslo REG ikke var oppdragsgiver gjorde det vanskelig å gjennomføre. I stedet gjennomførte vi flere møter

med NoIS for å kartlegge Oslo REG sine behov, det også fordi NoIS har gjort et grundig arbeid i forbindelse med deres forprosjekt.

2.3 Prosessmodell

Gjennom undersøkelser tilknyttet forprosjektet ble det identifisert flere faktorer som utelukket de plandrevne prosessmodellene. Hverken vi eller oppdragsgiver hadde tidligere erfaring med gjennomføring av bachelorprosjekter. Når vi heller ikke fikk kontakt med sin tildelte veileder fra skolen i løpet av forprosjektet, ble det vanskelig å beregne omfanget av oppgaven. Det ble også ganske tidlig klart at det var begrenset med informasjon om det valgte programmeringsspråket Vue 3 tilgjengelig på nett, fordi det ble lansert i september 2020 (Ulrich & Massigner, 2021).

En smidig utviklingsprosess tilbyr fleksibilitet i forhold til kravspesifikasjonen. Det er åpent for å tilføye krav etter hvert som de dukker opp, samtidig som at ustabile krav ikke hindrer progresjonen (Sommerville, 2018, s. 87). Dermed sto det mellom Scrum og Kanban. Det ble brukt et *Kanban Board* under implementeringen for å lettere holde oversikt over oppgavene. Med unntak av Kanban Boardet ble metodikkene til Scrum valgt fremfor Kanban. *Timeboxing* skaper en rytme i utviklingen og gjør det enklere å planlegge møter og andre aktiviteter med oppdragsgiver, enn om man bruker Kanban sin *taskboxing*. Videre var det et ønske om å benytte seg av fordelene knyttet til daglige møter og *sprint reviews*.

2.3.1 Product backlog

I Scrum er *product backlog* en prioritert liste over funksjonalitetene som skal implementeres (Sommerville, 2018, s. 85). Med utgangspunkt i kravspesifikasjonen gikk vi i gang med å finne ut hvilke krav som var viktigst for oppdragsgiver og hvilke som var avhengig av andre krav, for å finne ut hva som burde prioriteres. Kravene ble delt opp i mindre og mer spesifikke funksjonaliteter før de ble lagt inn i backloggen. Her ble liste-applikasjonen Trello (se kap. 2.6.9) brukt for å enkelt organisere backloggen. De ulike punktene ble videre fordelt i sprinter.

2.3.2 Sprinter

I Scrum er det vanlig at sprintene varer 2-4 uker (Sommerville, 2018, s. 85). Kortere sprinter gjør det enklere å gjøre endringer og oppdage problemer raskere (Levison, 2019). Det vil også gjøre samarbeidet med oppdragsgiver tettere ettersom vi er forpliktet til å møtes oftere. Sprint-lengden falt derfor på 2 uker og med en gjennomføringsfase på 12 uker, vil arbeidet blir fordelt på 6 sprinter.

2.3.3 Møter

2.3.3.1 Daglige stand-up

For hver dag det ble jobbet på prosjektet skulle det først avholdes et *stand-up* møte, der vi diskuterte det som hadde blitt gjort siden sist, hva som var planen videre, samt eventuelle hindringer. Møtene skulle ifølge prosessmodellen holdes så korte som mulig. Det var åpent for å holde noe lengre møter av hensyn til det faktum at vi ble nødt til å jobbe mye selvstendig grunnet Covid-19.

2.3.3.2 Sprint Review

I slutten av hver sprint ble det holdt et *sprint review* møte. Møtene ble den primære dialogen mellom oss og oppdragsgiver, hvor følgende sto på agendaen:

- Gjennomgang: Vi fortalte hva som hadde blitt gjort og hva som eventuelt ikke hadde blitt gjort. Om det hadde vært noen utfordringer som måtte løses eller lignende.
- Fremvisning av systemet: Vi viste frem det som hadde blitt gjort, samt hvordan dette interagerte med resten av systemet. Oppdragsgiver fikk mulighet til å komme med innspill og tilbakemeldinger til både den implementerte funksjonaliteten og designet.
- Videre plan: Vi fortalte hva som var planen for neste sprint. Om det var noen endringer i planen skulle det legges frem og diskuteres.
- Eventuelle spørsmål: Dersom det var andre ting som dukket opp i forkant av møte. Dette kunne være alt fra spørsmål om krav, design, teknisk eller organisatorisk.

Det ble tatt notater under møte for å lage et møtereferat. Dette for å få med oss alt som ble gjennomgått og for å huske eventuelle detaljer som var viktig for videre utvikling.

2.4 Kravspesifikasjon

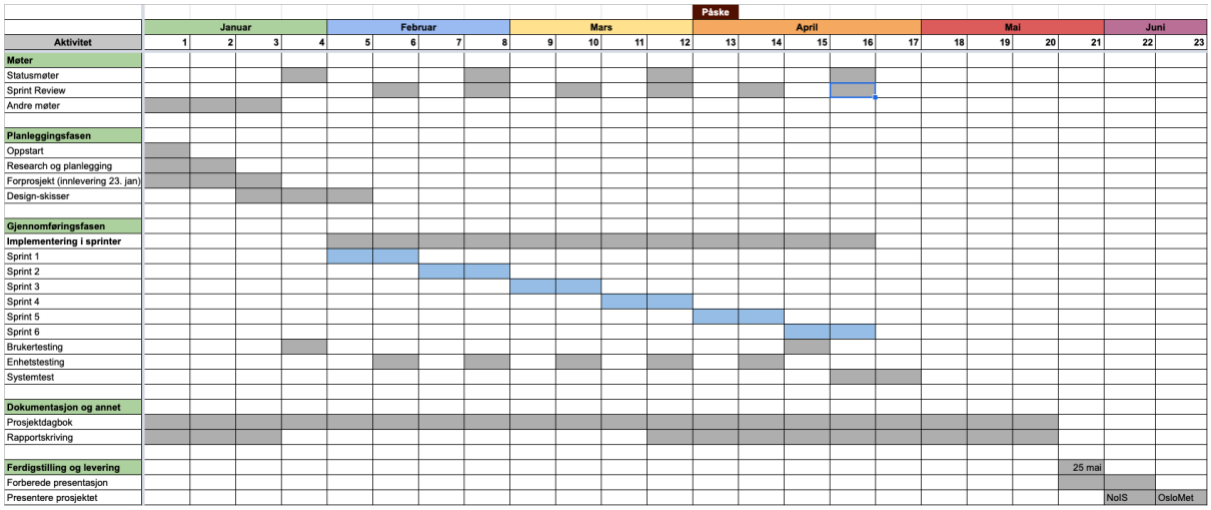
Kravspesifikasjonen ble utarbeidet helt i startfasen av prosjektet i samarbeid med oppdragsgiver. Kravspesifikasjonen ble primært utviklet for å danne et oversiktlig bilde av hva sluttproduktet skulle inneholde. Den utgjør sin nytte ved at alle parter er inneforstått med kravene til systemet, for å sikre at det ikke blir uenighet gjennom utviklingsprosessen. Kravspesifikasjonen har også sin hensikt i å gi en rask innføring i systemet, for å gi leseren oversikt over produktets funksjonaliteter.

Grunnet usikkerhet knyttet til at vi ikke har funnet lignende produkter på markedet, har kravspesifikasjonen vært åpen for endring under utviklingens gang. De ulike kravene ble derfor inndelt i kategoriene funksjonelle krav, ikke-funksjonelle krav og opsjonelle krav. Funksjonelle krav er en beskrivelse av hvordan systemet skal oppføre seg. For eksempel hva applikasjonen skal gjøre og ikke gjøre, tilgjengelighet, brukervennlighet og sikkerhet fra brukerens ståsted. Ikke-funksjonelle krav er rammene for systemet. For eksempel responstid, oppetid, ressursforbruk, antall samtidige brukere og lignende. Det ble i tillegg utviklet en liste over opsjonelle krav. Listen ble utviklet for å ha utvidet krav og funksjonalitet, dersom vi hadde tid til overs etter implementasjon av de funksjonelle og ikke-funksjonelle kravene.

Kravspesifikasjonen kan leses i sin helhet under kapittel 2 i Produktdokumentasjonen.

2.5 Fremdriftsplan

Fremdriftsplanen er utformet som et Gantt diagram (se figur. 1). Diagrammet gav en oversikt over hvilke deler av prosjektet som skal jobbes med når, og hvordan vi ligger an underveis i utviklingen. Det gjorde det også enklere å planlegge ulike aktiviteter i forhold til hverandre. For eksempel hadde utviklings-sprintene mye å si for hvordan vi la til rette for møter med oppdragsgiver.



Figur 1 - Fremdriftsplan

2.6 Teknologi og verktøy valg

Alle verktøy og programmeringsspråk ble valgt basert på hva vi ønsket å benytte, med innspill fra oppdragsgiver i henhold til hva de bruker i en daglig praksis. Det ble gjort både for å skape et optimalt samarbeid, men også for å gi oss et innblikk i hva en bedrift bruker av teknologier og verktøy. Valg av verktøy ble gjort basert på at alle skulle ha tilgang og kunne jobbe samtidig. Det ble også valgt verktøy for å støtte en smidig utviklingsmodell.

2.6.1 Vue.js med TypeScript

Vue.js er et JavaScript rammeverk utviklet for å være lett å lære. Med Vue kan man lage det meste innen web-grensesnittet, inkludert såkalte *single-page applications* (SPA) (Vue, u.å). Vue ble valgt på bakgrunn av at NoIS bruker det og at vi ønsket å lære noe nytt. Vue 3, som er siste versjon av rammeverket, ble lansert i september 2020 og var dermed relativt nytt når det ble bestemt at det skulle brukes. NoIS hadde ikke begynt å bruke det enda, men vi så ikke poenget med å lære oss en eldre versjon av rammeverket og Vue 3 ble dermed valgt.

Selv om Vue er et JavaScript rammeverk har det også innebygd støtte for TypeScript (Vue, u.å). Med strengere type deklarering er TypeScript mer forutsigbar og enklere å

lese og debugge, noe som gjør det mer egnet der teamet ikke sitter å jobber samlet (Sharma, 2018). Valget falt derfor på TypeScript fremfor JavaScript.

2.6.2 SCSS

For styling av applikasjonen ble det valgt å bruke SCSS, som er en utvidelse av klassisk CSS. Med SCSS har man blant annet mulighet til å definere variabler og på den måten jobbe mer objektorientert (Sass, u.å). Vi har fra tidligere god erfaring med både CSS og det å jobbe objektorientert, derfor så vi på SCSS som en fin måte å utfordre oss på.

2.6.3 Visual Studio Code

Visual Studio Code (heretter VSCode) er en gratis kildekode editor (IDE) som støtter de aller fleste kjente programmeringsspråk, inkludert Vue.js med både JavaScript og TypeScript. I 2019 ble den også rangert som den mest populære editoren i StackOverflow sin årlige undersøkelse (StackOverflow, 2019). Gjennom studiet har vi brukt en rekke ulike editorer. I Vue-kurset vi tok i forkant av prosjektet ble VSCode brukt, og det var derfor naturlig at editoren ble brukt videre gjennom prosjektet.

2.6.4 Microsoft Azure

Azure er en skyplattform for infrastruktur og administrerte tjenester som virtuelle servere med nettverk, fillagring og databaser (Bigelow, 2020). Azure er en plattform som NoIS ønsker integrert i sin teknologi stack etter hvert som de går over til skybaserte løsninger, og det var derfor en fordel om bachelorprosjektet kunne benytte seg av Azure sine tjenester.

2.6.5 Firebase

Firebase er en tjeneste som tilbyr en rekke funksjoner, som blant annet skytjenester, autentisering og database (Npm, u.å). Vi valgte å benytte Firebase til lagring av bilder og videoer, da vi hadde kjennskap til det fra Vue-kurset vi tok i forkant av prosjektet.

2.6.6 GitHub

Vi brukte GitHub som versjonskontroll. GitHub gjorde det mulig for oss å jobbe på kodebasen samtidig. Den ga god oversikt over hvem som hadde gjort hva og når det

ble gjort. Vi ønsket å bruke *branches* for å ha bedre kontroll under utviklingen av vår applikasjon. Her knytter vi inn tankegangen fra CI, det at *master-branch* bare blir benyttet til ferdig sjekket ut kode. All implementering/utvikling skjedde på egne *branches*.

2.6.7 GitKraken

Vi brukte git-klienten GitKraken for å enklere se flyten i og mot programvarelageret. GitKraken er et GUI som fremviser historikken, og alle operasjons-muligheter som «Pull», «Push» m.m. I tillegg får man enkelt oversikt over alle endringer som er gjort før de eventuelt skal pushes inn i systemet. Det tilbys også mulighet til å lage egne *brancher* for ulike funksjoner ved applikasjonen.

2.6.8 Google drive

Vi brukte Google Drive til regneark, rapporter, dagbok og diverse notater. Med Google Drive fikk alle på gruppen mulighet til å skrive på dokumenter til enhver tid og samtidig.

2.6.9 Trello

Trello er et kanban-board som ble brukt til å organisere prosjektet i et tavle-format. Trello gjorde det enklere å strukturere tildeling av oppgaver, da vi enkelt kunne se hva som ble jobbet på, hva som var ferdig og hva som manglet å gjøre. Oppdragsgiver fikk også mulighet til å følge med på progresjon underveis i utviklingen.

2.6.10 Microsoft Teams

Teams er et samarbeidsverktøy som gjør det enkelt å organisere og holde møter, og chatte (Microsoft, u.å). Teams ble brukt for kommunikasjon mellom oss i gruppen, og mellom gruppen og oppdragsgiver. Daglige møter og sprint reviews ble gjennomført over Teams.

2.6.11 Adobe XD

Adobe XD er et vektorbasert designverktøy for web- og mobilapplikasjoner (Purdila, u.å). Verktøyet ble brukt til å designe brukergrensesnittet, samt lage klikkbare prototyper av applikasjonen. Valget falt på dette verktøyet etter anbefaling fra Cecilie (se kap. 1.2.2).

3. Utviklingsprosessen

3.1 Kort om utviklingsprosessen

I denne delen av rapporten er det tatt for seg prosjektets utviklingsprosess som hadde sin startsdato 01.02.2021 og varte frem til 02.05.2021. Det skildres om ulike betraktninger, valg og løsninger som er foretatt gjennom denne fasen, og til slutt refleksjoner omkring hvilke utfordringer vi har møtt på underveis.

3.2 Om prosjektdagboka og dens funksjon i prosessen

Under hele prosjektet ble det skrevet en prosjektdagbok for å få oversikt over hva som ble gjort til enhver tid. Terskelen for å skrive i dokumentet skulle holdes lav, slik at dokumentet skulle bli benyttet. Dagboken var fin å ha dersom vi ikke jobbet samtidig, og den fikk en spesielt stor verdi når vi skulle skrive sluttrapporten. Da kunne vi gå tilbake å se hvordan vi løste problemene som oppsto underveis.

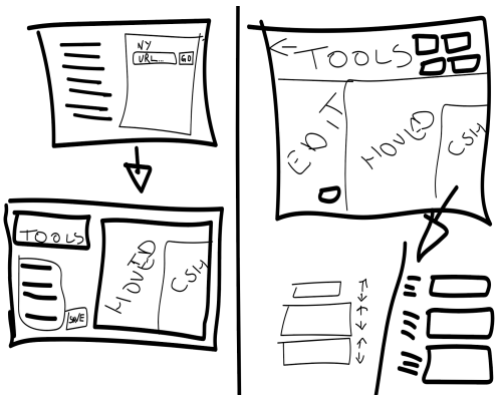
Dagboken var et felles dokument på Google Drive, slik at alle gruppens medlemmer alltid hadde tilgang. For hver dag gruppen arbeidet med prosjekt ble følgende ført inn:

- Dato (mm.dd.yyyy) som tittel
- Hva som må utføres (ToDo-liste)
- Arbeid som ble utført og evt. ikke utført
- Diverse problemer/utfordringer
- Møtereferater
- Evt. mål

3.3 Design

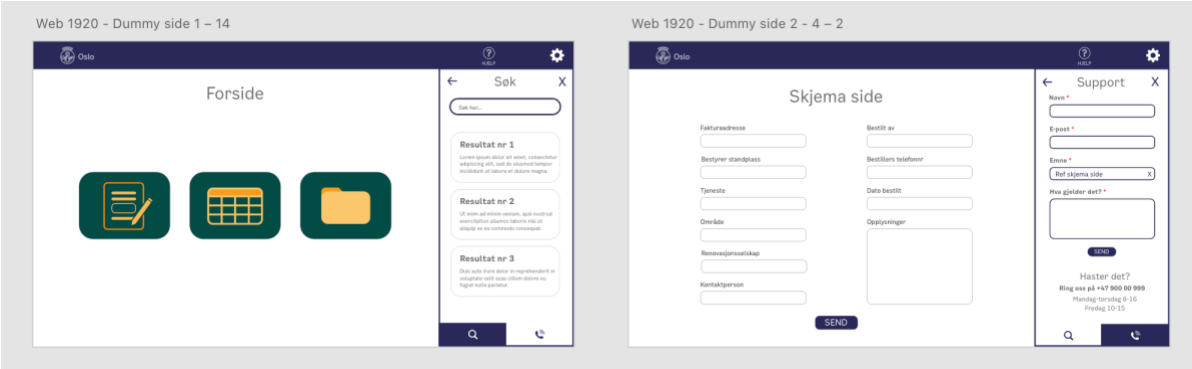
Utviklingen av applikasjonens design startet under forprosjektet, og fortsatte ut i den første sprinten. Det ble opprettet et prosjekt i Adobe XD (se kap. 2.6.11) der sidene i applikasjonen fikk hvert sitt *artboard* etter hvert som de ulike delene av applikasjonen ble kartlagt. Når alle applikasjonens undersider var på plass, startet arbeidet med å utforske forskjellige måter å skape et gjennomført design.

Det ble gjennomført flere runder med idémyldring der det ble laget raske skisser etter hvert som ideene dukket opp (se figur 2). Når vi hadde landet på et design, ble det laget realistiske skisser i Adobe XD. Skissene ble benyttet til å lage en klikkbar prototype, som ble vist til oppdragsgiver og Oslo REG. Se figur 3 for et utvalg skisser fra prototypen, eller klikk på lenken under for å se og teste prototypen i sin helhet.



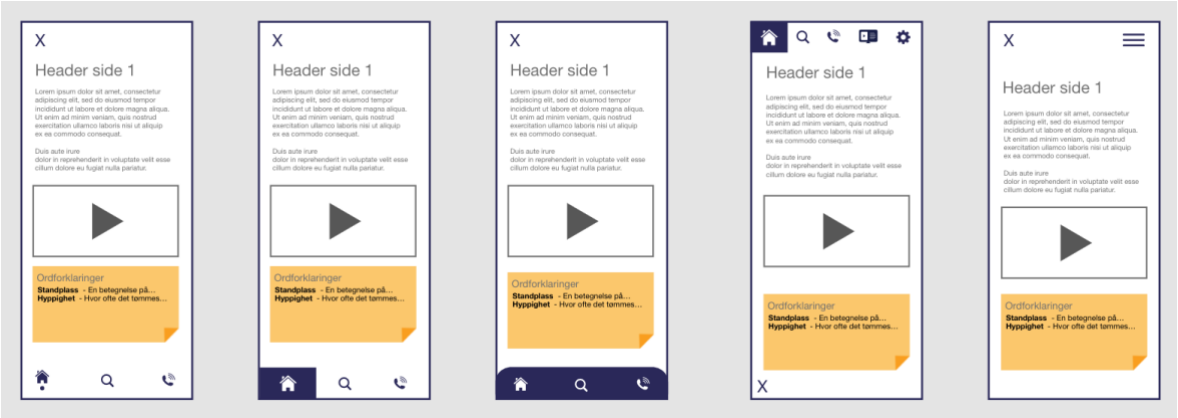
Figur 2 - En rask skisse av CMS

[Klikkbar prototype]



Figur 3 - Et utvalg skisser fra den klikkbare prototypen

Hvis vi skal trekke frem en spesifikk problemstilling i designprosessen som det ble brukt mye tid på, var det utformingen av navigasjonsbaren. Figur 4 viser ulike utforminger som ble utforsket på et tidlig stadium av prosessen. Tekstlige knapper ble tidlig utelukket ettersom det ville blitt problematisk i det smale formatet til CSH'en. Eneste løsningen dersom tekstlige knapper skulle vært brukt, var om disse lå i en nedtrekksmeny (se versjonen helt til høyre i figuren). En nedtrekksmeny ville skjult valgmulighetene, og vi konkluderte med at synlige valgmuligheter ville øke sannsynligheten for at en bruker velger å fortsette jakten på hjelp.



Figur 4 - Utforsket ulike måter å designe navigasjonsbar i CSH

Videre falt det seg mest naturlig å plassere navigasjonen til de ulike sidene i CSH'en på bunn. Hvis navigasjonen skulle vært på toppen, hadde lukke-knappen måtte vært på bunn. Det ville vært ulogisk for en bruker med erfaring, ettersom knappen som regel er plassert i toppen av et vindu. Valget falt til slutt på utformingen på bilde nr to fra venstre, da vi følte den viser hvilken side brukeren befinner seg på tydeligst.

3.3.1 WCAG

Under utviklingen av designet var det et stort fokus på universell utforming. Vi ønsket å lage en applikasjon som kunne være tilgjengelig for alle, uavhengig av funksjonsgraden til brukeren. Retningslinjene i Web Content Accessibility Guidelines (WCAG) ble derfor kartlagt tidlig i prosessen. Det ble laget et dokument med stikkord på punktene som var relevante for vår applikasjon, og beskrivelser av hvordan punktene kunne tilfredsstilles.

Dokumentet ble brukt som et oppslagsverk under utviklingsprosessen. For eksempel ble kontrastraten mellom farge på tekst og bakgrunnsfarge sjekket etter hvert som det ble brukt farger i applikasjonen. Under *code review* ble det sjekket at koden var semantisk strukturert, som vil si at den var bygget opp på en logisk måte som samsvarer med virkeligheten. På den måten ble applikasjonen testet for tilgjengelighet kontinuerlig gjennom utviklingen.

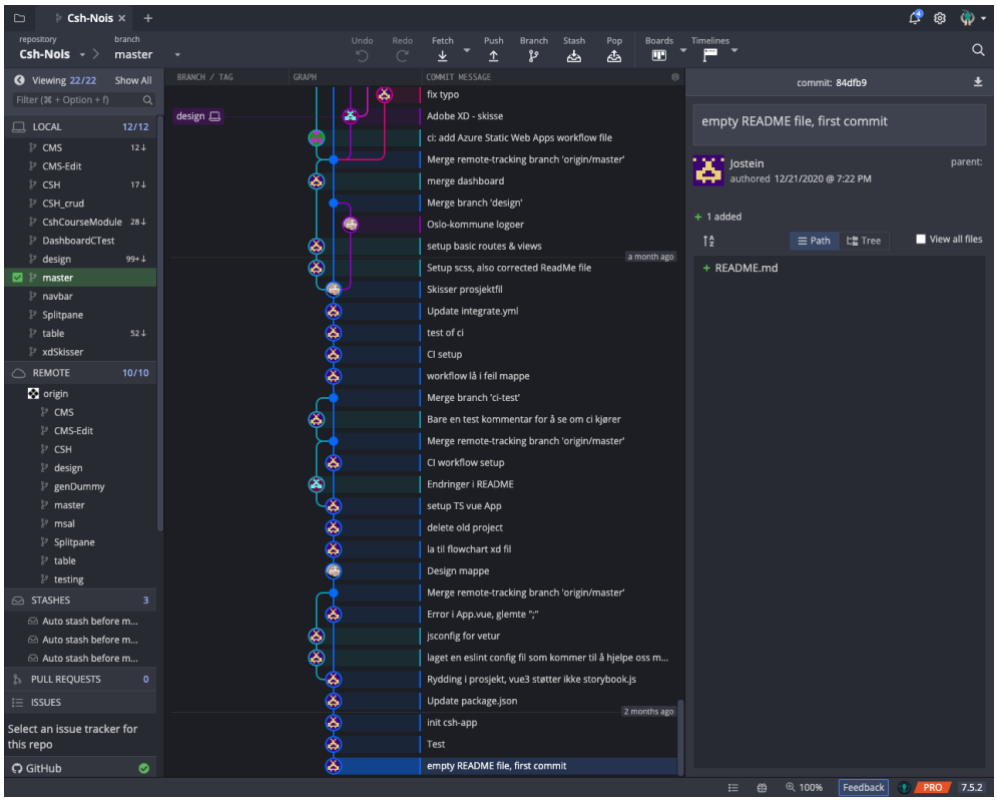
Applikasjonen ble også testet mot retningslinjene til WCAG helt i slutten av utviklingsprosessen. Les mer om planlegging og gjennomføring av testene i kapittel 7 i Produktdokumentasjonen.

3.4 Oppsett av prosessverktøy

3.4.1 Git

Programvarelageret (repository) ble satt opp av NoIS avdeling Fundator. Dette for at det skulle være enklere for NoIS å videreutvikle produktet etter bachelorprosjektet var over.

Oppsettet benyttet seg av en *master-branch* som ikke kodes på, men at det lages egne *brancher* som hentes ut fra en felles *master-branch*. I egne *brancher* jobbes det med de forskjellige oppgavene med applikasjonen. Når de er ferdig, pushes de til master. På denne måten inneholder master alltid siste versjon av programmet. I figur 5 under er det vist et utklipp av hvordan GUI'et i valgt klient viser *commit* historien.

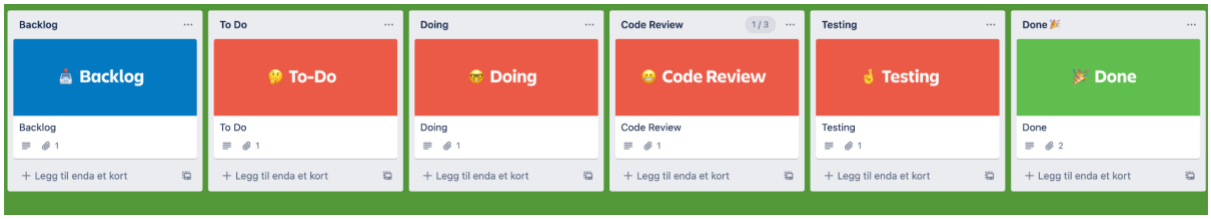


Figur 5 - Gitkraken

Fordelen med å jobbe i egne *brancher* og deretter *push* til *master-branchen* var at det støttet vår *CD* (continuous deployment) operasjon. *CD* operasjonen benyttet en Github Actions fil laget av Azure, for å hente ut siste versjon av applikasjonen fra *master-branchen,* og publisere den til Azure Static Web Apps. Da var det en fordel at funksjonaliteter man ikke ønsket å publisere enda, kunne pushes til andre *brancher* i mellomtiden. I tillegg støttet jobbing i *brancher* vår *CI-operasjon*. *CI* (Continous Integration) ble benyttet for å kjøre alle enhetstester skrevet til applikasjonen når *master-branchen* ble oppdatert.

3.4.2 Trello

Gjennom utviklingsprosessen ble det brukt en Trello tavle, for å holde oversikt over alle funksjonalitetene som skulle implementeres. Tavlen ble inndelt i lister for hver del av prosessen: *Backlog, to-do, doing, code review, testing* og *done* (se figur 6). Alle funksjonalitetene som skulle implementeres ble lagt inn i *backlog'en*.



Figur 6 - Trello-board

I starten av hver sprint ble funksjonalitetene som skulle implementeres i sprinten flyttet fra *backlog'en* til *to-do*. Når et av gruppemedlemmene begynte på en funksjonalitet ble den flyttet over i *doing*. Når funksjonaliteten var ferdig implementert ble den flyttet til *code review*, der en av de andre gruppemedlemmene gikk gjennom koden og sjekket at alt så greit ut. I *code review* listen var det ikke lov å legge inn fler enn tre funksjonaliteter, for å unngå at den ble en flaskehals. Når *code review* var gjort, ble det skrevet enhetstester til funksjonaliteten. Dersom feil ble oppdaget, ble feilen rettet før funksjonaliteten ble flyttet til *done*.

3.5 Sprinter

3.5.1 Sprint 1 - Oppsett & dummy-applikasjon (Uke 5-6)

Innhold

- Oppsett (prosjekt, system arkitektur, database osv)
- Implementere dummy applikasjon

Milepæler

- Prosjektet kan kjøres lokalt
- Få kontroll på system arkitekturen
- Ferdigstille dummy applikasjonen

Starten av sprint 1 var preget av usikkerhet rundt system arkitekturen. Det var i løpet av forprosjektet planlagt å sette opp en *microservice* arkitektur. Når vi i uke 4 endelig fikk kontakt med den interne veilederen vår, ble vi gjort oppmerksomme på at vi hadde lagt opp til mye mer arbeid enn hva som var realistisk å få til på den tiden som var til rådighet. Det ble bestemt at vi i stedet skulle gå for en serverløs *backend*, slik at vi kunne fokusere på *frontend* delen av applikasjonen. Det gjorde at vi i starten av sprint 1 måtte gjøre en del undersøkelser rundt hvordan Azure Funksjoner kunne brukes i kombinasjon med Vue.js.

I løpet av sprinten ble det avholdt en rekke møter med NoIS. System arkitekturen ble avklart med bedriftens veileder, og vi ble satt i kontakt med et par av NoIS sine utviklere som hadde erfaring med Azure. De gav oss tips om ulike tjenester som Azure tilbyr for den type arkitektur vi nå hadde landet på. Vi fikk også en gjennomgang av hvordan NoIS setter opp prosjekter i forhold til mappestruktur, programmeringsspråk, eksterne biblioteker osv. Det for å gjøre videreutvikling av applikasjonen enklere for NoIS, ettersom det var et av prosjektets rammebetingelser.

Når prosjektet var ferdig satt opp, startet implementering av dummy-applikasjonen. Dummy'en ble første prioritet ettersom de andre delene av applikasjonen (CSH og CMS) skulle bygges på denne delen. I ettertid har vi også sett fordelen av å starte med relativt enkel funksjonalitet for å bli kjent med språk og teknologi. Dummy-applikasjonen skulle bestå av 3 enkle sider: Forside, skjema-side og tabell-side. Etter

hvert så vi også behovet for en «404 ikke funnet» side, som brukeren kunne bli navigert til dersom hen forsøkte å besøke en ugyldig URL. Videre ble det bestemt at det ikke skulle legges ned mer arbeid enn nødvendig i dummy-applikasjonen, ettersom dens formål kun var å gi kontekst til CSH'en.

3.5.2 Sprint 2 - CSH & CMS grunnmur (Uke 7-8)

Innhold

- CSH layout (sider, skjemaer og navigasjonsbar)
- CMS layout (skjema for å legge inn innhold)
- Innlogging (autentisering) i CMS

Milepæl

- De ulike komponentene fungerer hver for seg

Etter at usikkerheten knyttet til systemarkitekturen var kartlagt og løst, var hovedfokuset i denne sprinten å få layout og navigasjon i CMS'en og CSH'en på plass. Det ble gjennomført et møte med NoIS og Oslo REG, der vi viste frem den klikkbare prototypen. Under møtet ble vi oppmerksomme på varierende behov for brukerstøtte. For å demonstrere de ulike mulighetene for brukerstøtte i en CSH, ble det bestemt at den skulle inneholde en FAQ-side med en underside for rapportering av feil.

Prototypen inneholdt et brukerstøtte-ikon i form av en telefon. Ettersom ikonet indikerte aktiv hjelp til brukere, ble det søkt etter et ikon som bedre kunne indikere typen hjelp en bruker nå kunne få ved trykk. Valget falt til slutt på et lyspære-ikon selv om vi ikke helt følte det var innlysende (se figur 7).

Figur 7 - Gammelt og nytt ikon

For at ikke alle brukere skal kunne gjøre endringer i CMS, skal det kreves innlogging for å få tilgang. Innloggings-funksjonaliteten ble prioritert, da den var et krav til applikasjonen. Ettersom Azure ble benyttet i andre deler av *backend*, ble det bestemt at autentisering også skulle gjøres ved hjelp av Azure.

3.5.3 Sprint 3 - Kommunikasjon & database (Uke 9-10)

Innhold

- Kommunikasjon mellom komponentene (Dummy, CSH og CMS)
- Lagring til og henting fra databasen
- Funksjonalitet til CSH-forside
    - Multimedia-støtte

Milepæl

- Innhold lagt inn i CMS dukker opp i CSH når man er på tilsvarende side i dummy-app

Etter at grunnmuren til CSH'en og CMS'en ble implementert i forrige sprint, var kommunikasjonen mellom komponentene i hovedfokus i denne sprinten. Det ble jobbet med å få på plass lagring av data fra CMS til databasen. Deretter ble det jobbet med å hente data fra databasen og til CSH'en sin forside. Dataene skulle hentes basert på siden bruker befinner seg på i dummy-applikasjonen, når hen åpner CSH.

Vi så på en løsning for lagring av multimedia i Azure, men grunnet dårlig dokumentasjon valgte vi å bruke Firebase. Valget var basert på tidligere kjennskap til Firebase. For visning av multimedia ble det søkt etter eksterne biblioteker. Det viste seg å være vanskelig å finne, på grunn av manglende kompatibilitet med Vue 3. Vi ble nødt til å bruke det standard video-elementet fra *HTML 5*, selv om det gav svært begrensede muligheter for tilpasninger.

3.5.4 Sprint 4 - Brukerstøtte, søk & varslinger (Uke 11-12)

Innhold

- Funksjonalitet til CSH-brukerstøtte
- Funksjonalitet til CSH-søk
- Varslinger

Milepæl

- Alle CSH'ens sider skal ha tiltenkt funksjonalitet implementert
- Forklarende varslinger om brukers handlinger

I den første delen av sprinten var fokuset på implementering av funksjonalitet i CSH'en. Funksjonalitetene som ble implementert var en søkefunksjon og en FAQ (*Ofte stilte spørsmål*), samt en feilrapportering-side. Når søkefunksjonen skulle lages, ble det diskutert hva den skulle kunne søke gjennom. Valgene det stod mellom var: Tilgjengelig hjelp i CSH eller dummy-applikasjonens innhold. Ettersom søkefunksjonen er plassert i CSH'en fant vi det mest naturlig at den søkte gjennom tilgjengelig hjelp i CSH.

I andre del av sprinten var fokuset på generelt design, responsivitet med tanke på nettbrett og tilgjengelighet for personer med nedsatt funksjonsevne. Det ble også implementert varslinger gjennom bruken av *toast*. Toastene er en fin måte og gi brukeren tilbakemelding på forskjellige handlinger. Tilbakemeldingene bestod av en ledende tekst og ulike fargesymboler for å indikere hva det gjelder.

3.5.5 Sprint 5 - PDF export & Kursmodul (Uke 13-14)

Innhold

- Implementering av opsjonelle krav
    - Kurs-modul
    - Pdf-eksportering
- Diverse finjusteringer

Milepæl

- Ferdigstille applikasjonen

I sprint 5 var hovedfokuset til gruppen å implementere utvalgte funksjonaliteter fra opsjons listen. Valgte opsjoner ble kurs-modul og eksportering av hjelp til PDF-dokument. Stegene i kurs-modulen ble utviklet med tanken om at brukerne skulle oppfordres til å interagere med applikasjonen.

Implementering av PDF-eksportering innebar en god del undersøkelser. Eksportering av de tekstlige delene av CSH'en og formateringen av disse var uproblematisk, mens bildene slet vi mye med før vi fant ut at det kunne gjøres ganske enkelt. Mer om utfordringen kan leses under kapittel 3.6.1.

Under forrige *sprint-review* fikk vi tilbakemelding fra oppdragsgiver om at det burde implementeres funksjonalitet for å hente opp igjen siste søk. Dersom man for eksempel trykket på et søkeresultat, byttet CSH'en til forsiden, søkeresultatet ble dermed nullstilt og brukeren måtte søke på nytt. På bakgrunn av tilbakemeldingene ble det derfor implementert en «hent siste søk» knapp.

Under statusmøtet med oppdragsgiver ble vi gjort oppmerksomme på at innholdet på CSH'ens forside burde gjøres mer relevant for å bedre demonstrere bruken av konteksten. Hittil var det kun lagt inn fylltekst og tilfeldige bilder vi hadde på maskinene våre for å programmere det grafiske. Det ble laget videoer som demonstrerte de ulike sidene, forklarende tekster og bilde av database-designet. Innholdet ble organisert slik at vi får vist variasjonen av mulig innhold på best mulig måte, noe som bygger på tanken om at produktet skal brukes til demonstrasjonsformål.

På sprint-review i slutten av sprinten viste vi frem kurs-modulen og eksportering til PDF. Kontaktpersonene våre i NoIS hadde positive tilbakemeldinger til implementasjonen av opsjoner.

3.5.6 Sprint 6 - Testing & feilretting (Uke 16-17)

Innhold

- Testing
    - Tilgjengelighetstesting
    - Brukertesting
    - Systemtesting
- Feilretting

Milepæl

- Applikasjonen er ferdig testet
- Eventuelle feil er rettet

I sprint 6 skulle vi ha fokus på testing av programmet, fiksing av feil vi kom over, samt sørge for at WCAG ble fulgt gjennom hele applikasjonen. Det var i hovedsak planlagt systemtesting og brukertesting i denne sprinten, men det ble også gjort justeringer på enhetstester (les mer under kap. 7 i Produktdokumentasjonen). Som et resultat av

testingen, ble det funnet feil i programmet, som ledet til små justeringer i flere komponenter. Les mer under kapittel 3.5.6.1 og 3.5.6.2.

Etter alle rettinger var fikset, ble det ryddet i koden. Under ryddingen, la vi inn beskrivende kommentarer til funksjoner og andre steder det måtte trenges. I tillegg slettet vi kode og kommentarer som ikke var i bruk lengre, samt sjekket for gjennomgående riktig *syntax* (*clean code*).

I slutten av sprinten ble det gjennomført en demonstrasjon av applikasjonen for NoIS og Oslo REG. Demonstrasjonen førte til en fin diskusjon, om hvordan en på best mulig måte kan videreutvikle et slikt produkt. Vi fikk her mange gode tilbakemeldinger til arbeidet vi hadde gjort, samt skryt.

3.5.6.1 Rettinger etter brukertest

Som et resultat av brukertestingen ble et par ikoner byttet ut. For eksempel ble tannhjul-ikonet i navigasjonsbaren i dummy-applikasjonen byttet ut med et hamburger meny-ikon. Bruken av det nye ikonet skulle være med på å skape mindre forvirring, da tannhjul ofte er forbundet med innstillinger. Pil-ikonet i topp-navigasjonen til CSH'en ble byttet ut med et hus-ikon, da hus-ikonet enklere symboliserte ønsket funksjonalitet.

Brukertestingen resulterte også i implementering av laster-skjermer som gir brukeren en tilbakemelding om hva som skjer når data lastes. I tillegg ble en *tooltip* funksjonalitet implementert på alle knapper som bare benytter ikoner, for å hjelpe bruker å enklere forstå hva en handling vil føre til (Joyce, 2019).

3.5.6.2 Rettinger etter systemtest

Som et resultat av systemtestingen fant vi *bugs* som måtte rettes opp. Trykk på «les mer» knapp i resultatlisten til søkefunksjonen kunne til tider fungere ustabilt. Funksjonen rakk ikke alltid å sette riktig side i dummy-applikasjonen før den åpnet forsiden i CSH. Buggen ble løst ved å konvertere funksjonen til en Async-funksjon.

Dersom et tomt objekt eksisterte i databasen, ville ikke CSH'en vise *fallback* skjerm som forventet. Feilen ble rettet ved å sjekke om objektet fra databasen var tomt før innhold lastet til skjermen.

Den største *buggen* som ble rettet var i kurs-modulen, der selve modulen ikke funket slik vi hadde tenkt, og ble derfor skrevet helt om. Resultatet av omskrivingen gjorde at vi fikk kuttet ned på antall variabler, samt variabler som nå er i bruk har mer logiske navn. Navnene vil gagne videre utvikling da koden blir betydelig lettere å lese.

3.6 Refleksjoner og utfordringer

Prosjektet har gitt oss nye erfaringer og lærdom som i hovedsak er følger av en rekke varierende utfordringer.

3.6.1 Dokumentasjon og eksterne biblioteker:

Valget om å bruke JavaScript-rammeverket Vue 3 gav oss en del utfordringer knyttet til at det ble lansert i september 2020 (Ulrich & Massigner, 2021), og var derfor relativt nytt når prosjektet ble gjennomført. Rammeverket var en oppgradering fra Vue 2 og det ble etter hvert tydelig at veldig mange eksterne biblioteker fremdeles baserte seg på den gamle versjonen. I tillegg til Vue 3 hadde vi også bestemt oss for å bruke TypeScript, som dermed utelukket biblioteker som baserte seg på JavaScript. Det ble derfor brukt mye tid på å lete etter kompatible biblioteker og alternative løsninger.

Mangelfull dokumentasjon var gjennomgående i flere av bibliotekene vi endte opp å benytte oss av. For eksempel jsPDF, der dokumentasjonen kun belaget seg på at nøyaktig samme innhold skulle skrives til PDF'en hver gang. For Vue 3 var dokumentasjonen i utgangspunktet god, men det at rammeverket var såpass nytt gjorde at det var lite informasjon å hente fra diskusjonsforumer. Det gjorde at det måtte mye prøving og feilsøking til for å få på plass funksjonaliteten.

3.6.2 Azure

Ettersom vi ikke hadde jobbet med Azure tidligere, var det mange tjenester å få oversikt over. Azure er en stor plattform, som tok tid å bli kjent med, før tjenester kunne kartlegges og benyttes. For eksempel var det utfordrende å koble Azure Active Directory (Azure AD) til Vue applikasjonen. Etter dialog med utvikler hos NoIS fikk vi tips om det eksterne biblioteket MSAL (les mer under kap. 3.4 i Produktdokumentasjonen), som løste utfordringen vår.

Et større problem var knyttet til Azure Blob storage, som vi ønsket å bruke til lagring av bilder og videoer. Utviklerne hos NoIS hadde ikke erfaring med tjenesten. Vi ble stående helt fast og ble nødt til å se etter andre løsninger. For å ikke bruke unødvendig med tid ble det bestemt å heller benytte Firebase Cloud Storage, som vi hadde kjennskap til fra før av.

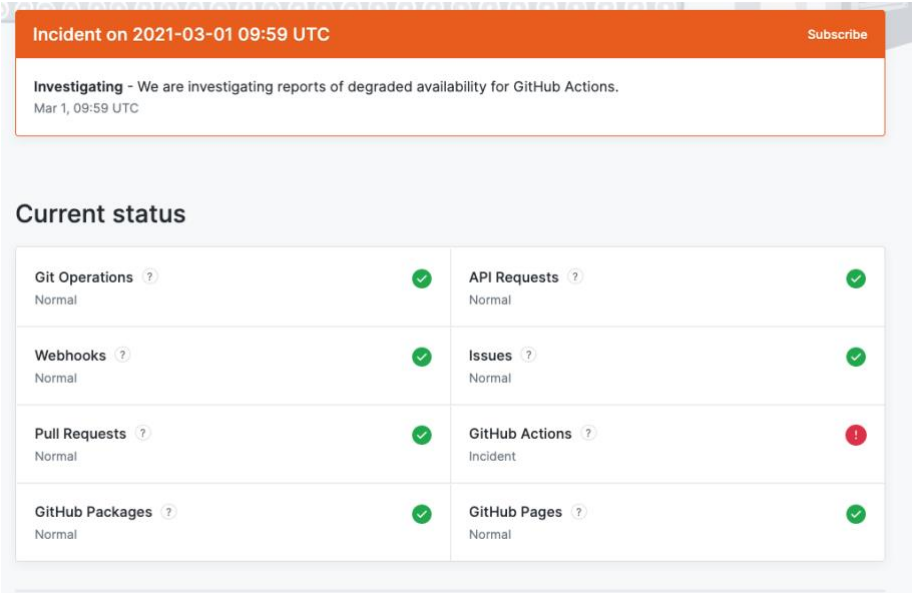
3.6.3 Testprosessen

Under testing i applikasjonen kom vi over en rekke problemer, som vi måtte jobbe rundt. For enhetstesting visste vi at en linje dekning på 100% ikke ville være mulig å oppnå. Vue Test Utils var ikke kompatibelt med Vue 3 Composition API (GitHub, 2020). Det gjorde at koden i *setup*-metoden ikke var mulig å aksessere. Kode som var avhengig av variabler i *setup*-metoden kunne derfor ikke testes. Funksjoner vi ikke fikk testet i enhetstesten, ble derfor testet i systemtesten i stedet.

Under systemtesten måtte det logges inn for å få tilgang til funksjonaliteten i CMS'en. Cypress var ikke i stand til å logge seg inn via brukergrensesnittet. Etter omfattende søk, ble det brukt en Cypress *Command* som hermet etter kallet brukergrensesnittet gjør når den logger inn. Løsningen gjorde det også enklere å gjenbruke innlogging-kommandoen i andre deler av testen.

3.6.4 Eksterne problemer

Den 01/03-2021 opplevde vi at GitHub Actions *workflows* var satt i kø, noe som resulterte i at applikasjonen ikke ble oppdatert når vi *pushet* til *master-branchen*. Etter litt søking fant vi ut at Github hadde et problem med denne tjenesten (se figur 8) (Github, u.å). Problemet gjorde at vi ikke fikk gjennomført *deployment* av applikasjonen den dagen.



Figur 8 - Github status som viser feil med actions den 01/03/21

Et annet uforutsett problem var grunnet en intern misforståelse hos administrasjon hos Oslomet. Misforståelsen gjorde at vi ikke fikk drøftet vår tilnærming til prosjektet med veilederen vår før slutten av forprosjektet. Når vi fikk kontakt med veileder, måtte oppgaven nedskaleres. Heldigvis var NoIS fleksible på oppgaven.

4. Resultatet

Det er vanskelig å forutse hva det endelige resultatet av et produkt vil bli i oppstarten av et prosjekt. Det er ofte sånn at man tror man skal få gjort alt man ønsker, men i realiteten må det oftest gjøres noen kompromisser.

En måte å analysere sluttproduktet på er ved å sammenligne hvorvidt kravene fra kravspesifikasjonen oppfylles. Det er her de funksjonelle og de ikke-funksjonelle kravene som vektlegges. Vi har oppfylt alle kravene, foruten det ikke-funksjonelle kravet «Systemet skal møte WCAG-krav nivå AA». Det er to krav som er delvis oppfylt. Det vil si at deler av kravet har blitt nedprioritert, da nytteverdien ville vært lavere enn innsatsen som skal til for å møte kravet. Les mer om kravene under kapittel 6.1 i Produktdokumentasjonen.

I tillegg er det utviklet funksjonalitet hentet fra opsjons listen. Applikasjonen oppfyller dermed flere krav enn den originale kravspesifikasjonen. Det er fortsatt punkter under opsjon som ikke er implementert, men disse ble sett på som for store til å implementere på tiden som var satt av til gjennomføringen av prosjektet i denne omgang.

Underveis i prosjektet ble planlagt bruksområde for sluttproduktet endret. Originalt ønsket NoIS at sluttproduktet skulle være tilnærmet klart til å brukes hos kunden Oslo REG. Etter nedskaleringen ble planlagt bruksområde justert til å kun brukes til demonstrasjonsformål. Både gruppen og NoIS anser sluttproduktet til å møte tenkt bruksområde.

Da en av rammebetingelsene til prosjektet var at applikasjonen skulle legges til rette for videreutvikling hos NoIS, ble bedriftens teknologier benyttet. I tillegg ble det skrevet kommentarer i koden, for å gjøre den lettere å lese for en utvikler som skal jobbe videre på produktet. Applikasjonen er samtidig optimalisert for videre arbeid gjennom gode clean code rutiner.

Sluttproduktet er testet og demonstrert til oppdragsgiver. Både gruppen og oppdragsgiver er fornøyde med at applikasjonen demonstrerer hvordan en kontekstsensitiv hjelp kan gjøre hjelpen lettere tilgjengelig for brukere.

5. Avslutning

Gjennom arbeidet med sluttprosjektet har vi som utviklere utviklet oss faglig. Det med tanke på at sluttproduktet er utviklet i flere teknologier som var nye for oss. Sett tilbake i tid vil det være ting som kunne vært gjort annerledes. Bruken av støtteverktøy kunne vært bedre gjennom prosessen. For eksempel kunne dagboken vært ført mer detaljert. Selv om dagboken ikke er ført like konsekvent gjennom hele prosessen, har den vært til stor hjelp når sluttrapporten skulle skrives. Alt satt i perspektiv er vi fornøyde med hvordan prosessen har utspilt seg dette semesteret, og hva resultatet har blitt til. Vi som utviklere vil kunne se tilbake på prosjektet som en lærerik utviklingsprosess. Det å gjennomføre et prosjekt av denne størrelsen, ser vi på som en stor bragd.

Under prosjektet har forholdet mellom oss og NoIS utviklet seg positivt. Hele gruppen fikk tilbud om videre jobb etter endt studie. Ettersom vi har jobbet sammen gjennom hele studietiden, synes vi det er veldig gøy at vi også skal starte arbeidskarrieren sammen etter sommeren.

5.1 Videre

Per i dag oppfyller utviklet produkt minstekravet satt som avtale mellom oss og NoIS. Videre er det opp til NoIS hvordan de ønsker å bruke produktet. Det har vært diskutert en videreutvikling av produktet, slik at det kan benyttes som en hyllevare i NoIS sine systemer.

De opsjonelle kravene i kravspesifikasjonen er funksjonaliteter som NoIS har uttrykt ønske om å implementere. For eksempel ønsket de en søkefunksjon som benytter en AI til å analysere søkehistorikken. De så videre for seg at historikken kunne benyttes til å generere innhold i en FAQ-bot.

Foruten de opsjonelle kravene har det vært diskutert å lage en mobilvennlig versjon av CSH'en. For et produksjonsklart produkt bør det også være mulighet for å legge inn flere bilder og videoer, i tillegg til å kunne endre rekkefølgen på innholdet i CSH'en.

6. Referanser

Bigelow, S. (2020, april). *Microsoft Azure.* TechTarget https://searchcloudcomputing.techtarget.com/definition/Windows-Azure

Cypress. (u.å.). *Component testing Introduction*. https://docs.cypress.io/guides/component-testing/introduction#Vue

Github. (u.å). *Current status.* Hentet 1. Mars 2021 fra https://www.githubstatus.com/

Github. (2020, 23 oktober). *setData #228.* https://github.com/vuejs/vue-test-utils-next/issues/228

Joyce, A (2019, 27. Januar) *Tooltip Guidelines.* Nielsen Norman Group. https://www.nngroup.com/articles/tooltip-guidelines/

jsPDF. (u.å.). *A library to generate PDFs in JavaScript.* *http://raw.githack.com/MrRio/jsPDF/master/docs/*

Levison, M. (2019, 5. november). *Choosing a scrum sprint length - shorter beats longer.* Agile pain relief. https://agilepainrelief.com/blog/choosing-scrum-sprint-length.html

Microsoft. (u.å). *Support.* Hentet 19. Mai 2021 fra https://support.microsoft.com/en-us/office/welcome-to-microsoft-teams-b98d533f-118e-4bae-bf44-3df2470c2b12

Norconsult Informasjonssystemer. (u.å). *Om oss*. https://www.nois.no/om-oss/om-nois/

Npm. (u.å). *Firebase - app success made simple.* Hentet 12. Mai 2021 fra https://www.npmjs.com/package/firebase

Oslo Kommune. (u.å.). *Designmanual: Oslo kommunes visuelle identitet*. https://designmanual.oslo.kommune.no/

Purdila, A. (u.å). *What is Adobe XD.* Envato tuts+. Hentet 19.mai 2021 fra https://webdesign.tutsplus.com/articles/what-is-adobe-xd--cms-36536

Sass. (u.å). *Documentation.* Hentet 19. Mai 2021 fra https://sass-lang.com/documentation

Schwarzmuller, M. (2020, desember) *Vue - the complete guide(w/ Router, Vuex, Composition API)* [Video]. Udemy. https://www.udemy.com/course/vuejs-2-the-complete-guide/

Sharma, A. (2018, 10. oktober). *Why you should use typescript for developing web applications.* DZone. https://dzone.com/articles/what-is-typescript-and-why-use-it

Sommerville, I. (2018). *Software Engineering* (10. utg.). Pearson.

StackOverflow. (2019). *Developer Survey Results: Most Popular Development Environment.* https://insights.stackoverflow.com/survey/2019#development-environments-and-tools

StackOverflow. (2021). *Jest error: "preset @vue/cli-plugin-unit-jest/presets/typescript-and-babel not found"*. https://stackoverflow.com/questions/66817469/jest-error-preset-vue-cli-plugin-unit-jest-presets-typescript-and-babel-not-found

Ulrich, A. & Massigner, M. (2021, 28. April). *Vue 3* - A roundup of infos about the new version of Vue.js. *Blog & Bookmarks.* https://madewithvuejs.com/blog/vue-3-roundup

Vue. (u.å). *Typescript support.* Hentet 19. mai 2021 fra https://v3.vuejs.org/guide/typescript-support.html#project-creation

Vue. (u.å). *What is vuejs?.* Hentet 30. Januar 2021 fra https://vuejs.org/v2/guide/

World Wide Web Consortium. (2020, 17. Oktober). *Web Content Accessibility Guidelines (WCAG) Overview.* https://www.w3.org/WAI/standards-guidelines/wcag/

PRODUKTDOKUMENTASJON

Gruppe 8

OSLOMET - BACHELOR 2020 / 2021

Forord

Produktdokumentasjonen har som formål å beskrive det endelige produktet som leveres. Den tar for seg systemarkitekturen - som innebærer de tekniske aspektene som utgjør de ulike komponentene, API-et og databasen. Videre skildres det omkring kravspesifikasjonen, systemets virkemåte og design.

Fullstendig forståelse av produktdokumentasjonen forutsetter en viss teknisk kompetanse og innsikt i oppbygging av *single-page* applikasjoner (SPA). Det er brukt tekniske terminologier og eksempler som inneholder kode for å forklare oppbyggingen og virkemåten til applikasjonen. Kjennskap til JavaScript rammeverket Vue.js vil være en fordel, men ikke et krav for å få utbytte av dokumentasjonen.

Innhold

1. Introduksjon av produkt og oppbygging

Produktet er en webapplikasjon som består av tre deler: en dummy-applikasjon, en kontekstsensitiv hjelp (CSH) og et publiseringssystem (CMS). Til sammen danner disse komponentene vårt bachelorprosjekt som gjøres for Norconsult Informasjonssystemer (NoIS). For mer informasjon om prosjektet, se prosessdokumentasjonen.

Applikasjonen kan aksesseres via lenken under.

CSH Demo

Applikasjonen har en serverløs *backend* bestående av Azure funksjoner som gjør jobben med å hente, skrive, oppdatere og slette data fra en MongoDB database. Azure benyttes også for autentisering i CMS gjennom Azure Active Directory (Azure AD), og hosting gjennom Azure Static Web Apps. I tillegg til Azure benytter vi oss av Firebase Cloud Storage for å lagre bilder og videoer.

Klientsiden av applikasjonen (*frontend*) er skrevet i JavaScript rammeverket Vue.js versjon 3, som ble utgitt i september 2020. Ved hjelp av HTTP-kall mot Azure funksjonene kan klientsiden sende data til og hente data fra databasen. Basert på hvilken side brukeren befinner seg på i dummy-applikasjonen, vil CSH'en hente innhold fra databasen. CMS delen av applikasjonen er beskyttet bak en innloggingsmur. Microsoft Authentication Library (MSAL) brukes her for å kommunisere med Azure AD. Når logget inn, kan en innholdsskaper gjøre endringer på dataene i databasen, og se hvordan det vil bli seende ut i CSH'en i sanntid. Så fort innholdsskaperen lagrer vil innholdet i CSH'en oppdatere seg.

Applikasjonen er optimalisert for *desktop* og nettbrett, og på disse plattformene er den designet responsivt. Når det gjelder font og farger følges Oslo Kommune sin designmanual. Alle ikoner er hentet fra samme bibliotek og hjelpetekster er tilgjengelig dersom man skulle bli usikker på hva ikonene betyr. Videre er applikasjonen universelt utformet for å møte personer med funksjonsnedsettelser.

2. Kravspesifikasjon og samsvar

2.1 Forord

Denne delen av produktdokumentasjonen tar for seg kravspesifikasjonen, som har sin hensikt å gi leseren en oversikt over innholdet som tilbys i applikasjonen. Den er utviklet i samarbeid med NoIS, der hensikten har vært å definere ulike krav for å dekke applikasjonens behov. Videre formål inkluderer å kunne opparbeide en bedre forståelse av systemet, også ved hjelp av datamodeller.

2.2 Leserveiledning for kravspesifikasjonen

Kravspesifikasjonen innledes med å beskrive de ulike krav som ble utformet i forprosjektet, for så å gjengi, illustrere og forklare eventuelle endringer som er foretatt. For å danne en god forståelse av systemet som skal utvikles er kravspesifikasjonen inndelt i funksjonelle krav, ikke-funksjonelle krav og opsjonelle krav, samt datamodeller.

2.3 Funksjonelle krav

Funksjonelle krav er de kravene som kan assosieres med systemets tekniske funksjonaliteter. Kravene vil beskrive hva systemet skal gjøre, og innebærer dermed hva systemet tilbyr brukerne (Jansen, 2018).

Figur 1 viser de opprinnelige kravene. «OK» i den siste kolonnen indikerer at kravet er implementert. Som man kan se i tabellen er alle de opprinnelige kravene gjennomført.

Nr.	Opprinnelige funksjonelle krav	
1.	CSH'en skal gi brukeren informasjon om innholdet i siden hen befinner seg.	OK
2.	CSH'en skal tilby funksjon for søk etter informasjon om systemet	OK
3.	CSH'en skal forklare ord og uttrykk i parent systemet som er potensielt vanskelige å forstå.	OK
4.	CSH'en skal kunne vise demo-videoer der det er aktuelt.	OK
5.	CSH'en skal tilby brukeren en snarvei til support dersom hen ikke finner hjelpen hen trenger	OK
6.	Applikasjonen skal gi brukeren tilbakemeldinger på ønskede og uønskede hendelser i programmet.	OK
7.	Systemet skal ha et eget brukergrensesnitt for konfigurasjon og publisering av informasjon (CMS).	OK

Figur 1 - De opprinnelige funksjonelle kravene

I tillegg til de opprinnelige kravene er det hentet inn to krav fra de opsjonelle kravene (se figur 2). Valg av hvilke krav som skulle prioriteres ble basert på hva oppdragsgiver ønsket implementert, samt hvor mye tid som var til rådighet.

Nr.	Funksjonelle krav hentet fra opsjon	
1.	CSH'en skal tilby en funksjon for å eksportere CSH'ens innhold i sin helhet.	OK
2.	CSH'en skal inneholde en kursmodul der brukeren kan få en enkel opplæring i systemet.	OK

Figur 2 - Funksjonelle krav fra opsjon

2.4 Ikke-funksjonelle krav

De ikke-funksjonelle kravene definerer hvordan systemet skal innfri de funksjonelle kravene. De sier noe om egenskaper applikasjonen har og hvordan den skal oppføre seg. Man skiller ofte mellom tre underkategorier av ikke-funksjonelle krav: produktspesifikke, organisasjons spesifikke og eksterne krav (Sommerville, 2018, s. 107-109).

De produktspesifikke kravene omhandler systemets effektivitet. For eksempel at en søkefunksjon ikke skal bruke mer enn 5 sekunder på å finne eventuelle resultater. De organisasjons spesifikke kravene er krav til hvordan applikasjonen fungerer, når innhold og funksjonalitet er implementert. For eksempel at systemet skal møte WCAG-krav nivå AA. Videre handler de eksterne kravene om personvern, etiske retningslinjer o.l. Dette er av type krav som ikke er implementert i denne applikasjonen da det var utenfor oppgavens relevans.

Figur 3 viser de ikke-funksjonelle kravene. «OK» i den siste kolonnen indikerer at kravet er implementert. Som man kan se i tabellen er det kun punkt 6. «Systemet skal møte WCAG-krav nivå AA» som ikke er 100% gjennomført. Opprinnelig var dette kravet kun nivå A, men etter å ha fått kjennskap til den norske forskriften for universell utforming (Forskrift om universell utforming av IKT-løsninger, 2013, § 1-11), ble kravet oppjustert til også å innebære nivå AA. For å innfri nivå A mangler applikasjonen ett punkt, og det samme gjelder nivå AA. Les mer om implementasjon av WCAG under kapittel 6.1.

Nr.	Ikke-funksjonelle krav	
1.	CSH'en skal være optimalisert for pc og nettbrett	OK
2.	CSH'en skal ikke dekke mer enn 30% av skjermen.	OK
3.	CSH'en skal kunne utvikles ved hjelp av smidige utviklingsmetoder.	OK
4.	Søkefunksjonen skal ikke bruke mer enn 5 sekunder på å finne resultat	OK
5.	CSH'en må være brukervennlig	OK
6.	Systemet skal møte WCAG-krav nivå AA	Delvis
7.	Systemets komponenter skal optimaliseres for gjenbrukbarhet hos NoIS.	OK
8.	CMS'en skal ikke være tilgjengelig dersom bruker ikke er logget inn.	OK

Figur 3 - Ikke funksjonelle krav

2.5 Opsjonelle krav

De opsjonelle kravene ble utviklet i en tidlig fase, og var i motsetning til de funksjonelle og ikke-funksjonelle kravene tenkt at skulle være noe som gruppen kunne strekke seg etter dersom det var tid og ressurser til rådighet. Kravene er av typen funksjonelle krav og er basert på ønsker for videre funksjonalitet i dialog med oppdragsgiver.

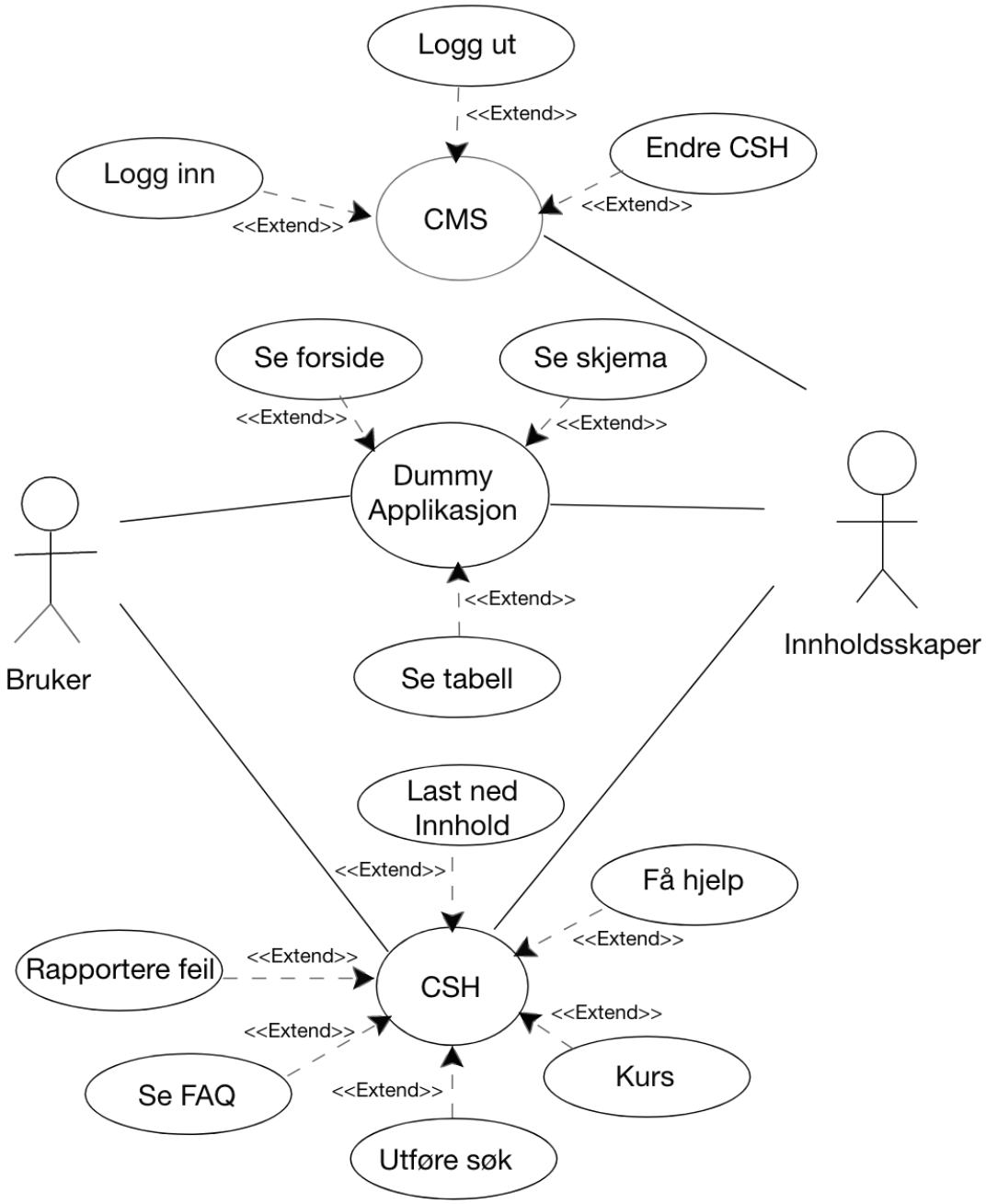
Figur 4 viser de opsjonelle kravene. «OK» i den siste kolonnen indikerer at kravet er implementert. Som man kan se i tabellen er krav 2. og 7. de eneste opsjonelle kravene vi hadde tid og ressurser til å fullføre.

Nr.	Opsjonelle krav	
1.	CSHen skal tilby brukeren å gjøre individuelle tilpasninger (e.g. språk, hvor stor del av skjermen den tar osv)	
2.	CMS'en skal tilby funksjon for å eksportere CSH'ens innhold i sin helhet	OK
3.	Søkefunksjonen skal bruke en AI til å analysere søkehistorikken for å forbedre søkeresultatene	
4.	CSH'en skal som en del av support tilby en FAQ-bot som benytter seg av søkefunksjonen sin AI	
5.	CSH'en skal kunne vise ulikt innhold avhengig av hvilke roller som er tildelt brukerne	
6.	CSHen skal respondere på hjelpe-knapper i systemet	
7.	CSH en skal inneholde en kursmodul der brukeren kan få en enkel opplæring av systemet	OK

Figur 4 - Opsjonelle krav

2.6 Datamodell

For å danne et større spekter av forståelsen for kravspesifikasjonen og dens krav ble det utformet en datamodell som visualiserer applikasjonens innhold. Datamodellen (se figur. 5) er av typen *Use Case* diagram, eller bruksmønster, som brukes for å vise de spesifikke oppgavene som brukerne av applikasjonen skal kunne utføre (Ødegaard, 2019). I vårt tilfelle viser diagrammet hvilke handlinger man kan utføre i de ulike delene av applikasjonen, samtidig viser den sammenhengen mellom handlingene og de ulike brukerne av applikasjonen. Handlingene er her knyttet til kravene i den funksjonelle kravspesifikasjonen.



Figur 5 – Use case diagram av system

3. Eksterne biblioteker

3.1 Axios

Axios gjør det enkelt å sende HTTP forespørsler til REST API'et (Azure funksjonene) for å gjennomføre CRUD operasjoner mot databasen.

3.2 Bootstrap

Bootstrap er et *open-source* bibliotek for frontend. Biblioteket inneholder CSS og JavaScript designmaler og pre-definerte klasser som gjør det enklere, og ikke minst raskere, å designe universelt og responsivt.

3.3 Vue-toastification

Vue-toastification er et bibliotek med varslings bokser som enkelt kan tilpasses ulike tilbakemeldinger man ønsker å gi brukere av en applikasjon. Eksempler på tilpasninger inkluderer plassering på skjermen, farge, ikon, tekst og tidsbegrensning.

3.4 Microsoft Authentication Library for JavaScript (MSAL)

MSAL-browser benyttes for å gi JavaScript applikasjoner tilgang til å koble til og autentisere brukere med Azure AD. Biblioteket bruker *Oauth 2.0* autorisasjonskode flyt med *PKCE* (*Proof key code exchange*) som standard (Npm, 2021).

3.5 Splitpanes

Splitpanes er et bibliotek som gjør det enkelt å dele skjermen i flere ruter. Biblioteket kan tilpasses antall ruter, om de skal kunne justeres og hvordan.

3.6 Vue-Router

Vue-Router er det offisielle navigasjons-biblioteket til Vue.

3.7 Vuex

Vuex er et tilstands administrasjons (*state-management*) bibliotek, som brukes i Vue applikasjoner. Vuex fungerer som en lagringsplass for tilstander for alle komponentene i applikasjonen (Vuex-store). Vuex-store har regler som sikrer at tilstandene bare kan muteres på en forutsigbar måte (Vuex, 2021).

3.8 jsPDF

JsPDF er et bibliotek som brukes til å generere PDF i JavaScript.

3.9 IconPark Icons

IconPark er et bibliotek som fungerer som en ikon-database. En kan her finne mange standardiserte ikoner som enkelt kan benyttes i Vue applikasjonen. Ikonene importeres og brukes på samme måte som komponenter under template-delen av Vue koden.

3.10 Firebase

Firebase biblioteket gir utvikler verktøyene en trenger for å jobbe mot skytjenesten (Firebase).

4. Frontend

4.1 Vue og oppsett

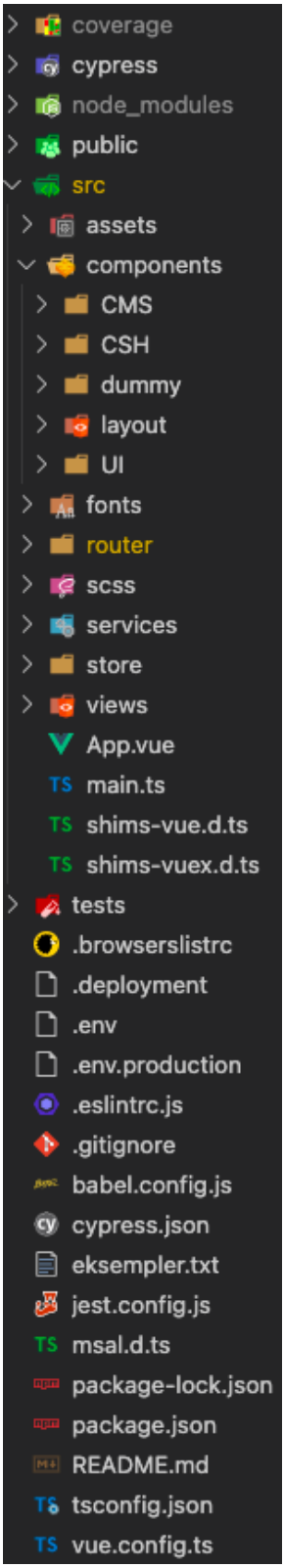
Hele front-enden er laget i Vue, hvor utgangspunktet er boilerplate-kode laget av «Vue create» kommandoen. Under denne kommandoen får man mulighet til å gjøre valg knyttet til oppsettet av applikasjonen. Her er følgende valgt:

- Vue 3
- Typescript
- Vue-router
- Vuex

Valg av versjon falt på Vue 3 ettersom det er den nyeste versjonen. I Vue 3 har man også mulighet til å benytte den nye strukturmodellen Composition API, som erstatter det gamle Option API'et. Fordelene med Composition API inkluderer at koden kan gjøres mer gjenbrukbar og enklere å lese ettersom strukturen er tydeligere.

Vue kan skrives i både JavaScript og TypeScript. Med strengere type deklarering er TypeScript mer forutsigbar og enklere å lese og debugge. Valget falt derfor på TypeScript.

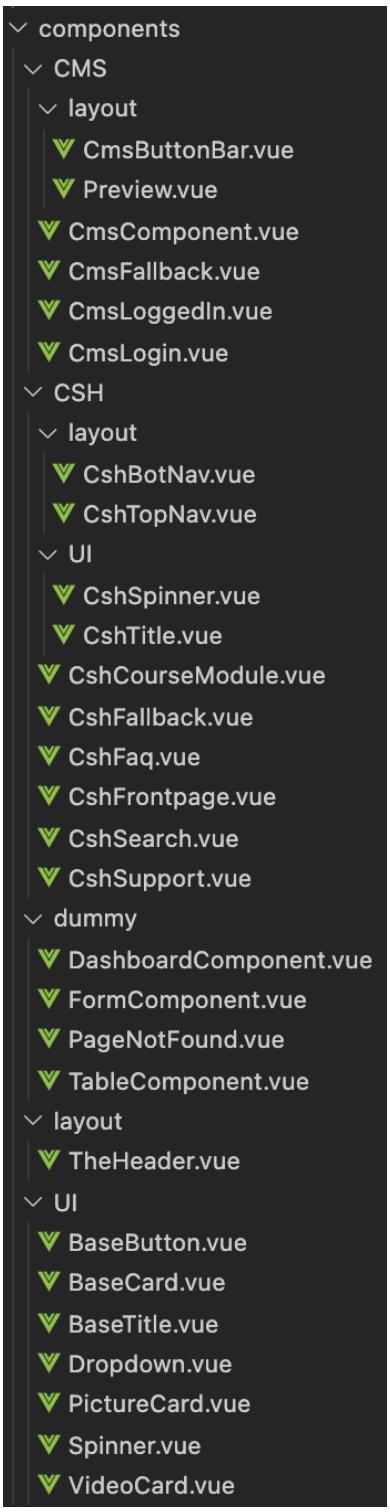
For navigasjon i applikasjonen benyttes Vue-router. Alle sider som skal navigeres gjennom Vue-router må registreres i en *index.ts* fil som ligger i en *router*-mappe under *src* i prosjektet. For at router-registreringene skal være tilgjengelige for applikasjonen, må *index.ts* filen importeres i *main.ts*. Registrerte sider vil da være tilgjengelige gjennom *router-view* komponenten. For å navigere mellom dem kan man enten bruke *router-link* komponenten eller en *router.push*-metode.



Figur 6 - Fil-tre

For å lagre tilstander (*state*) benytter applikasjonen Vuex, som er et bibliotek som lagrer informasjon tilhørende komponenter i applikasjonen. Ved bruk av Vuex blir tilstander lagret, og det blir sikret at endringer av tilstand gjøres på lik måte gjennom hele applikasjonen. Implementering av tilstandsvariablene og logikken (mutasjoner, handlinger osv.) ligger i en *store*-mappe under *src* i prosjektet. *Store*-mappen har en *index.ts* fil på rota som står for lagring av autentiserings-tilstanden. I tillegg har den en *modules*-mappe som inneholder tilsvarende filer for andre tilstander benyttet i applikasjonen. Ettersom det er lagret mange ulike tilstander i vår applikasjon, er det valgt å lage individuelle filer for å gjøre koden ryddigere. Filene i *modules*-mappen importeres som moduler i *index.ts* filen på rota i *Store*, og er på denne måten tilgjengelige for applikasjonen.

I vår applikasjon er *main.ts* rotkomponent, og har som oppgave å knytte større biblioteker til applikasjonen. Eksempler på store biblioteker er Firebase, Vuex, Vue-router og MSAL. Den andre oppgaven til filen er å vise *App.vue* ved å finne HTML-elementet med ID «app» i *index.html*. *App.vue* har i vår applikasjon ansvar for inndelingen av skjermen. Skjermen deles mellom dummy-applikasjonen og CSH ved hjelp av Splitpanes. Styling som er gjennomgående i hele applikasjonen (globalt) er også definert i denne komponenten.



Figur 7 - Components-mappen

I *Views*-mappen under *src* ligger filene som har med visningen av komponentene å gjøre. De forskjellige komponentene ligger videre i *components*-mappen under *src*. I *components*-mappen er komponentene sortert i undermapper tilsvarende de forskjellige delene av applikasjonen; Dummy, CMS og CSH. Komponenter som ofte brukes flere steder i applikasjonen og som representerer en mindre del av en større komponent er lagt i en egen UI-mappe. I layout-mappen ligger komponenter som er en fast del av layouten til den delen av applikasjonen som mappen ligger under. For eksempel er «CshBotNav», som ligger i layout-mappa til CSH, alltid synlig i CSH'en. Se figur 7 for oversikt over alle komponentene og mappestrukturen i *components.*

4.2 Dummy-applikasjonen

Dummy-applikasjonen er laget for å ha noe å bygge CSH'en på og vil dermed fungere som konteksten til CSH'en. Informasjonen i CSH'en vil handle om hvordan bruke dummy-applikasjonen. Dummy-applikasjonen inneholder 4 sider: Dashboard (forsiden), skjema-side, tabell-side og en feil-side. Dummy-applikasjonen er tilgjengelig for både standard brukere og innholdsskapere.

Alle dummy-applikasjonens undersider inneholder UI-komponenten «BaseTitle». Komponenten får undersidens tittel sendt gjennom *props*. Hensikten med tittelen er å gjøre det enklere å se sammenhengen mellom dummy-applikasjonens side og tilhørende side i CSH'en. Tittelen er kun implementert for å støtte demonstrasjonsformålet til applikasjonen, og CSH'ens funksjonalitet er med andre ord ikke avhengig av denne sammenhengen.

4.2.1 Navigasjonsbar

Navigasjonsbaren er tilgjengelig på toppen av skjermen på alle sider i applikasjonen. Navigasjonsbaren vil for en standard bruker inneholde to knapper (se figur 8). Den ene er Oslo kommune sin logo, som er en hjem-knapp. Trykk på knappen fører alltid til Dashboard-siden (se kap. 4.2.2). Den andre knappen inneholder et "?" ikon med teksten "Hjelp". Trykk på knappen åpner CSH'en og *trykk* en gang til lukker CSH'en.

Figur 8 - Navigasjonsbar bruker

For innholdsskaper vil navigasjonsbaren i tillegg ha en ekstra knapp, som inneholder funksjonaliteten som kun er tilgjengelig for innholdsskapere (se figur 9). Trykk på knappen åpner nedtrekksmenyen der innholdsskaperen har to valg:

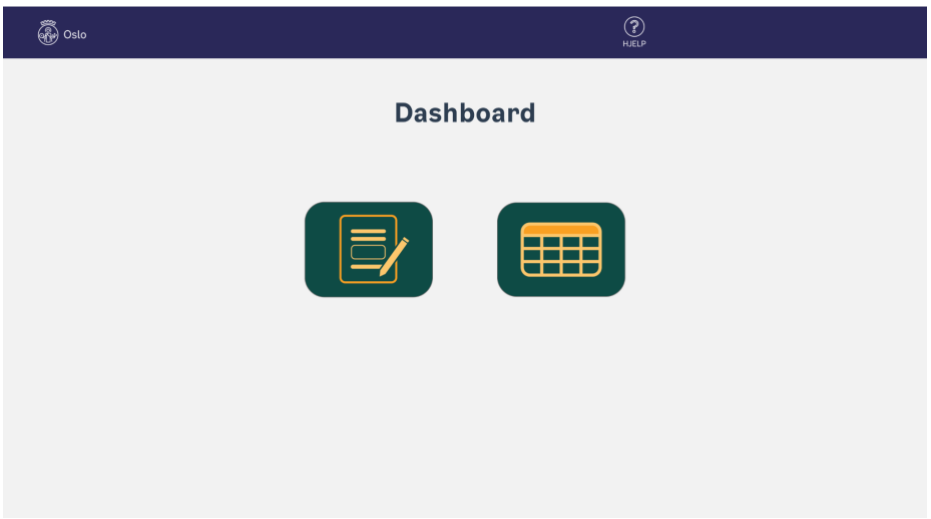
Valg 1: Trykk på “endre siden” for å endre innholdet i CSH’en.

Valg 2: Trykk på “logg ut” for å logge ut av CMS’en.

Figur 9 - Navigasjonsbar innholdsskaper

4.2.2 Dashboard

Dashboard er dummy-applikasjonens forside, og består av to knapper som fører brukeren til hver sin underside. Knappene benytter seg av Vue-router for å gjennomføre navigasjon til tabell-siden (se kap. 4.2.3) og skjema-siden (se kap. 4.2.4). Knappene er egendesignet og skal symbolisere sidene man havner på. De er hardkodet inn i *template*-delen av komponenten og kilden til ikonene befinner seg i *assets/icons* under *src* i prosjektet.



Figur 10 - Dashboard-side. Skjema-knapp til venstre og Tabell-knapp til høyre.

Hvis bruker trykker på skjema-knappen, vil «clickForm» funksjonen kjøres. Funksjonen gjennomfører Vue-router kommandoen *push*. En *push*-kommando navigerer brukeren til siden som tilsvarer verdien gitt i *path*. For tabell-siden vil fremgangsmåten for navigering være den samme.

```
const clickForm = (): void => {
  router.push({ path: "/form" });
};
```

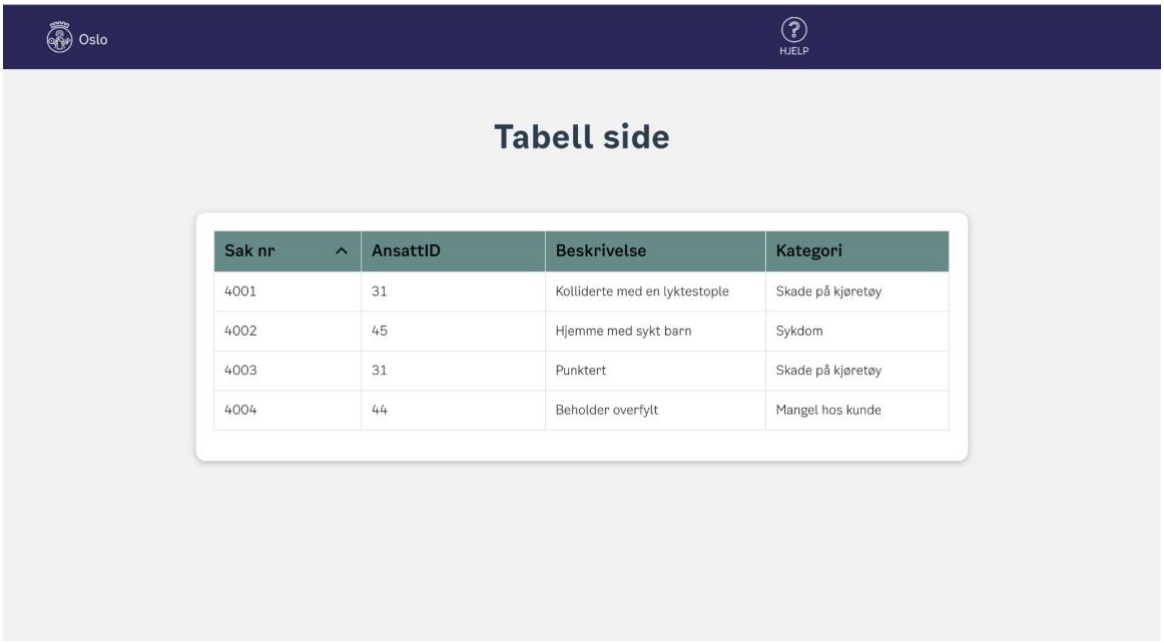
Figur 11 - «clickForm» funksjonen

4.2.3 Tabell-siden

Tabell-siden kan navigeres til fra dashboard-siden. Siden inneholder en tabell (se figur 13), der eneste funksjonalitet er sortering. En *onClick*-funksjon på kolonneoverskriftene lar brukeren sortere tabellen stigende/synkende basert på valgt kolonnenavn. Logikken bak selve sorteringen er gjort i en egen *service*, for å gjøre koden mest mulig ryddig. I Vue brukes servicer på funksjonalitet som gjør en fokusert oppgave som resulterer i eksport av noe. Servicer ligger separat fra komponentene i en egen mappe under *src*. Det er brukt en modell-klasse kalt «DataClass» for å holde strukturen på objektene i tabellen rydding når de sendes mellom tabell-komponenten og servicen (se figur 12). Dataene i tabellen er hardkodet ettersom det ikke er store mengder data og de ikke skal kunne endres.

```
class DataClass {
  caseNr: string;
  empId: number;
  description: string;
  category: string;
```

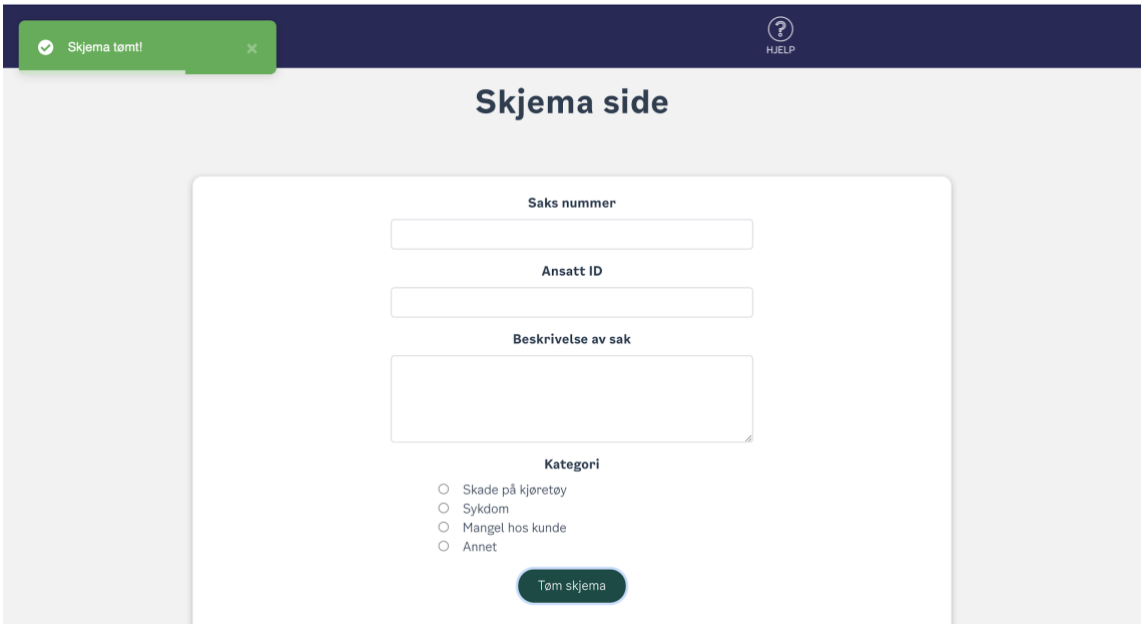
Figur 12 - tabell kode



Figur 13 - Tabell-siden

4.2.4 Skjema-siden

Skjema-siden kan navigeres til fra dashboard-siden. Siden inneholder et skjema som kan fylles ut (se figur 14). Ettersom skjemaet kun er laget for demonstrasjonsformål er det ingen form for validering på input-feltene. Feltene er likevel koblet til et *reactive* objekt gjennom *v-model,* slik at de kan tømmes.



Figur 14 - Skjema-siden, inkludert varslingen etter tømt skjema.

Nederst på siden er det en knapp, med teksten «Tøm skjema». Når knappen trykkes kalles «clearForm» funksjonen som nullstiller feltene i skjema, og returnerer en varsling i form av en *toast*. Toasten er av typen suksess, som gir den en grønn farge og et sjekk-ikon. Varigheten til toasten er satt til 2 sekunder ettersom teksten er relativt kort.

Dersom «Tøm skjema» knappen trykkes når modul 1 i kursmodulen i CSH'en er åpen, vil oppgave 2 i kursmodulen markeres som godkjent. Les mer om hvordan kursmodulen fungerer i kapittel 4.3.6.

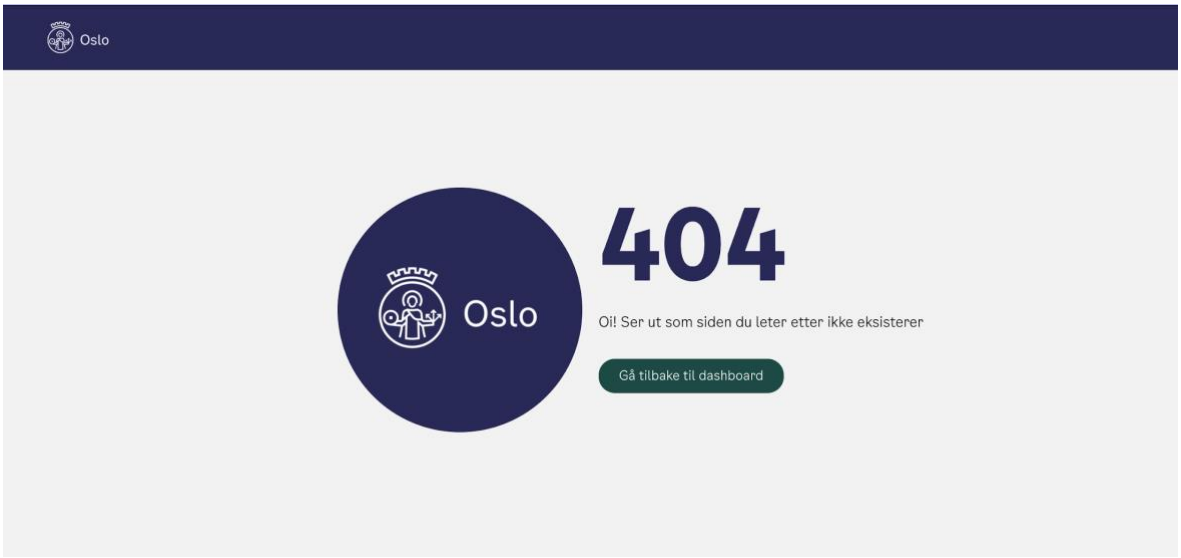
4.2.5 404 - Ikke funnet

404-siden vises når en ugyldig URL benyttes. En URL defineres som ugyldig når den ikke er dokumentert i Vue-router. Det sjekkes ved hjelp av en *path*: "/:pathMatch(.*)*". Alle filstier som ikke er definert fanges her og vil derfor kalle 404-siden.

```
{
  path: "/:pathMatch(.*)*",
  name: "PageNotFound",
  component: () => import("../components/dummy/PageNotFound.vue")
},
```

Figur 15 - PageNotFound i Vue-router

På 404-siden er det en tekst som sier at siden brukeren søker etter ikke eksisterer (se figur 16). Siden tilbyr også en knapp som vil navigere bruker tilbake til dashboard. Knappen fungerer med å kalle *router.push* kommando som peker på filstien til dashboard.



Figur 16 - 404-siden

4.3 CSH

CSH-delen av applikasjonen er der hjelp som har med bruk av dummy-applikasjonen vil vises for brukerne. Brukere har også mulighet til å laste ned denne hjelpen i form av et PDF-dokument. I tillegg har CSH'en en kursmodul, en søkeside, en side med ofte stilte spørsmål (FAQ) og skjema for å sende inn feilrapporter til support.

CSH'en befinner seg på høyre side av applikasjonen og kan justeres mellom 20% og 30% av skjermen. Når CSH'en åpnes fra navigasjonsbaren vil den ta 25% av skjermen. For å justere størrelsen kan en bruker dra i den venstre kanten av CSH'en.

4.3.1 Navigasjonsbarer

CSH'en inneholder to navigasjonsbarer. Den ene er plassert på toppen og den andre er plassert på bunnen av CSH'en.

Navigasjonsbaren på bunnen av CSH inneholder tre ikonér (se figur 17). Ikonene indikerer de ulike sidene som tilbys, som er: Søk-side, FAQ-side og kursmodul-side. Navigasjonen i CSH'en er knyttet mot Vuex, da vi ikke ønsket at denne delen av applikasjonen skal påvirke Vue-routeren. Eksempelvis vil trykk på lupe-ikonet i navigasjonsbaren kalle «clickSearch» funksjonen (se figur 18). Funksjonen kaller en mutasjon i Vuex-store for å sette tilstanden «page» til å være "search". En mutasjon

er en definert funksjon som endrer tilstanden som er lagret i Vuex-store. På denne måten fungerer mutasjonen som et mellomledd og forhindrer ulike endringer av samme tilstand. Videre er det en *computed* funksjon som følger med på tilstanden og setter side i CSH basert på denne. Samme funksjonalitet gjelder for alle knapper i navigasjonsbaren.

Figur 17 - Navigasjonsbar bunn

```
const clickSearch = (): void => {
  store.commit({
    type: "setPage",
    page: "search"
  });
};
```

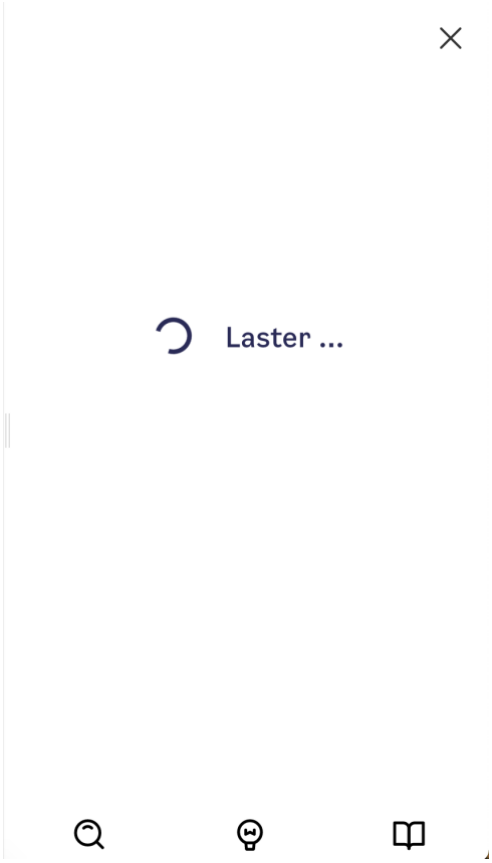
Figur 18 - CSH Eksempel navigasjon-kode

Topp-navigasjonen inneholder alltid et kryss, som ved trykk lukker CSH'en. Når brukeren befinner seg på CSH'en sin forside, er dette eneste knappen. Dersom bruker befinner seg på søk-siden, på FAQ-siden eller kurs-siden, vil topp-navigasjonen også ha en hjem-knapp. Ved trykk på hjem-knappen navigeres brukeren tilbake til forsiden. Befinner bruker seg derimot på feilrapportering-siden vil topp-navigasjonen ha en tilbake-knapp i stedet for hjem-knapp (se figur 19). Ved trykk på tilbake-knappen navigeres brukeren tilbake til FAQ-siden. For betydningen av ikonene se kapittel 6.3.

Figur 19 - Navigasjonsbar topp

4.3.2 Forside CSH

På CSH'ens forside vil det vises hjelp tilpasset siden brukeren befinner seg på i dummy-applikasjonen. Hjelpen hentes fra databasen i Azure ved hjelp av Axios, gjennom ett HTTP-kall i «getData» funksjonen. Funksjonen vil kjøres hver gang CSH'en åpnes og hver gang det navigeres i dummy-applikasjonen. Det gjøres ved bruk av en *watch*-metode. Metoden overvåker *path*, som er navnet på den aktuelle siden i dummy-applikasjonen, og kaller «getData» funksjonen hver gang *path* forandrer seg. *Path*-verdien brukes videre i HTTP-kallet for å hente riktig innhold til forsiden i CSH'en. Mens innholdet hentes, vil forsiden vise en laster-skjerm (se figur 20).



Figur 20 - Laster-skjerm

Figur 21 - Feil-skjerm

Dersom det eksisterer innhold til siden brukeren befinner seg på og innholdet kan hentes på riktig måte, vil laster-skjermen forsvinne og innholdet vises. Innholdet vil da kunne bestå av tittel, brødtekst, ordforklaringer, ett bilde og en video, men kan for eksempel også bare være tekst (se figur 22). Her er det opp til en innholdsskaper å

bestemme hva som skal vises på de ulike sidene. Dersom innhold til siden ikke eksisterer eller noe går galt under innhentingen, vil en feil-skjerm vises. Teksten på feil-skjermen vil da være tilpasset grunnen til at innholdet ikke kunne vises. Er for eksempel grunnen manglende innhold, vil teksten på figur 21 vises. Det gjøres ved at teksten blir satt i *catch'en* til HTTP-kallet, og sendes deretter som *props* til feil-skjerm komponenten.

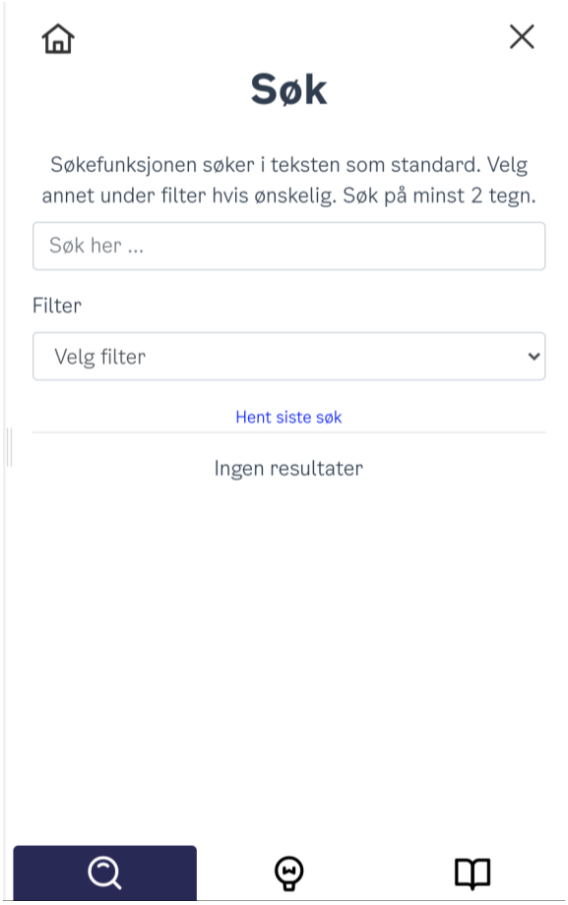
Figur 22 - CSH forside

Nederst på forsiden er det en knapp som gir bruker muligheten til å laste ned all hjelp i form av et PDF-dokument. Ved trykk på knappen vil «downloadPdf» funksjonen kalles. I første omgang vil funksjonen sette verdier som har med utforming av PDF å gjøre. Eksempel på slike verdier er; navn på filen som lastes ned, skrifttype, tekststørrelse, marg-verdi og linjelengde. Videre vil funksjonen gjøre et HTTP-kall mot Azure Functions for å hente all data fra databasen. For å forsikre oss om at all data blir med i PDF'en er funksjonen en *async*-funksjon som venter på HTTP-kallet. Det

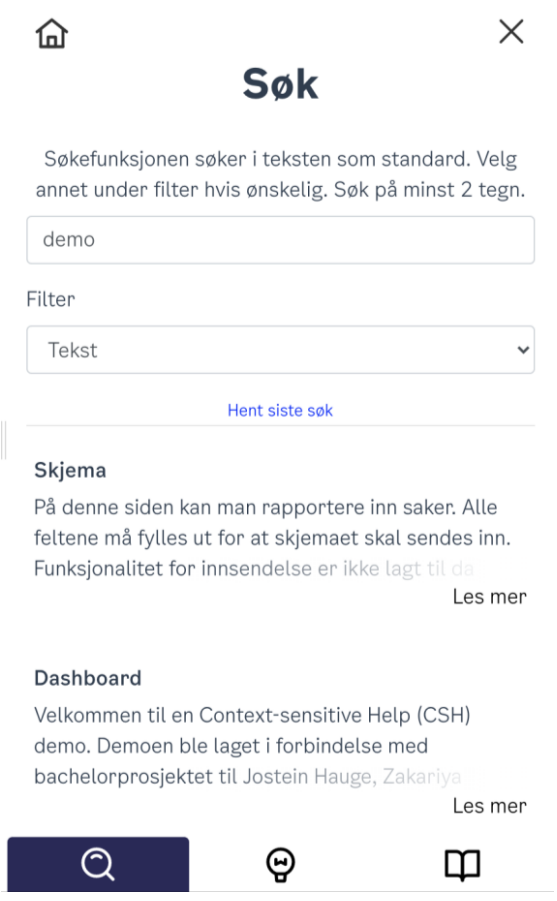
betyr at en kan sette deler av koden som må ventes på før annen kode kan kjøre. I vår kode ventes det på at Axios skal bli ferdig med HTTP-kall. Etter at alle data-objekter er hentet, lagres de i et array. Deretter løpes det gjennom arrayet for å skrive objektene til PDF i riktig rekkefølge. I PDF'en er det kun tekst og bilde fra objektene som blir med. Til slutt vil funksjonen lagre PDF med riktig navn, som i vårt tilfelle vil være «CSHcontent.pdf». Hvis en feil oppstår vil bruker få beskjed i form av en *error toast* som forklarer at noe gikk galt.

4.3.3 Søk-side CSH

På søkesiden har brukeren mulighet til å søke på innholdet i CSH'en (se figur 23 og 24). Det gjøres ved å legge inn ønsket søketekst i søkefeltet og evt. filtrere på enten tittel, tekst eller ordforklaringer. Standard filtrering av søk er satt til å være tekst. Brukeren har også mulighet til å hente siste søk, ved å trykke på «Hent siste søk» knappen over resultatlista.



Figur 23 - Søkefunksjon før søk



Figur 24 - Søkefunksjon etter søk

Når søkesiden åpnes, kjøres «onMounted» funksjonen, som kaller på «getAllData» funksjonen. «GetAllData» funksjonen gjør et HTTP-kall mot Azure Functions for å hente all data fra databasen, og lagrer deretter dataene til «allData» arrayet. Mens dataene hentes vil en laster-skjerm vises (se figur 20 under kap. 4.3.2). Dersom det ikke eksisterer noe data eller noe går galt under innhentingen, vil en feil-skjerm vises i stedet for søkefunksjonen. Går alt bra under innhentingen vil søkefunksjonen bli tilgjengelig.

For at et søk skal være gyldig må bruker skrive inn minst to tegn i søkefeltet. En *computed*-funksjon med navn «searchResult» vil kjøres hver gang søkefeltet eller filter-nedtrekksmenyen forandrer seg. Funksjonen inneholder en *switch-statement* som sjekker hvilket filter som er valgt (eller ikke valgt). Basert på resultatet av påstanden, vil «allData» arrayet filtreres, slik at kun objektene som samsvarer med valgt filter og valgt søketekst blir med videre. Det filtrerte arrayet blir til slutt returnert av funksjonen.

En *computed*-funksjon med navn «isResults» følger med på det som «searchResult» funksjonen returnerer. Hvis arrayet som returneres har minst ett resultat vil «isResults» returnere *true* og resultatlisten vil vises. I *template*-delen av komponenten løper en for-løkke gjennom arrayet og formaterer søkeresultatene i en liste. Hvis søket ikke gir noen resultater, vil teksten “Ingen resultater” vises i stedet.

I resultatlisten vises tittelen og de første tre linjene av teksten til et resultat. Brukeren kan også trykke på «les mer» knappen for å kalle «showMore» funksjonen. Funksjonen håndterer navigasjon til siden som resultatet representerer. *Async* er her brukt for å vente på at Vue-router navigerer dummy-applikasjonen før CSH’en kan bytte fra søkefunksjonen til forsiden. Grunnen til det er at forsiden i CSH’en er avhengig av at dummy-applikasjonen står på riktig side før den laster innholdet, for at riktig innhold skal lastes.

Komponenten har også en *watch*-metode som følger med på søkefeltet og filter-nedtrekksmenyen. Når søket forandrer seg vil søketeksten og filteret lagres i Vuex, slik at det siste brukeren søkte på alltid vil være tilgjengelig.

4.3.4 FAQ-side CSH

På FAQ-siden har brukeren tilgang til å få raske svar på ofte stilte spørsmål. Funksjonaliteten er tenkt å gjøre brukeropplevelsen enklere, ved at spørsmål og svar som mange lurer på er lett tilgjengelig.

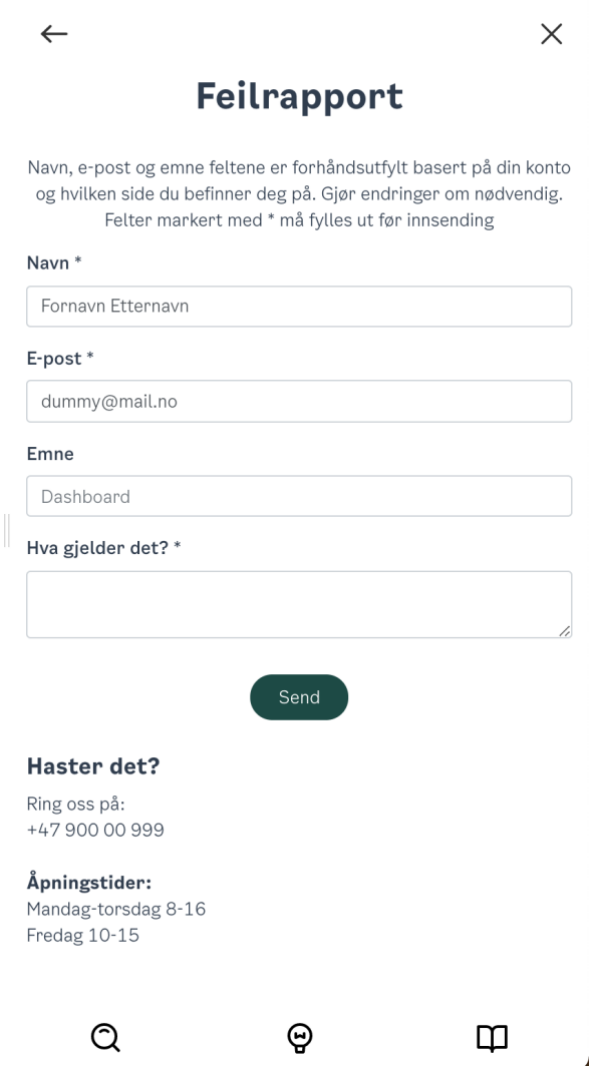
Når siden åpnes er det kun spørsmålene som er synlige. Ved å klikke på et spørsmål vil svaret på spørsmålet åpne seg under. Funksjonaliteten oppnås gjennom en *onClick*-funksjon som bytter mellom *true* og *false* på *boolean-variabelen* som tilhører det respektive spørsmålet. Når variabelen er *true* vil spørsmålet vises og ikonet som indikerer om spørsmålet kan åpnes eller ikke bytter fra pluss-ikon til minus-ikon.

På FAQ-siden har brukeren også tilgang til en «Rapporter feil» knapp. Når knappen trykkes, vil «clickSupport» funksjonen kalles. Funksjonen kaller en mutasjon i Vuex-store for å sette «page» tilstanden til å være “support”, og CSH’en bytter dermed til feilrapportering-siden (se kap. 4.3.5).

Figur 25 - FAQ forside

4.3.5 Feilrapportering-side CSH

På feilrapportering siden har brukeren mulighet til å fylle ut et skjema og tømme det. Feltene som kan fylles ut er; Navn, e-post, emne og et tekstfelt der brukeren kan forklare hva feilen gjelder (se figur 26). Ettersom applikasjonen skal brukes til demonstrasjonsformål fungerer ikke skjemaet slik som det ville gjort i et produksjons klart program. Feltene «Navn» og «E-post» ville vært forhåndsutfylt med informasjon hentet fra en brukerprofil. Feltene er derfor forhåndsutfylt med “Fornavn Etternavn” og “dummy@mail.no” for å simulere denne funksjonaliteten, men brukeren har også mulighet til å gjøre endringer hvis ønskelig. Feltene er koblet til et *reactive* objekt gjennom *v-model,* slik at endringer brukeren foretar i feltene dynamisk vil reflekteres i objektet.



Figur 26 - Feilrapportering-side

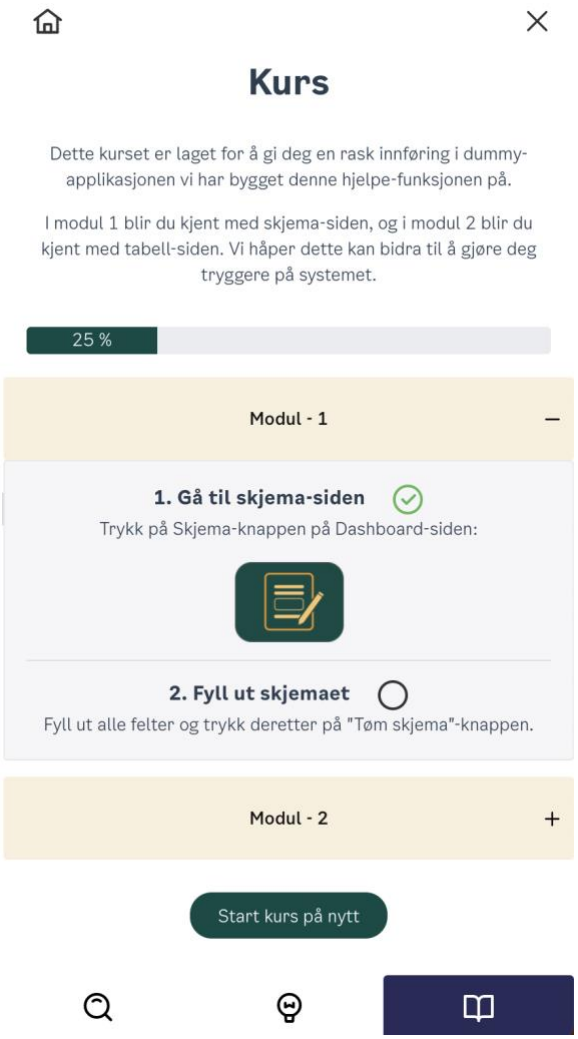
«Emne» feltet er forhåndsutfylt med informasjon om siden som brukeren befinner seg på i dummy-applikasjonen. Dersom brukeren ønsker å sende en feilrapport som ikke har med denne siden å gjøre, kan hen navigere seg til riktig side eller overskrive med ønsket tekst. I motsetning til «Navn» og «E-post» feltene er den forhåndsutfylte teksten i emnefeltet en *placeholder*. Det kommer av at teksten vil oppdatere seg dersom brukeren navigerer i dummy-applikasjonen mens hen fyller ut skjemaet. Her er teksten en *computed*-funksjon som returnerer navnet på siden som brukeren befinner seg på.

Under skjemaet er det en «Send» knapp. Trykk på knappen vil simulere innsending av skjemaet og tømme det. Etter at skjemaet er tømt vil brukeren få en *toast*-varsling om at skjemaet er sendt inn. Toasten er av typen *success*, som gir den en grønn farge og et sjekk-ikon. Varigheten til toasten er satt til 3 sekunder.

Nederst på feilrapportering-siden vises en tekst med informasjon om hva brukeren skal gjøre dersom feilen hen opplever er kritisk.

4.3.6 Kurs-side CSH

På kurs siden kan brukerne gjennomføre et kurs for å bli bedre kjent med funksjonalitetene som applikasjonen tilbyr og hvordan de kan anvendes. I komponenten er det implementert to moduler med to deloppgaver hver. Når kurssiden åpnes, er modulene lukket. Ved trykk på en modul, vil dens deloppgaver bli synlig. En deloppgave består av en overskrift, en forklarende tekst og evt. et bilde. En progresjonslinje viser progresjonen i kurset, der hver deloppgave dekker 25% av progresjonslinjen slik at linjen er 100% fylt når alle fire oppgavene er gjennomført. Brukerne har også mulighet til å nullstille progresjonen ved å trykke på «Start kurs på nytt» knappen (se figur 27).



Figur 27 - Kursmodul

For å holde orden på progresjonen er Vuex tatt i bruk for å lagre tilstanden til kursmodulene og statusen til deloppgavene. Hver modul har én *boolean*-variabel som reflekterer om modulen er gjennomført, og én *boolean* som reflekterer om modulen er åpnet eller lukket. En modul må være åpen for at dens deloppgaver skal kunne godkjennes. Videre har hver deloppgave en *boolean* som reflekterer om oppgaven er gjennomført. Når en oppgave er gjennomført vil et status-ikon bli grønt og få et sjekk-ikon i seg. Når begge deloppgavene til en modul er godkjent, vil fargen på modulen skifte fra lys beige til lys grønn.

Progresjonslinjen har også en variabel i Vuex av typen *number*. For å øke progresjons-variabelen må «increment» metoden i kurs-komponenten gjøre en *commit* til «increment» mutasjonen i Vuex. Mutasjonen øker så progresjons-variabelen og

progresjonslinjen oppdateres automatisk. Det gjøres ved hjelp av en *computed*-funksjon som henter verdien til variabelen fra Vuex.

I modul 1 oppgave 1 skal brukeren navigere seg fra dashboard-siden til skjema-siden. En *watch*-metode følger med på hvilken side brukeren befinner seg på i dummy-applikasjonen. Variabelen «path», som blir sendt fra CSH-*view* til kursmodulen som en *prop*, inneholder navnet på den respektive siden. Dersom modul 1 er åpen og brukeren navigerer seg til «Form», vil oppgavens *boolean*-variabel i Vuex bli satt til *true* og status-ikonet oppdateres. «Increment» metoden, blir så kalt for at progresjonslinjen skal oppdateres.

I modul 1 oppgave 2 skal brukeren fylle ut skjemaet på skjema-siden, for så å tømme det. I dette tilfellet er det skjema-komponenten som står for oppdatering av oppgavens status-variabel i Vuex. Når brukeren trykker på «Tøm skjema» knappen vil det sjekkes at alle feltene er fylt ut før oppgavens *boolean*-variabel i Vuex blir satt til *true* og «increment» metoden blir kjørt.

I modul 2 oppgave 1 skal brukeren navigere seg fra dashboard-siden til tabell-siden. Logikken bak denne oppgaven er lik logikken bak modul 1 sin oppgave 1.

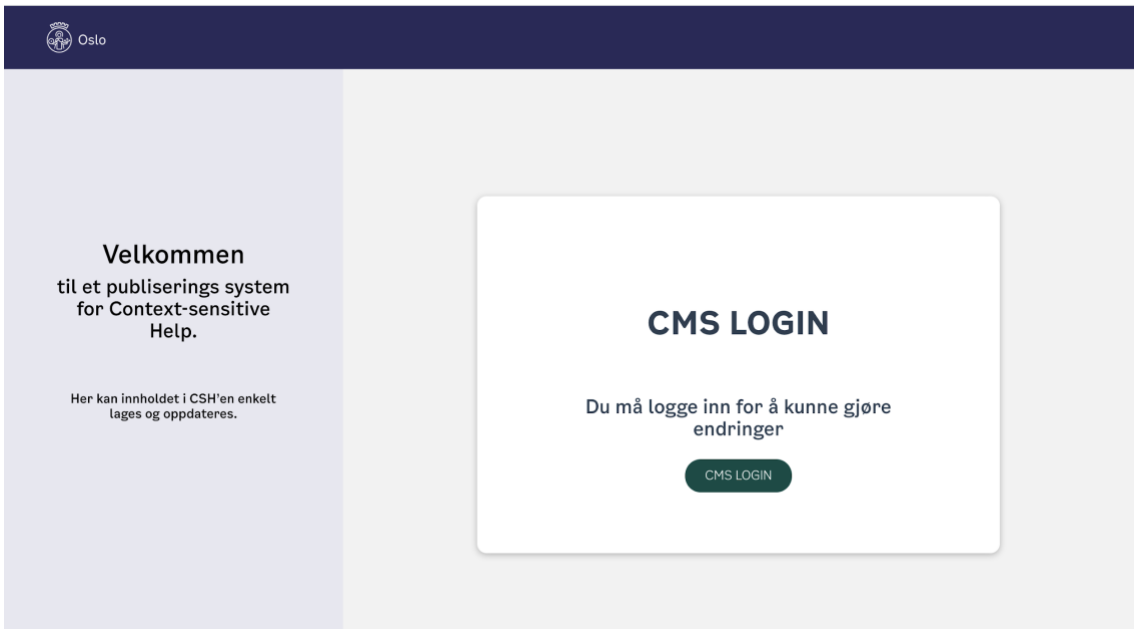
I modul 2 oppgave 2 skal brukeren sortere tabellen på tabell-siden. Brukeren blir bedt om å trykke på AnsattID-feltet for å se at tabellen sorteres stigende på AnsattID. I dette tilfellet er det tabell-komponenten som står for oppdatering av oppgavens status-variabel i Vuex. Når «sortBy» funksjonen, som står for sorteringen av tabellen, blir kalt, sjekkes det om riktig kolonne sorteres. Dersom det stemmer, vil oppgavens *boolean*-variabel i Vuex bli satt til *true* og «increment» metoden blir kjørt.

Kurs siden har også en knapp som lar en bruker tilbakestille all progresjon gjort i kurs så langt. Siden alle verdier som jobbes mot på denne siden er lagret i Vuex-store, gjør det jobben med å tilbakestille verdiene enklere. Ved trykk på knappen kalles «clear» funksjonen. Når denne kjøres vil brukeren kunne se en varslingsboks, som ber hen bekrefte handlingen hen er i ferd med å gjennomføre. Hvis brukeren velger å gjennomføre handlingen vil det sendes et kall til «clear» mutasjonen i Vuex, som resetter alle tilstandene til kursmodulen.

4.4 CMS

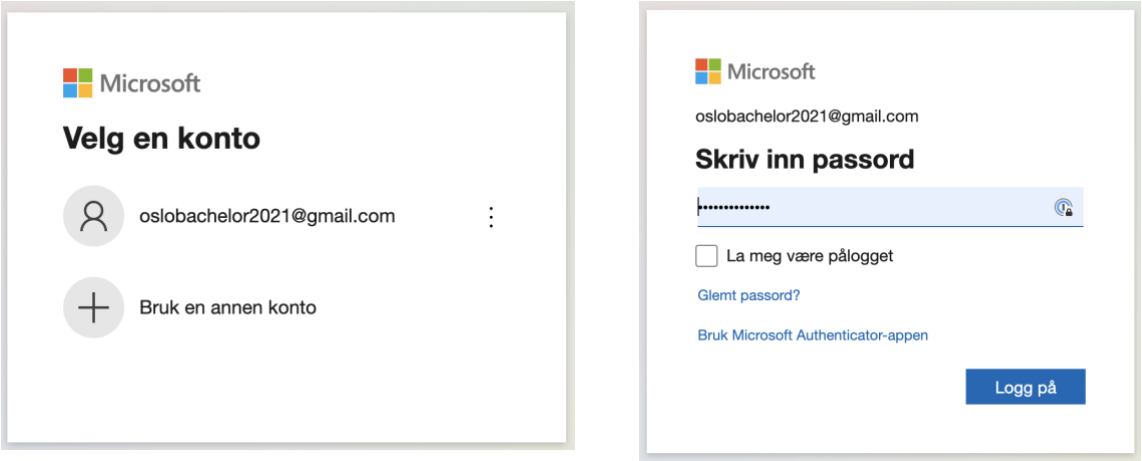
CMS står for Content Management System (publiseringssystem), og i vårt tilfelle er det her innholdsskapere kan legge til, slette eller endre innholdet som vises i CSH'en. Denne delen av applikasjonen krever at brukeren er logget inn for å få tilgang til funksjonene.

4.4.1 Logg inn



Figur 28 - Innloggingsside for CMS

Bruker kommer til denne siden ved å skrive */cms* bak base URL'en. Visuelt består logg inn siden av tekstlig informasjon og en knapp (se figur 28). For å logge inn må brukeren trykke på «CMS login» knappen. Her kalles «login» metoden, som deretter kaller på en Vuex-action som også heter «login». Når denne funksjonen kjøres, vil bruker se et Microsoft popup-vindu for innlogging (se figur 29).

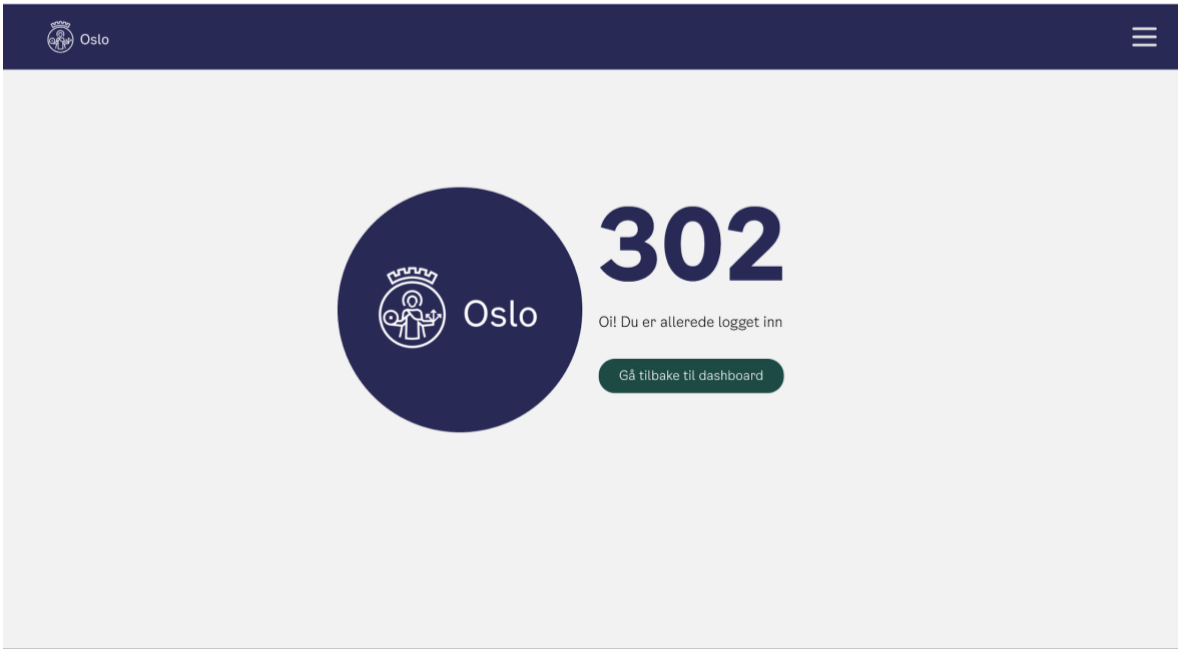


Figur 29 - Microsoft innloggingsvindu

```
actions: {
  // Azure msal login
  async login(context) {
    try {
      const loginRequest = {
        scopes: ["user.read"]
      };
      const loginResponse = await msalInstance.loginPopup(loginRequest);
      if (loginResponse.accessToken) {
        context.commit("setAuth", { isAuth: true });
        context.commit("setAccessToken", {
          token: loginResponse.accessToken
        });
      }
    } catch (error) {
      console.error(error.message);
    }
  },
  // Azure msal logout
  logout(context) {
    msalInstance.logout();
    context.commit("setAuth", { isAuth: false });
  }
},
```

Figur 30 - Vuex actions som bruker MSAL

Når logg inn er gjennomført vil bruker bli returnert til dashboard-siden i dummy-applikasjonen, hvor et ikon for nedtrekksmeny nå vil være tilgjengelig helt til høyre i navigasjonsbaren på toppen av siden. Ikonet er bare tilgjengelig for innholdsskapere. Dersom brukeren forsøker å gå til innloggingssiden på nytt, vil hen bli omdirigert til en feilside (se figur 31), der hen igjen får mulighet til å navigere tilbake til dashboard-siden.



Figur 31 - Allerede logget inn side

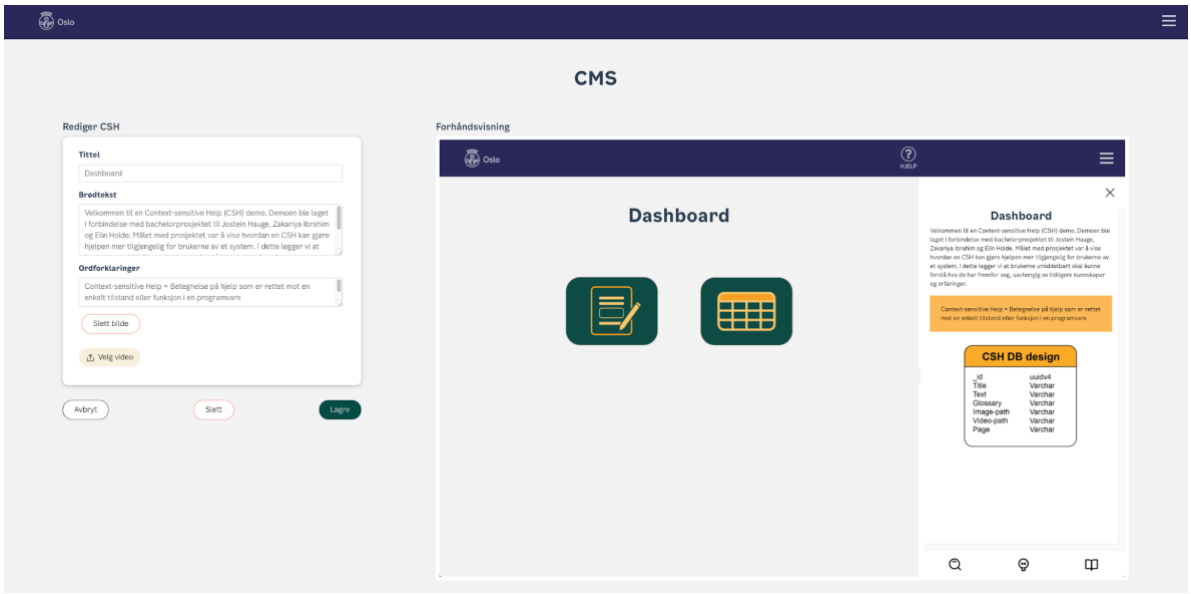
4.4.2 Logg ut

Logg ut knappen er definert i nedtrekksmenyen. Når knappen trykkes, vil «logout» funksjonen kalles. Funksjonen kaller på en Vuex-action med navn «logout». Funksjonen bruker dataene som befinner seg i MSAL-konfigurasjonen (se kap. 5.4) for å vite hvilken applikasjon som det skal logges ut av. Deretter gjøres utlogging gjennom et Microsoft-vindu.

Figur 32 - Microsoft logg ut vinduer

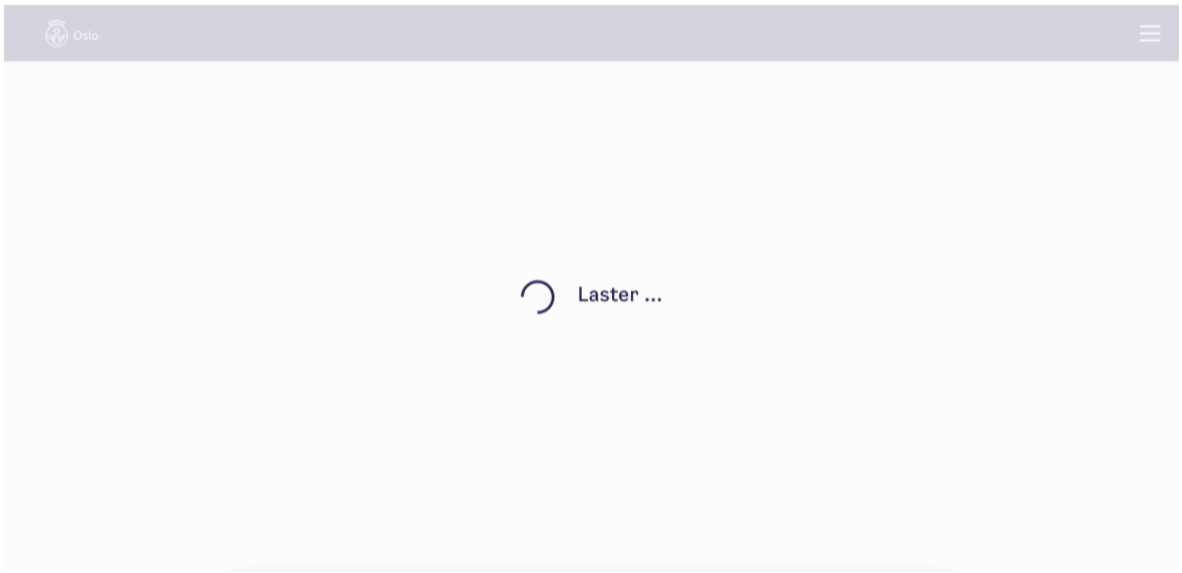
4.4.3 Endre-side

På denne siden kan en innholdsskaper (heretter “bruker”) gjøre endringer på innholdet i CSH’en. Innholdet i CSH’en kan bestå av tittel, brødtekst, ordforklaringer, ett bilde og en video, men kan for eksempel også bare være tekst. Ettersom applikasjonen kun skal brukes til demonstrasjonsformål er det gjort et valg om å ikke gjøre denne delen mer komplisert enn den trenger å være. Poenget er å vise hvordan en CMS kan gjøre det enklere å administrere innholdet i CSH’en. Det vil derfor kun være mulig å legge inn ett felt med brødtekst, ett felt med ordforklaringer, ett bilde og en video, selv om det i et produksjons klart produkt burde være mulig å legge inn flere. Det vil heller ikke være mulig å gjøre om på rekkefølgen til de ulike feltene.



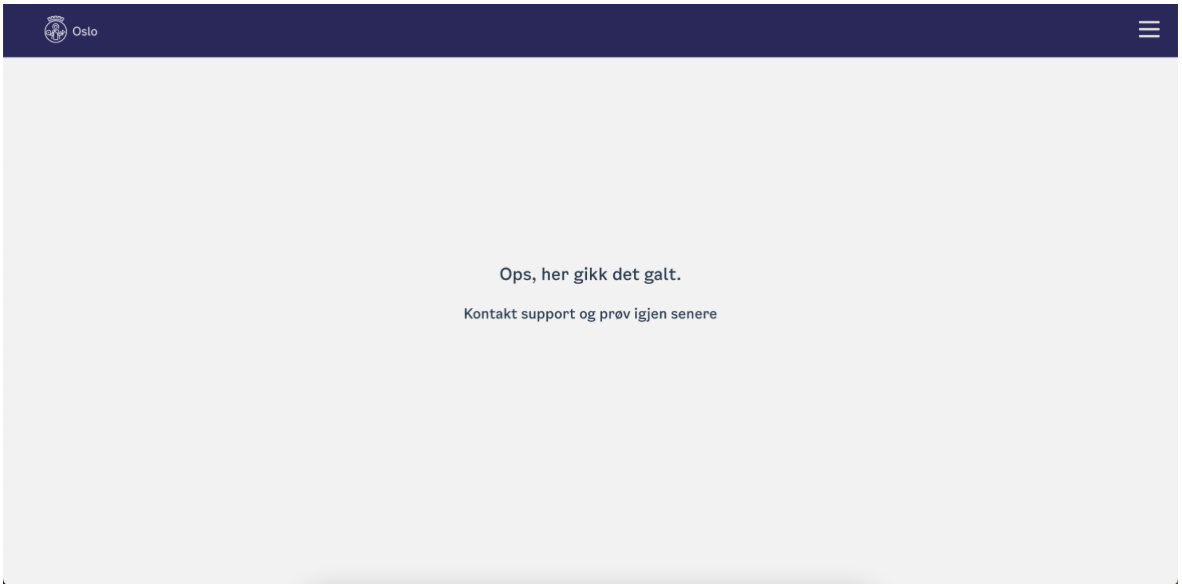
Figur 33 - Endre-siden

For å gjøre endringer på innholdet i CSH’en må brukeren stå på siden som hen ønsker å endre, deretter trykke på «Endre siden» knappen i nedtrekksmenyen i navigasjonsbaren. Endre-siden vil da åpnes og «onMounted» funksjonenen til komponenten vil kalle «getData» funksjonen. Et HTTP-kall ser etter innhold tilhørende valgt side og hvis mulig returnerer innholdet tilbake til funksjonen. Mens dette skjer vil en laster-skjerm vises for å fortelle brukeren at innlasting foregår i bakgrunnen (se figur 34).



Figur 34 - Laster-skjerm

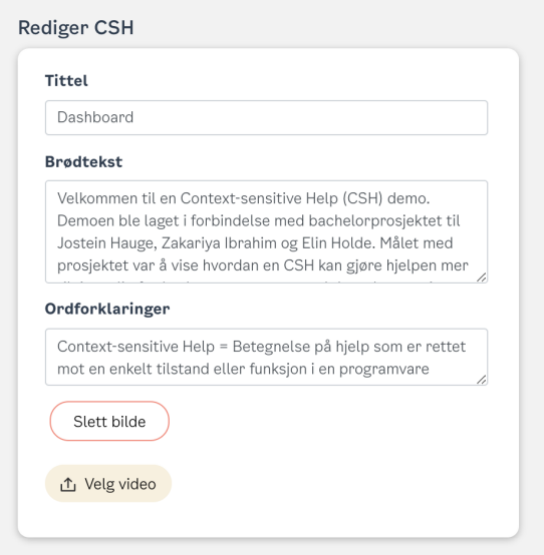
Når innholdet er ferdig lastet inn, forsvinner laster-skjermen. Hvis det eksisterer innhold tilhørende valgt side, vil innholdet plasseres i sine respektive felter i skjemaet og vises i forhåndsvisningen. Hvis innhold ikke eksisterer, vil feltene i skjemaet være tomme og i forhåndsvisningen vil det stå “Forhåndsvisning kommer her”. Går noe galt under innlastingen, vil en feil-skjerm som forteller at noe gikk galt vises i stedet (se figur 35).



Figur 35 - Feil-skjerm

4.4.3.1 Skjemaet

På skjema-delen av siden kan brukeren lage innhold eller endre på eksisterende innhold. De ulike delene av innholdet har hver sine respektive input-felter. Tittel, brødtekst og ordforklaringer har tekstlige input-felter som brukeren kan skrive i. Bilde og video kan lastes opp ved hjelp av to filvelgere, en for bilde og en for video. Innholdet vil lagres i «formData» objektet til komponenten. Objektet er av typen *reactive*, og det er koblet til feltene ved hjelp av *v-model*, slik at både feltene og objektet vil være oppdatert til enhver tid. Endringer i skjemaet vil også gjenspeiles i forhåndsvisningen (les mer om den under kap. 4.4.3.2 Forhåndsvisning).



Figur 36 - Skjema-delen

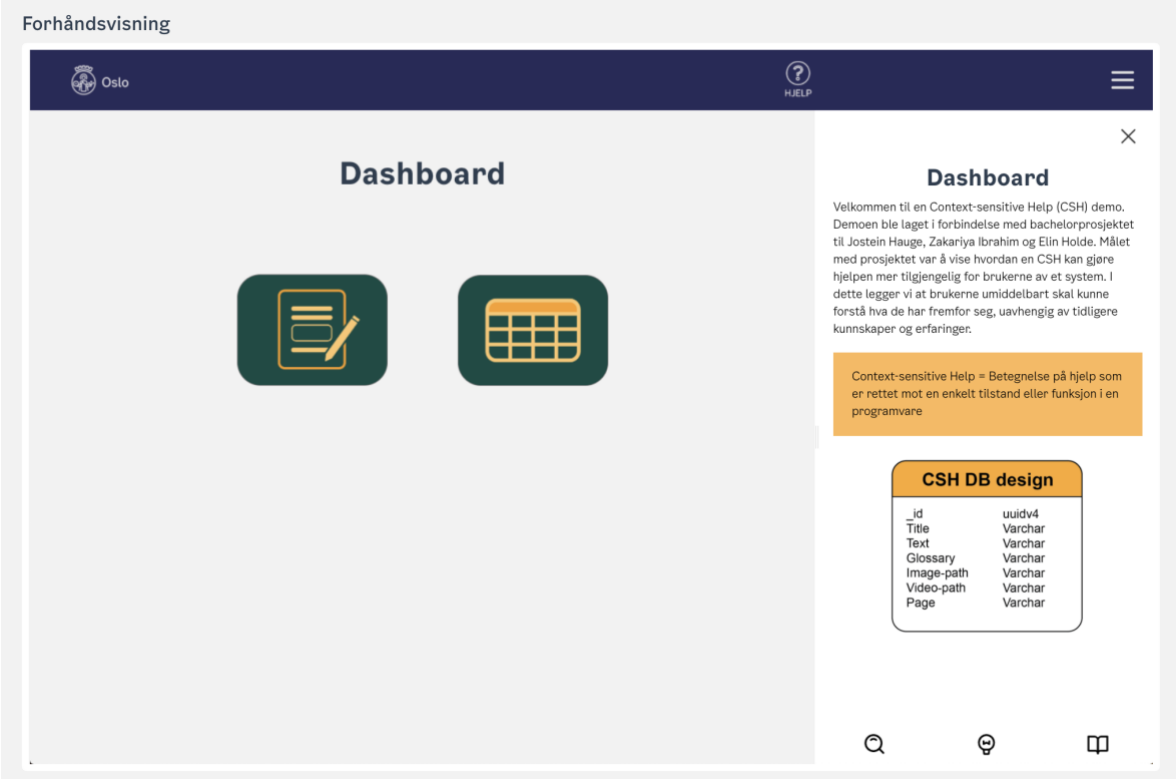
Filvelgerene til bilde og video fungerer likt foruten at bilde-velgeren krever en bilde-filtype og video-velgeren krever video i mp4-format. Under opplasting av video-filen vises det i tillegg «laster-skjermen», ettersom en video-fil kan ta lengre tid å laste opp. Når «velg bilde» eller «velg video» knapp trykkes, åpnes en standard html filvelger som lar brukeren velge blant sine lokale filer. En *onChange*-metode starter «onImgFileSelected» eller «onVideoFileSelected» funksjonen når brukeren har valgt en fil. Hvis en ugyldig fil velges, vil brukeren få beskjed gjennom en *toast*-varsling av typen *warning*. Er filen gyldig, vil den lastes opp til Firebase ved hjelp av et Firebase bibliotek. Dersom opplasting er vellykket laster funksjonen ned en URL til filen, som så lagres i «formData» objektet. Brukeren får da beskjed gjennom en *toast*-varsling av typen *success*. Går noe galt under opplastingen vil brukeren få beskjed gjennom en *toast*-varsling av typen *error*.

Hvis det er lagt inn bilde eller video vil filvelgeren erstattes med slette knapper («Slett bilde» og «Slett video»). Når en slett-knapp trykkes vil «clearImage»/«clearVideo» funksjonen kjøres og bilde/video fjernes fra «formData» objektet. Bildet/videoen vil

dermed også forsvinne fra forhåndsvisningen. Slettingen vil ikke påvirke databasen før «lagre» knappen trykkes, og brukeren har derfor mulighet til å angre handlingen ved å trykke på «avbryt» knappen (les mer om knappene under kap. 4.4.3.3). Bilde/videoen vil heller ikke slettes fra Firebase.

4.4.3.2 Forhåndsvisning

Forhåndsvisningen er ment å vise hvordan innholdet vil bli seende ut i CSH'en. Den består av et bakgrunnsbilde av dummy-siden som det skal lages innhold til. Bildet er hardkodet inn og kilden ligger i *assets* under *src* i prosjektet. I et produksjons klart produkt vil det måtte løses på en annen måte. Vi har valgt å gjøre det på denne måten ettersom dummy-applikasjonen kun inneholder tre sider, og produktet kun skal brukes til demonstrasjonsformål.



Figur 37 - Forhåndsvisning-delen

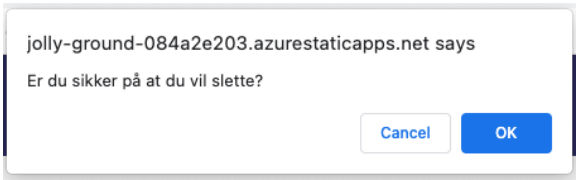
Over høyre del av bakgrunnsbildet, der CSH'en er, vil innholdet i skjemaet posisjoneres slik det vil bli seende ut i CSH'en. Innholdet oppdaterer seg dynamisk, slik at det som skrives i input-feltene til skjemaet vil fortløpende bli synlig i forhåndsvisningen. Forhåndsvisningen mottar innholdet i alle input-feltene gjennom *props*. En *watch*-metode følger med på *props*-objektet, og når det forandrer seg vil «updatePreview» funksjonen kalles. Funksjonen har en *if*-setning som sjekker om verdiene i *props* er tomme. Er feltene tomme, vil *boolean*-variabelen «isPreview» settes til å være *false*. I *template*-delen av komponenten brukes variabelen i en *v-if*, slik at når «isPreview» er *true* blir innholdet i *props*-objektet vist. Er «isPreview» *false* vil en "Forhåndsvisning kommer her" tekst vises i stedet.

4.4.3.3 Avbryt, slett og lagre

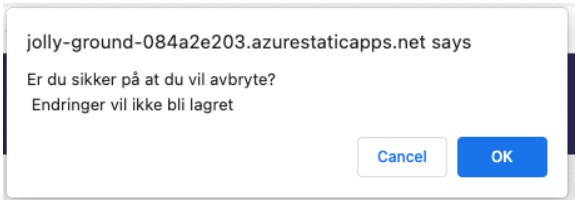
De tre knappene «Avbryt», «Slett», og «Lagre» ligger i en egen komponent kalt «CmsButtonBar». Avbryt for å avbryte endringer, slett for å slette innholdet og lagre for å lagre innholdet.

Figur 38 - «CmsButtonBar»

Når «Avbryt» knappen trykkes åpnes en standard varslings boks som spør om brukeren er sikker på at hen vil gjennomføre handlingen. Hvis "ok" trykkes vil alle endringer som er gjort siden forrige lagring bli slettet og brukeren blir navigert tilbake til dashboard-siden.



Figur 39 - Slett varslingsboks



Figur 40 - Avbryt varslingsboks

Når «Slett» knappen trykkes åpnes en standard varslings boks som spør om brukeren er sikker på at hen vil gjennomføre handlingen. Hvis “ok” trykkes vil «deleteData» funksjonen gjøre et HTTP-kall mot databasen, der innholdets ID blir brukt for å slette innholdet. Innholdet vil så forsvinne fra forhåndsvisningen og input-feltene i skjemaet, og bruker får en varsling i form av en *toast* av typen *success*. Dersom innholdet av en eller annen grunn ikke kunne slettes fra databasen vil brukeren få beskjed gjennom en *toast* av typen *error*.

Når «Lagre» knappen trykkes vil innholdet (eller endringer på innholdet) bli lagret til databasen. «saveObject» funksjonen sjekker om innholdet skal lagres som et nytt objekt eller om et allerede eksisterende objekt skal oppdateres i databasen. En «Status» *boolean* blir satt til *false* i «getData» funksjonen dersom det ikke eksisterer noe objekt i databasen når innholdet skal lastes inn. Er «Status» *false* betyr det at et nytt objekt må lagres og dermed blir «createData» funksjonen kjørt. Er «Status» *true* betyr det at et objekt skal oppdateres og dermed blir «updateData» kjørt.

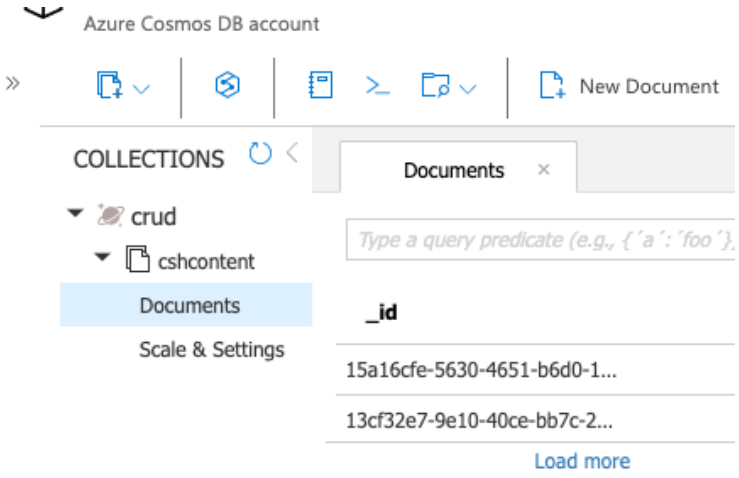
Når «CreateData» funksjonen kalles vil «formData» objektet sendes i et POST-kall, som vil si at et nytt objekt skal legges til i databasen. Når «UpdateData» funksjonen kalles vil «formData» objektet og innholdets ID sendes i et PUT-kall, som vil si at et allerede eksisterende objekt skal oppdateres. Dersom handlingen er vellykket, vil status-kode 200 returneres. Brukeren får beskjed om at handlingen er gjennomført i form av en *success toast* og hen blir navigert til dashboard. Går noe galt under lagringen vil brukeren få beskjed i form av en *error toast*. Er manglende internett-tilgang grunnen til feilen vil brukeren få spesifikt beskjed om dette. Andre typer feil vil gi en standardisert feilmelding.

5 Backend

Applikasjonen vi har laget har ingen lokal backend, det vil si at all backend-kode befinner seg i Azure, som er vår valgte løsning for BaaS (backend as service). Systemets backend er satt sammen av flere deler, hvor databasen, autentisering, hosting og Azure-funksjonene ligger på Azure. Den siste delen av backenden er Firebase Cloud Storage, som brukes for å lagre bilder og videoer.

5.1 Database

For lagring av data til applikasjonen har vi valgt å benytte MongoDB, som er en del av en Azure service som heter CosmosDB. Det fordi den er gratis opp til 25GB data, og vil dekke alle våre behov for applikasjonen. Databasen er av typen noSQL dokumentdatabase, som vil si at data blir lagret i JSON-lignende dokumenter. Hvert dokument blir lagret i en samling (*collection*). Eksempelvis er en samling i en noSQL database sammenlignbart med en tabell i SQL-databaser. I vårt tilfelle heter databasen «crud», og i den ligger samlingen, som hos oss heter «cshcontent». I samlingen vil dokumentene lagres (se figur 41).



Figur 41 - MongoDB struktur

Dataene som lagres i databasen er følgende: En autogenerert ID, tittel, tekst, ordforklaringer, bilde-link, video-link og side (se figur 42). Her er det bare video og bilde linker som blir gjort forhåndsarbeid på før vi sender objektet fra frontend til databasen, se mer om denne operasjonen under kapittel 5.2.

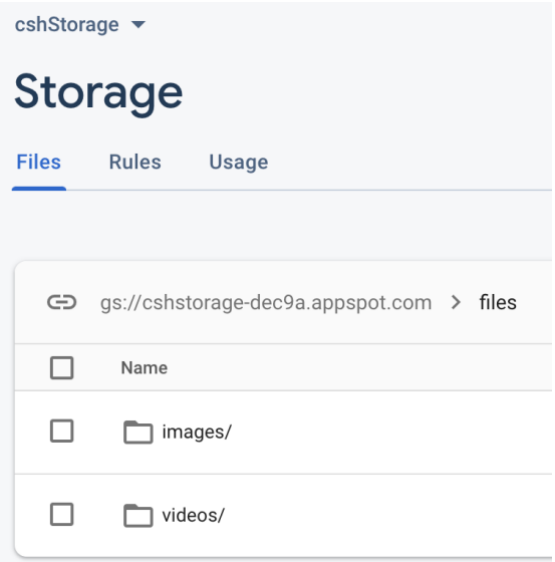

```
{
    "_id" : "13cf32e7-9e10-40ce-bb7c-278c5f72124e",
    "title" : "Dashboard",
    "text" : "Velkommen til en Context-sensitive Help (CSH) demo. Demoen ble la
    "glossary" : "Context-sensitive Help = Betegnelse på hjelp som er rettet mo
    "imgpath" : "https://firebasestorage.googleapis.com/v0/b/cshstorage-dec9a.a
    "videopath" : "",
    "page" : "Dashboard"
}
```


Figur 42 - Eksempel av data som lagres

5.2 Firebase skylagring

For lagring av bilder og videoer ble det først sett på Azure Blob Storage. Ettersom det var vanskelig å finne dokumentasjon på hvordan man kunne bruke Blob Storage med Vue 3, valgte vi å gå for Firebase Cloud Storage, som vi hadde kjennskap til fra før.

I Firebase er bilder og videoer sortert i hver sine mapper under «Files» (se figur 43). Filstien til bildene er *files/images* og til videoer er den *files/videos*. Filstien må brukes når man i frontend-delen av applikasjonen skal koble til Firebase for å laste opp en fil. Når et bilde eller en video er lastet opp til Firebase, returneres en URL til filens lokasjon. URL'en vil bli lagret sammen med tilhørende informasjon i databasen i Azure når brukeren trykker en «lagre» knapp. Når applikasjonen henter data fra databasen vil URL'en bli brukt som kilden til bilde/video i CSH'en.



Figur 43 - Firebase skylagring filstruktur

5.3 Azure Functions

Azure Functions kan brukes for å bygge serverløse applikasjoner gjennom enkel kode som kan trigges til å gjøre spesifikke operasjoner. Løsningen krever mindre kode enn om man skulle bygget infrastrukturen fra bunn, noe som igjen gjør det enklere å vedlikeholde. Azure Functions kan for eksempel brukes til å prosessere filer som lastes opp, kjøre planlagte rutiner, respondere på endringer i en database osv. (Microsoft, 2020). I vårt tilfelle brukes funksjonene til å gjøre CRUD-operasjoner mot MongoDB databasen.

Name	Trigger	Status
create-list-item	HTTP	Enabled
delete-list-item	HTTP	Enabled
get-list	HTTP	Enabled
get-list-item	HTTP	Enabled
initialize-list	HTTP	Enabled
update-list-item	HTTP	Enabled

Figur 44 - Oversikt over Azure funksjonene

Funksjonene skrives i et utviklingsverktøy i Azure portalen. Verktøyet har støtte for en rekke programmeringsspråk. Eksempler inkluderer C#, JavaScript, TypeScript og Python (Microsoft, 2019). I vårt tilfelle er funksjonene skrevet i JavaScript. Hver funksjon består av to filer; en *index.js* fil og en *functions.json* fil. Index inneholder logikk for å kommunisere mot databasen og functions står for konfigurasjonen av funksjonen, som i vårt tilfelle består av å definere bindingene til HTTP triggerne.

cshcrud \ create-list-item \ index.js

```
1   const { MongoClient } = require("mongodb");
2   const { v4: uuidv4 } = require("uuid");
3
4   const url = "mongodb://oslobachelortestdb:hOMcNx8J6ev2IMfnc3eOCXzsZWq
5   const client = new MongoClient(url);
6
7
8   module.exports = async function (context, req) {
9       await client.connect();
10      const database = client.db("crud");
11      const collection = database.collection("cshcontent");
12      let data = { _id: uuidv4(), ...req.body };
13      await collection.insertOne(data);
14
15    return (context.res = {
16      status: 200,
17      body: data,
18    });
19  };
```

Figur 45 - Eksempel på *index.js* fil hentet fra «Create-list-item» funksjon

I figur 45 vises et eksempel på en *index.js* fil fra «Create-list-item» funksjonen. Først importeres nødvendige pakker og kobling til databasen defineres. Under vil selve logikken skrives som en *async* funksjon. Det vi si at de delene av koden som er markert med *await* må gjøres før funksjonen kan kjøre videre. I vårt tilfelle gjelder det operasjonene mot databasen, da disse er vesentlige for videre gjennomføring av funksjonen. Når koblingen er satt opp, finnes det frem til samlingen der dataene skal lagres.

Objektet som skal lagres i databasen ligger i variabelen *req.body*. Denne benyttes sammen med den tilfeldig genererte ID'en, som også er navnet på dokumentet som lagres. Data blir deretter satt inn i databasen ved hjelp av kommandoen *insertOne*, som er en MongoDB kommando. Dersom handlingen er vellykket vil funksjonen returnere statuskode 200, ellers vil Azure sende ut en feilkode som respons på funksjonen.

cshcrud \ create-list-item \ function.json

```
{
  "bindings": [
    {
      "authLevel": "anonymous",
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "route": "cshcontent",
      "methods": [
        "post"
      ]
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    }
  ]
}
```

Figur 46 - Eksempel på *function.json* fil hentet fra «Create-list-item» funksjon

I figur 46 vises *function.json* tilhørende «Create-list-item» funksjonen. Filen er en konfigurasjon som forteller hvordan *index.js* skal brukes. I dette tilfelle er funksjonen av typen *httpTrigger*, som vil si at koden i index-filen startes ved kall mot URL. POST-metoden indikerer at et objekt vil bli sendt med URL'en, med endepunkt som tilsvarer rute-variabelen "cshcontent". Videre indikeres de to retnings variablene (*direction*) inn og ut: "req" for *request* på inn-variabelen og "res" for *response* på ut-variabelen. Variablene benyttes i *index.js* filen.

5.4 Autentisering

Azure Active Directory (Azure AD) er et verktøy for identitet- og tilgangsstyring (IAM). Verktøyet kan brukes til å administrere autentisering og autorisering i både eksterne og interne ressurser (Microsoft, 2020). I vårt tilfelle benyttes Azure AD under innlogging i CMS delen av applikasjonen. Applikasjonen hvor autentiseringen skal implementeres registreres, for å få en *client ID* og en *tenant ID.*

MSAL benyttes for å knytte Azure AD til applikasjonen. I rot-komponenten til applikasjonen (*main.ts*) settes en konfigurasjon opp til å inneholde *client ID* og *tenant ID* fra Azure AD (se figur 47). Autoritets-taggen i konfigurasjonen inneholder URL'en til Microsoft sin innloggingsportal, som må inkludere *tenant ID.*

```
const msalConfig = {
  auth: {
    clientId: "b4b3d62f-43eb-4013-a40e-8e9431f558c3",
    authority:
      "https://login.microsoftonline.com/ee07e5f1-6968-4a03-aefa-71cfca69c820/",
    redirectUri: window.location.origin + "/",
    requireAuthOnInitialize: false,
    postLogoutRedirectUri: window.location.origin + "/"
  },
  request: {
    scopes: [`user.read`]
  },
  graph: {
    callAfterInit: true
  },
  cache: {
    cacheLocation: "localStorage"
  }
};
```

Figur 47 - MSAL konfigurasjon

5.5 Hosting

Azure Static Web Apps er en tjeneste som automatisk bygger og publiserer applikasjonen på nett etter som koden forandrer seg. Tjenesten tar seg av automatisk fornying av SSL sertifikater. Den har også integrert støtte mot GitHub slik at nettsiden oppdaterer seg når det *pushes* til master *branchen* i prosjektet (Microsoft, 2021). En *continuous deployment* (CD) strategi blir brukt når Azure generer en GitHub Actions *workflow*, i en YAML-fil (Microsoft, 2021). Ved hjelp av tjenesten fikk oppdragsgiver tilgang til siste versjon av applikasjonen gjennom hele utviklingsprosessen. Tjenesten var enkel å sette opp så lenge man hadde de riktige rettighetene til å gjøre det. Rettighetene innebærer:

- Bruker har en github konto som er koblet opp mot riktig programvarelager (*repository*).
- Bruker har tillatelse fra bedrift som eier *repository* til å la tredjepart app (Azure) få tilgang, i dette tilfellet kalt organisasjons aksess.

Ettersom *hostingen* er i et gratis abonnement, er det kun tilgjengelig båndbredde opp til 100 GB i måneden. Etter båndbredden er brukt opp vil Azure stoppe hostingen av siden.

6. Design

6.1 WCAG

WCAG (Web Content Accessibility Guidelines) er en samling retningslinjer laget av W3C (The World Wide Web Consortium). Hensikten med retningslinjene i WCAG er å gjøre nettsider tilgjengelige for alle, også mennesker med funksjonshemninger (Henry, 2021). Brukere med funksjonshemninger bruker ofte teknologiske hjelpemidler, og for at disse hjelpemidlene skal kunne tilby brukeren best mulig hjelp er det viktig at webapplikasjoner er tilpasset disse. Videre følger forklaringer og begrunnelser for de ulike punktene i WCAG sine retningslinjer (W3C, 2019).

6.1.1 Prinsipp 1: Mulig å oppfatte

Punkt 1.1 dekker alt av ikke-tekstlig innhold. For at skjermlesere skal kunne gjenfortelle innhold i applikasjonen som ikke er tekstlig, skal alle slike elementer ha et tekstlig alternativ (W3C, 2019). Alle applikasjonens bilder har derfor en alt-attributt som forteller hva bilde viser. Knapper og andre elementer som for eksempel ikonér har enten aria-label eller aria-labelledby.

Punkt 1.2 dekker tidsbaserte medier, som innebærer lyd- og videoklipp. Denne typen medier skal ha undertekster som beskriver det som sies, hvem som sier det, samt andre lyd-effekter, musikk, latter osv (W3C, 2019). Et transkript som beskriver alt som sies, gjøres og vises skal også være tilgjengelig for å innfri nivå A og norsk lov. De resterende underpunktene, som går på alternative lyd-spor, undertekster for live-video og tegnespråk, er ikke omfattet av forskriften. Uavhengig av dette er det her opp til skaperen av innholdet å følge disse retningslinjene. Videoen som er brukt i CSH'en til demonstrasjonsformål inneholder ingen form for lyd, og er et alternativ til den tilgjengelige teksten.

Punkt 1.3 handler om hvordan innholdet på siden kan tilpasses og forståes på forskjellige måter uten å miste noe informasjon (W3C, 2019). For å oppnå dette er det viktig å sørge for en semantisk struktur på koden. HTML elementer er brukt til det de er ment til og CSS er ikke brukt til å endre rekkefølgen eller style et element til å se ut

som et annet. Videre er innholdet tilpasset både landskap- og portrettmodus, og hensikten med input-feltene er gjort åpenbare ved hjelp av etiketter og type-attributter. Alle nivåer er dermed dekket på dette punktet.

Punkt 1.4 handler om å gjøre innhold enkelt å skille fra hverandre, inkludert lyd (W3C, 2019). Dersom en skjermleser brukes og det spilles av lyd fra applikasjonen samtidig, kan det være vanskelig for brukeren å skille lydene fra hverandre. Ettersom vår applikasjon ikke spiller av noen form for lyd automatisk anses denne delen som irrelevant. De aller fleste underpunktene omhandler likevel størrelse og format på tekst, fargekontrasten mellom tekst og bakgrunn og annen bruk av farger. Der farge er brukt for å formidle informasjon, er også tekst brukt for å støtte formidlingen. For eksempel i fremdriftslinjen på kurssiden er prosentandelen av fremdriften også presentert (se figur 48).

Figur 48 - Eksempel på at ikke bare farge er brukt for å formidle informasjon

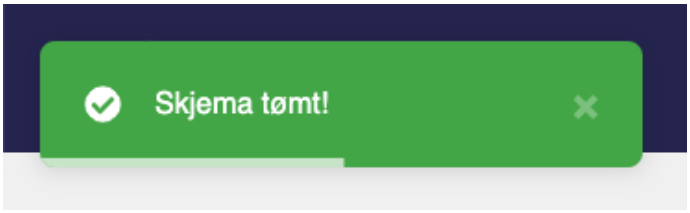
Videre er det gjennomgående valgt fargekontraster som innfrir kravet til punkt 1.4.6 nivå AAA, og det er sørget for at all tekst kan skaleres opp til 200% uten at den forsvinner eller mister sin funksjon. Når det gjelder bilder av tekst er bakgrunnen i forhåndsvisningen i CMS'en et bilde som inneholder tekst. Bilde er et skjermbilde som

visuelt formidler informasjon gjennom mer enn bare tekst og teksten er i seg selv ikke relevant i denne konteksten. Bilde er dermed omfattet av nivå AA sine unntak, men tilfredstiller ikke AAA der ingen unntak er godkjent. Også punkt 1.4.8, som handler om den visuelle presentasjonen av tekst på nivå AAA, er ikke tilfredsstilt ettersom funksjonalitet for endring av forgrunn- og bakgrunnsfarge mangler.

6.1.2 Prinsipp 2: Mulig å bruke

Punkt 2.1 dekker bruk av tastaturet. Brukere som er avhengig av tastaturet for å navigere seg rundt i applikasjonen skal ha like gode muligheter som andre (W3C, 2019). Det er sørget for at alle handlingsbare elementer kan tabbes til og fra. På skjemaer er det brukt standard skjema kontroller, med innebygget funksjonalitet for tabbing på lik linje med knapper og lenker så lenge man har brukt <button> og <a> elementer. Det siste punktet, punkt 2.1.4 handler om bruk av hurtigtaster, noe som ikke er implementert i vår applikasjon. Alle nivåer er dermed dekket på dette punktet.

Punkt 2.2 handler om at brukere skal ha nok tid til å lese og bruke applikasjonens innhold. Dersom innhold starter automatisk og foregår parallelt med annet innhold, eller har en tidsbegrensning skal det være mulig å enten skru den av eller justere den (W3C, 2019). Eneste innholdet i applikasjonen vår som har tidsbegrensing er *toast* meldingene som dukker opp i venstre hjørnet. Varslingene pauses når de holdes over og kan fjernes ved å trykke på krysset (se figur 49). Ideelt sett skulle det vært lagt bedre til rette for personer med nedsatt kognitive og motoriske egenskaper å stille inn tidsbegrensningen på forhånd. Når det er sagt, vil en skjermleser fortsette å lese varslingen etter tiden er gått ut og vi så derfor ingen grunn til å bruke mer tid på problemstillingen, selv om det betyr at vi ikke skårer fullt ut på dette punktet.



Figur 49 - Varsling med tidsbegrensning

Punkt 2.3 handler om å tilpasse innholdet til de som kan få anfall eller andre fysiske reaksjoner som følge av blinkende elementer (W3C, 2019. Applikasjonen vår inneholder ingen elementer som faller inn under dette punktet, og kravet er dermed tilfredsstilt.

Punkt 2.4 handler om at det skal være enkelt for brukeren å navigere i applikasjonen, samt kunne vite hvor hen er til enhver tid (W3C, 2019). Også her vil en sekvensiell og semantisk struktur på koden, der man bruker HTML elementer til det de er ment til, gjøre siden enklere å navigere i. For eksempel har vi brukt <header> elementet rundt navigasjonsbaren og <main> elementet rundt resten av innholdet, slik at brukeren kan velge å hoppe over navigasjonsbaren. Videre har applikasjonen en tittel, overskrifter, etiketter og linker som beskriver dens formål. Dersom tabbing brukes for å navigere, vil elementer med fokus indikeres ved en blå lysene linje rundt objektet. Eneste punktet som mangler for å innfri nivå AA er punkt 2.4.5, som handler om at det skal være flere veier til en side. Ettersom dummy-applikasjonen skal brukes til demonstrasjonsformål ble dette nedprioritert.

Punkt 2.5 dekker brukers input utover tastatur. Her er det ikke mange punkter som er relevante for vår applikasjon (W3C, 2019). Punkt 2.5.2 som omhandler klikk-avbrytelse er ivaretatt gjennom bruk av HTML elementene <button> og <a>, da funksjonen blir fullført når man slipper trykket og den kan avbrytes ved å fjerne pekeren fra objektet før man slipper trykket. Også punkt 2.5.3 som sier at objekters tekst skal samsvare med dens etikett-navn er ivaretatt. Alle nivåer er dermed dekket på dette punktet.

6.1.3 Prinsipp 3: Forståelig

Punkt 3.1 handler om å gjøre innholdet leselig (W3C, 2019). Det er kun underpunkt 3.1.1 som er nivå A. For å møte dette punktet er lang-attributten brukt på HTML-elementet, slik at applikasjonens primære språk er tilgjengelig for skjermlesere. Dersom ulike språk er brukt, skal lang-attributten plasseres på avsnitt eller fraser der annet enn det primære språket er brukt, men dette forekommer ikke i vår applikasjon. Videre underpunkter på nivå AAA omhandler mekanismer for å forklare uvanlig ord, forkortelser og tvetydige uttalelser. Disse punktene er ikke blitt prioritert grunnet tid.

Punkt 3.2 handler om at applikasjonen skal fungere på en forutsigbar måte (W3C, 2019). For at det skal være mulig å bruke tastaturet til å tabbe seg forbi elementer som kan utløse en forandring i konteksten, vil dette ikke skje når elementet får fokus. Det er ikke en forventet oppførsel av denne typen elementer og det kan forvirre brukeren dersom hen for eksempel blir sendt til en ny side. Konteksten vil heller ikke forandre seg som en følge av input fra brukeren. For eksempel blir ikke applikasjonens skjemaer automatisk sendt inn når input er utfylt. Brukeren kan selv velge når denne typen handlinger skal skje, noe som støtter forutsigbarhet i applikasjonen. Videre er repeterende funksjonaliteter presentert i samme rekkefølge og på samme måte på alle sider, og forandringer i konteksten forekommer kun på forespørsel av bruker. Alle nivåer er dermed dekket på dette punktet.

Punkt 3.3 handler om å hjelpe brukeren til å unngå feil ved utfylling av input-felter, samt rettelse av eventuelle feil som blir gjort (W3C, 2019). I vår applikasjon er det grunnet demonstrasjonsformålet ikke implementert funksjonalitet utover tømming av feltene i noen av skjemaene. Av den grunn er det heller ikke brukt tid på validering av input-felter og feilmeldinger i tilknytning til skjemaene. Likevel er det brukt tid på informative etiketter til alle input-felter og CSH'en tilbyr noe veiledning i bruk av skjemaet på skjema-siden, slik at punkt 3.3.2 og 3.3.5 er innfridd.

6.1.4 Prinsipp 4: Robust

Prinsipp 4 har kun ett punkt, punkt 4.1, som handler om at applikasjonen i størst mulig grad skal være kompatibel med potensielle programvarer en bruker måtte benytte seg av, som for eksempel hjelpemidler for funksjonshemmede (W3C, 2019). For å oppnå dette har alle HTML-elementer start og slutt tagger, og de er stablet på riktig måte i henhold til spesifikasjoner. Ingen elementer har duplikate attributter og alle ID'er er unike. Alle applikasjonens komponenter har navn og roller som kan fastslåes programmatisk. Dersom bruker endrer en tilstand, egenskap eller verdi til et element, vil dette også være synlig i koden gjennom Vue.js sin *data-binding*. Alle nivåer er dermed dekket på dette punktet.

6.2 Fargevalg

En av rammebetingelsene til prosjektet var at Oslo kommunes designmanual skulle brukes for farger, ettersom applikasjonen skulle tilpasses Oslo REG. I applikasjonen er alle fargene som er benyttet lagt inn i en egen fil ved navn «_colors.scss» (standard navnekonvensjon for rene SCSS filer).

Applikasjonen benytter seg av flere farger, men har en primærfarge, en sekundærfarge og en kontrastfarge. Fargene er følgende:

- Primærfargen er «Oslo mørk blå».
- Sekundærfargen er «Oslo mørk grønn».
- Kontrastfarge er «Oslo gul»

Foruten om disse er «Oslo rød», «Oslo mørk beige», «Oslo lys beige» og «Oslo lys grønn» brukt på enkelte knapper og bannere.

Figur 50 - Farger fra Oslo kommunes designmanual.
(https://designmanual.oslo.kommune.no/farger).

Det ble gjort ett unntak fra rammebetingelsen. Når en deloppgave i kurs-modulen (se kap. 4.3.6) blir gjennomført, er en grønnfarge med fargekode «#5cb85c» brukt i sjekk-ikonet (se figur 51). Grønnfargen er benyttet i stedet for “Oslo grønn”, som ble vanskelig/ubehagelig å se på mot den lyse bakgrunnen. Valgt farge er roligere og resulterer i bedre synlighet. Vi testet også alternativet «Oslo mørk grønn», men det resulterte i at ikonet ikke så grønt ut.

Figur 51 - Grønnfargen brukt i sjekk-ikonet

6.3 Ikonér

Ikoner er en viktig del av designet og utformelsen av applikasjonen. Det er derfor viktig at ikonene som benyttes er selvforklarende, eller at bruker raskt lærer hva de gjør. Som en støttefunksjon er det implementert hjelpetekster. Teksten blir synlig hvis bruker holder musepeker over ikonet. Teksten sier noe om handlingen som vil gjennomføres ved trykk på ikonet. Videre følger en oversikt over alle ikoner som er benyttet i applikasjonen, samt en kort forklaring.

Oslo kommunes logo

Logoen benyttes i applikasjonen som en hjem-knapp.

Hjelp ikon

Hjelp ikonet består av både tekst og bilde og skal indikere at ved trykk kan en finne hjelp.

Hamburger ikon

Hamburger ikonet er benyttet for å indikere at en meny vil være tilgjengelig ved trykk.

Skjema ikon

Ikonet benyttes i applikasjonen til å indikere at trykk på knapp vil ta bruker til skjema-siden

Tabell ikon

Ikonet benyttes i applikasjonen til å indikere at trykk på knapp vil ta bruker til tabell-siden

Pil opp/ned ikon

Pil ikonene er brukt på tabell-siden i applikasjonen for å indikere retning data er sortert.

Last ned ikon

Ikonet benyttes i applikasjonen i sammenheng med nedlasting av PDF.

Opplasting ikon

Ikonet indikerer opplasting og brukes i applikasjonen i sammenheng med opplasting av bilde og video.

Kryss ikon

Kryss ikonet benyttes i applikasjonen for å lukke CSH.

Hjem ikon

Hjem ikonet benyttes på flere av sidene i CSH i applikasjonen. Ved trykk på ikonet vil bruker bli returnert til forsiden i CSH.

Pil tilbake ikon

Ikonet benyttes på «rapporter feil»-siden i applikasjonen og indikerer at trykk tar bruker tilbake til siden hen kom fra.

Lupe ikon

Ikonet indikerer at søk kan gjennomføres. I applikasjonen benyttes det som en knapp for å navigere til søkesiden i CSH.

Lyspære ikon

Ikonet indikerer hjelp. I applikasjonen benyttes det som en knapp

for å navigere til FAQ-siden i CSH.

Bok ikon

Ikonet indikerer opplæring. I applikasjonen benyttes det som en knapp for å navigere til Kurs-siden i CSH.

Endre/rediger ikon

Ikonet indikerer at en redigering kan gjøres. I applikasjonen benyttes det som en knapp for å navigere til CMS.

Logg ut ikon

Ikonet indikerer logg ut og er i applikasjonen benyttet på knapp for utlogging fra CMS.

Pluss ikon

Ikonet indikerer at noe mer kan vises. I applikasjonen er ikonet benyttet for felter som kan ekspanderes for å vise mer innhold.

Minus ikon

Ikonet indikerer at noe kan gjøres mindre. I applikasjonen er ikonet benyttet for felter som er ekspandert for å indikere at innhold kan skjules.

Sirkel ikon

Ikonet indikerer ikke noe i seg selv, men i konteksten (kurs), så indikerer det en uferdig deloppgave.

Sjekk ikon

Ikonet i sammenheng med kontekst (kurs) indikerer at en deloppgave er gjennomført.

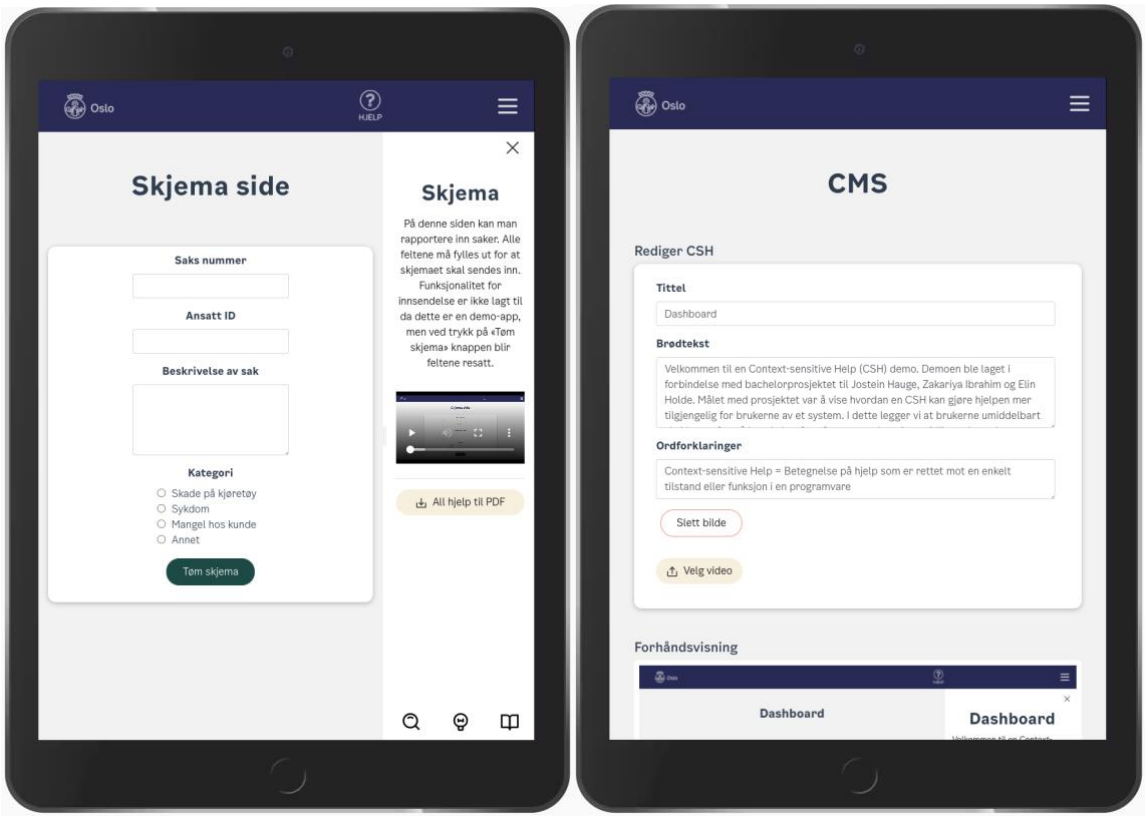
6.4 Font

Ettersom Oslo Kommune sin designmanual skulle følges, er fonten Oslo Sans brukt i hele applikasjonen. Fonten er importert i *App.vue* og satt globalt i samme fil. På den måten vil fonten automatisk bli brukt på all tekst.

Figur 52 - Eksempler på Oslo Sans (https://designmanual.oslo.kommune.no/skrifttype)

6.5 Responsivitet

Responsivitet i webdesign handler om at applikasjonen skal se bra ut uavhengig av hvilken enhet den blir brukt på. Applikasjonen bør kunne justere seg automatisk etter skjermstørrelsen (W3schools, u.å). Ettersom applikasjonens formål er å vise hvordan en CSH kan fungere, noe som kan oppnås på en desktop, var det ikke viktig for oppdragsgiver at applikasjonen fungerte på alle enheter. Design for mobil ble utelukket, ettersom det ville tatt mye tid å designe en god løsning for denne enheten, med tanke på CSH'ens format. Vi ønsket likevel å bruke tid på å gjøre applikasjonen responsiv mellom desktop og nettbrett for å lære mer om hvordan man kan designe responsivt.



Figur 53 - Eksempler i iPad-format (768x1024 px)

Bootstrap sin *container*-klasse er brukt på de fleste sidene i applikasjonen. Klassen gir elementer en responsiv fast bredde som forandrer seg på brytepunktene mellom ulike skjermoppløsninger (Bootstrap, u.å). Der bootstrap sin *container*-klasse ikke kunne brukes er det i stedet brukt egendefinerte klasser med samme formål.

Bootstrap er også brukt på alle input felter og på progresjonslinjen til kursmodulen i CSH'en for å gi dem en dynamisk bredde. På input feltene er *form-control* klassen brukt, og på progressjonslinjen er *progress* og *progress-bar* klassene brukt. Fordelen ved bruk av Bootstrap på denne typen elementer inkluderer for øvrig også at de er universelt utformet.

Ved hjelp av *media queries* kan man gi elementer ulikt design avhengig av oppløsningen på skjermen (Mozilla, u.å). Ettersom Bootstrap sine brytepunkter er implementert gjennom *container*-klassen, er de samme brytepunktene brukt i *queriene*. *Media queries* er både brukt for å endre størrelsen på elementer (se figur

54), og for å vise forskjellige elementer (se figur 55). Her er også SCSS variabler brukt på skjermoppløsningene for å støtte gjennbrukbarhet.

```
@media screen and (max-width: $small-screen) {
  width: 100%;
}
@media screen and (max-width: $medium-screen) {
  width: 80%;
}
```

Figur 54 - *Media query* eksempel 1

```
@media screen and (min-width: $large-screen) {
  .desktop-buttons {
    display: flex;
  }
  .pad-buttons {
    display: none;
  }
```

Figur 55 - *Media query* eksempel 2

Foruten bruk av Bootstrap og *media queries* er relative måleenheter brukt gjennomgående i applikasjonen. Relative måleenheter gir, i motsetning til absolutte måleenheter, elementene en fleksibel størrelse som justerer seg etter størrelsen på foreldre-elementet (Full scale, 2020). Eksempler på relative måleenheter inkluder prosent (%), *em*, *rem* og *viewport* enhetene *vh* og *vw*. I vår applikasjon er hovedsakelig prosent og *rem* brukt.

7. Testdokumentasjon

7.1. Innledning

Grunnen til å gjennomføre testing, er for å verifisere at eksisterende funksjonalitet i et system fungerer. Test dokumentasjonen omfatter enhetstest, systemtest, brukertest og tilgjengelighets test av CSH demo applikasjon. Under enhetstesten ble applikasjonens enheter testet hver for seg. Testene ble utført for å få bekreftet at enhetene fungerte individuelt og som tenkt (STF, 2020). Under gjennomføringen av systemtesten var det derimot applikasjonen i sin helhet som ble testes. Dette for å få bekreftet at klientsiden og serversiden fungerte sammen, og at applikasjonen samsvarte med kravspesifikasjonen (STF, 2020). Under brukertesten ble også hele applikasjonen testet, men her var det mennesker og ikke en maskin som gjennomførte testen. Formålet med brukertesten var å finne ut om applikasjonen var forståelig og enkel å bruke (STF, 2020). Testing av tilgjengelighet ble også utført for å undersøke om applikasjonen er tilrettelagt for brukere med spesielle behov.

Når det gjelder integrasjonstest, gjorde vi en del undersøkelser under forprosjektet. Det ble søkt etter testverktøy som kunne kombineres med Vue 3. Resultatet av undersøkelsene viste at det var svært begrensede alternativer til integrasjonstest. Test-rammeverket Cypress, som vi hadde bestemt oss for å bruke til systemtesting, hadde foreløpig kun støtte for Vue 2.x når det kom til integrasjonstesting. Under et møte med vår interne veileder ble denne problemstillingen tatt opp, og det ble konkludert med at en mer detaljert systemtest ville kunne dekke formålet til integrasjonstesten. Integrasjonstesting ble derfor prioritert bort.

7.2. Testobjekter

7.2.1 Enhetstest

Under enhetstesten ble alle komponentene i «components» mappen testet hver for seg. Her ble bruker input testet og generelt at komponentene i applikasjonen vises. Eksempler på bruker input inkluderer input felter og trykk på knapper. Vuex mutasjoner ble testet i isolasjon under denne testfasen, fordi det er mutasjonen som endrer

tilstands-variabelen (State). Det ble også testet at router (navigasjon) kan kalles, men ikke gjennomføres.

Komponentene som ble testet:

- DashboardComponent.vue
- TableComponent.vue
- FormComponent.vue
- PageNotFound.vue
- Dropdown.vue
- TheHeader.vue
- CshFrontpage.vue
- CshSearch.vue
- CshFaq.vue
- CshSupport.vue
- CshCourseModule.vue
- CshTopNav.vue
- CshBotNav.vue
- CmsLogin.vue
- CmsLoggedIn.vue
- CmsComponent.vue
- CmsButtonBar.vue
- Preview.vue
- BaseButton.vue
- BaseCard.vue
- BaseTitle.vue
- PictureCard.vue
- VideoCard.vue
- Spinner.vue
- CshTitle.vue

Vuex som ble testet:

- Vuex auth
- Vuex visibility
- Vuex pages

- Vuex RouteName
- Vuex course

7.2.2 Systemtest

I systemtesten ble systemet testes i sin helhet.

7.2.3 Brukertest

I brukertesten ble systemet testes i sin helhet.

7.2.4 Tilgjengelighet

I tilgjengelighet testen ble klientsiden av systemet testes.

7.3. Testverktøy

7.3.1 Enhetstest:

Under enhetstest brukte vi Jest som vårt testmiljø, da Jest kom med i konfigurasjonen av prosjektet vårt som en standard. Jest sitt grensesnitt var oversiktlig, noe som gjorde det enklere å se om en test har feilet eller passert. Det var også enkelt å genere dekningsrapporter (code coverage) som en del av testresultatet. Rapporten viste blant annet antall prosent av kodelinjer som var dekket av testen. I tillegg brukte vi Vue Test Utils for å prøve å få tilgang til Vue spesifikk kode.

7.3.2 Systemtest:

For systemtest ble det bestemt å gå for Cypress som vårt testmiljø. Det var flere grunner til dette valget. Et av de viktigste argumentene for valget var at et av gruppemedlemmene hadde noe erfaring med Cypress, i motsetning til andre *e2e* verktøy. Cypress er begrenset til å gjennomføre testene i Chromium (Cypress u.å). Begrensningen var ikke et problem for oss, fordi vi ikke hadde noen spesifikasjon i form av nettleser. Oppdragsgiver hadde i tillegg uttalt seg om at de benyttet Chrome, og ønsket at andre også skulle bruke nettleseren.

7.3.3 Tilgjengelighet

7.3.3.1 Acart Contrast Checker

Acart Communications sin «contrast checker» ble brukt til å regne ut kontrast-raten mellom tekstfarge og bakgrunnsfarge i applikasjonen.

7.3.3.2 Lighthouse

Lighthouse er en del av Google Chrome sine DevTools, et verktøy for utviklere. Verktøyet kan kjøres direkte i nettleseren, og avleverer en rapport etter å ha scannet applikasjonen.

7.3.3.3 Chrome Screen Reader

Google Chrome har en skjermleser tilgjengelig som en utvidelse av nettleseren. Skjermleseren ble brukt for å simulere hvordan det ville vært å bruke applikasjonen for en bruker med en funksjonsnedsettelse.

7.4. Fremgangsmåte

7.4.1 Overordnede teststrategier

7.4.1.1 Enhetstest

Test rammeverkene; Vue Test Utils og Jest ble benyttet til å gjennomføre enhetstesting. Testene ble skrevet underveis i utviklingen, og kjørt hver gang ny kode ble integrert i applikasjonen. For å gjøre det enklere å gjennomføre dette i praksis, ble det brukt GitHub Actions til å kjøre testene automatisk hver gang kode ble *pushet* til master. Det gjorde at eventuelle feil ble oppdaget tidlig.

7.4.1.2 Systemtest

I systemtesten ble hele programmet testet av det automatiserte testverktøyet Cypress. Testene simulerte en brukers interaksjon med systemet, noe som inkluderte kall til *backend*. Under testen hadde ikke testverktøyet kjennskap til applikasjonens interne strukturer eller kode. Denne metoden er også kjent som *black-box* testing (STF, 2020). Testene fulgte en testplan bestående av brukerhistorier som var basert på kravspesifikasjonen.

Ettersom vi fulgte prinsippene til kontinuerlig integrering (CI), ville det vært naturlig at systemtesten også ble kjørt gjennom GitHub Actions. Det å sette opp en systemtest i GitHub Actions var utfordrende og det var en risiko for at det ville koste penger. Det skyldes at gratisversjonen hadde en begrensning (GitHub, 2021). Samtidig var det få fordeler knyttet til en implementering i GitHub Actions, da systemtesten i vårt tilfelle ble kjørt helt mot slutten av utviklingen. Det ble av den grunn ikke prioritert i vårt prosjekt.

7.4.1.3 Brukertest

I brukertesten ble applikasjonen som en helhet testet av utenforstående. En utenforstående er en som ikke er med i gruppen, og ikke er oppdragsgiver. Som en konsekvens av Covid-19 og nedstengning i Oslo, utgikk alfa-testingen som normalt finner sted i begynnelsen av prosjektet. Selv om alfa-testing utgikk, ble det gjennomført beta-testing. Testingen fant sted mot slutten av utviklingen. På bakgrunn av virussituasjonen besto testpersonene av nærkontaktene til gruppemedlemmene. Testmiljøet var i nærkontaktene sitt eget hjem.

7.4.1.4 Tilgjengelighet

Testing av tilgjengelighet ble gjort kontinuerlig gjennom utviklingen. Vi satt oss tidlig inn i kravene for tilgjengelighet på nett, slik at vi kunne oppdage mulige utfordringer så raskt som mulig. Det var for eksempel naturlig å teste kontrast-raten når vi brukte bakgrunnsfarge bak tekst. Applikasjonen i sin helhet ble testet for tilgjengelighet på slutten av utviklingsprosessen.

7.4.2 Prosedyre for planlegging og gjennomføring av test

7.4.2.1 Enhetstest

For gjennomføring av enhetstesting, måtte test rammeverkene; Vue Test Utils og Jest installeres i prosjektet. Testene ble kjørt i rekkefølge basert på hvilke mapper de lå i. Mappene i test-mappen reflekterte mappestrukturen i det faktiske programmet. For at enhetstestene skulle kunne kjøres var det nok at testmiljøet var satt opp, og at eventuelle test-variabler var lagt til i koden der det var ønskelig. Testene ble spesifisert med en beskrivende blokk som kort fortalte hvilken komponent som ble testet. Inne i

den beskrivende blokken ble det lagt opp til flere test-blokker som inneholdt de ulike testene.

7.4.2.2 Systemtest

For gjennomføring av systemtesting, måtte test rammeverket Cypress installeres i prosjektet. Testene ble organisert i to test suiter, en for admin og en for bruker. Selve utførelsen fulgte brukerhistoriene som var satt opp i testplanen, og brukte input som var definert i tabelloversikt som er visualisert i figur 56 og 57 nedenfor.

Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
AC-0001	Som en bruker ønsker jeg å trykke på hjelp- knapp, for å finne informasjon om gjeldene side i systemet	csh åpnes og viser informasjon			Not Started			
AC-0002	Som en bruker ønsker jeg å ha tilgang til et kurs	kursmodul er tilgjengelig			Not Started			
AC-0003	Som bruker ønsker jeg å gjennomføre et kurs, for å få opplæring i systemet	kursProsent =100%			Not Started			
AC-0004	Som en bruker ønsker jeg å kunne søke i dokumentasjonen, dette for å spare tid	resultater vises når utrykk er søkt på			Not Started			
AC-0005	Som en bruker ønsker jeg en å kunne få svar på spørsmål jeg har gjennom en FAQ	faq finnes og viser tekst			Not Started			
AC-0006	Som bruker ønsker jeg og laste ned all dokumentasjon som PDF	pdf blir lastet ned			Not Started			
AC-0007	Som bruker ønsker jeg og kunne rapportere feil i systemet	support side vises og felter kan fylles ut	navn: test, epost: test@gmail.com, tekst: test		Not Started			
AC-0008	Som bruker ønsker jeg å enkelt kunne sortere tabell for bedre innsikt i informasjon	tabell kan sorteres			Not Started			
AC-0009	Som bruker ønsker jeg og kunne fylle ut et skjema, og trykke på knapp for tømming av skjema	fyller ut felter og tømmer på trykk	saksnummer: 1, ansattID: 1, beskrivelse: tests, sjekk av sykdom		Not Started			

Figur 56 - Oversikt tabeller til systemtest for bruker

Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
AC-0001	Som admin ønsker jeg å kunne logge inn i CMS	Logget inn	clientId: "b4b3d62f-43eb-4013-a40e-8e9431f558c3", tenantId: "ee07e5f1-6968-4a03-aefa-71cfca69c820", clientSecret: "FISPC_V5y~9_.QkY4v9342~Ds0GpT57pi7"	admin blir logget inn	Not Started			Bruker bypass for å komme forbi azure ad login, dette er standard for cypress
AC-0002	Som admin ønsker jeg å bruke CMS, for å kunne gjøre endringer på innhold i CSH	viser cms		viser cms	Not Started			
AC-0003	som admin ønsker jeg å legge til innhold der det ikke allerede finnes innhold	nytt innhold blir lagt inn	tittel: test, brødtekst: test, ordforklaring: test, bilde: myFixture1.png, video: myFixture2.mp4		Not Started			Brukt Cypress-file-upload for upload av bilde og video. Laget en egen mappe kalt fixtures som kalles på for å innhente bilde og video
AC-0003	som admin ønsker jeg å avbryte en endring i CMS	cms operasjon blir avbrutt		handling ble avbrutt	Not Started			Går inn på CMS siden, trykker så på avbryt. Sjekk deretter mot at endringer ikke er utført
AC-0004	som admin ønsker jeg å bruke CMS, for å slette innhold fra CSH	innhold blir slettet		innhold blir slettet	Not Started			Går inn på CMS siden, trykker så på slett. Sjekk deretter mot at endringer ikke er utført
AC-0005	som admin ønsker jeg å oppdatere allerede eksisterende innhold	innhold blir oppdatert	tittel: Tabell, brødtekst: dette er en test, ordforklaring: test1, bilde:slett bilde, video: la stå		Not Started			Oppdatering blir gjort ved å først lage content forså å slette tilslutt
AC-0006	som admin ønsker jeg å laste opp et bilde i CMS	bilde blir lastet opp	bilde: myFixture1.png		Not Started			
AC-0007	som admin ønsker jeg å laste opp en video i CMS	video blir lastet opp	video: myFixture2.mp4		Not Started			
AC-0008	som admin ønsker jeg å slette bilde i CMS	bilde blir slettet			Not Started			
AC-0009	som admin ønsker jeg å slette video i CMS	video blir slettet			Not Started			
AC-0010	som admin ønsker jeg å logge ut når jeg er ferdig å jobbe	Logget ut			Not Started			

Figur 57 - Oversikt tabeller til systemtest for admin

7.4.2.3 Brukertest

For at alle testpersoner skulle gjennomføre en tilsvarende lik test, ble testen gjort i en bestemt rekkefølge og med instruksjoner underveis. Det ble samtidig skrevet notater og til slutt ble testpersonene spurt om det helhetlige inntrykket, og om de hadde forslag til ting som kunne utbedres. Normalt vil disse oppgavene blitt fordelt på flere personer under en brukertest, men ettersom Covid-19 gjorde det utfordrende å møtes, var det en person som fylte alle rollene. Det vil si at det var samme personen som ga instruksjoner, observerte og tok notater.

Instruksjoner

Dummy-app:

1. Gå til skjema siden, fyll ut skjemaet, trykk på knappen og gå tilbake til hovedsiden.
2. Gå til tabell siden, sorter tabellen på de ulike kolonnene og gå tilbake til hovedsiden.
3. Finn hjelpefunksjonen.

CSH:

1. Mens du er på forsiden, naviger deg rundt i dummy-appen.
2. Last ned PDF.
3. Gjør et søk.
4. Finn ofte stilte spørsmål.
5. Finn support og send en feilrapport.
6. Gjennomfør et kurs.

CMS:

Du er nå admin og har mulighet til å logge inn ved å skrive “/cms” bakerst på url.

1. Trykk på logg inn knappen og velg riktig bruker.
2. Finn en side du vil gjøre endringer på og gå til CMS’en.
3. Endre på informasjon og trykk avbryt.
4. Gå tilbake til CMS og slett innholdet.
5. Gå tilbake til CMS, legg inn nytt innhold og trykk lagre.
6. Logg ut av admin-modus.

7.4.2.4 Tilgjengelighet

For å gjennomføre test av tilgjengelighet for skjermlesere måtte utvidelsen til Chrome (se 7.3.3.3) legges til i nettleseren. Et av gruppemedlemmene gikk så gjennom hele applikasjonen med lukkede øyne og navigerte seg rundt kun ved hjelp av skjermleseren.

På slutten av utviklingsprosessen ble de relevante punktene i «Web Content Accessibility Guidelines» (WCAG) testet (se figur 58 for oversikt over punktene). Lighthouse ble kjørt gjennom applikasjonen, og eventuelle feil som dukket opp ble rettet.

Punkt	Nivå	Status
1.1.1	A	
1.2.1	A	Irrelevant
1.2.2	A	Irrelevant
1.2.3	A	Irrelevant
1.2.4	AA	Irrelevant
1.2.5	AA	Irrelevant
1.2.6	AAA	Irrelevant
1.2.7	AAA	Irrelevant
1.2.8	AAA	Irrelevant
1.2.9	AAA	Irrelevant
1.3.1	A	
1.3.2	A	
1.3.3	A	
1.3.4	AA	
1.3.5	AA	
1.3.6	AAA	
1.4.1	A	
1.4.2	A	Irrelevant
1.4.3	AA	
1.4.4	AA	
1.4.5	AA	
1.4.6	AAA	
1.4.7	AAA	
1.4.8	AAA	
1.4.9	AAA	
1.4.10	AA	
1.4.11	AA	
1.4.12	AA	
1.4.13	AA	
2.1.1	A	
2.1.2	A	
2.1.3	AAA	
2.1.4	A	Irrelevant
2.2.1	A	
2.2.2	A	Irrelevant
2.2.3	AAA	
2.2.4	AAA	
2.2.5	AAA	
2.2.6	AAA	
2.3.1	A	Irrelevant
2.3.2	AAA	Irrelevant
2.3.3	AAA	Irrelevant
2.4.1	A	
2.4.2	A	
2.4.3	A	
2.4.4	A	
2.4.5	AA	
2.4.6	AA	
2.4.7	AA	
2.4.8	AAA	
2.4.9	AAA	
2.4.10	AAA	
2.5.1	A	Irrelevant
2.5.2	A	
2.5.3	A	
2.5.4	A	Irrelevant
2.5.5	AAA	Irrelevant
2.5.6	AAA	
3.1.1	A	
3.1.2	AA	Irrelevant
3.1.3	AAA	
3.1.4	AAA	
3.1.5	AAA	
3.1.6	AAA	
3.2.1	A	
3.2.2	A	
3.2.3	AA	
3.2.4	AA	
3.2.5	AAA	
3.3.1	A	Irrelevant
3.3.2	A	
3.3.3	AA	Irrelevant
3.3.4	AA	Irrelevant
3.3.5	AAA	
3.3.6	AAA	Irrelevant
4.1.1	A	
4.1.2	A	
4.1.3	AA	

Figur 58 - Oversikt over punktene i WCAG

7.4.3 Kriterier for grundighet i testen

7.4.3.1 Enhetstest

Overordnet var ikke dekningsgraden det viktigste for denne oppgaven, men heller det å skape et innblikk i at vi kunne skrive slike tester. Grunnet manglende støtte for versjon 3 av Vue i tilgjengelig test-rammeverk, visste vi på forhånd at deler av komponentene i applikasjonen ikke ville være mulig å teste i enhetstesten. Basert på disse opplysningene var kriteriet for grundighet satt til at enhetstesten burde dekke minst 50% av komponentene.

7.4.3.2 Systemtest

Kriteriet for grundighet i systemtesten var satt til at alle definerte brukerhistorier skulle testes.

7.4.3.3 Brukertest

Kriteriet for grundighet i brukertesten var satt til at alle gruppens medlemmer skulle gjennomføre minst to brukertester hver. Den ene testpersonen skulle være i aldersgruppen 15-30 år og den andre 40-70 år. Testpersonene skulle teste applikasjonen i sin helhet, og det ble sørget for gjennom å følge alle instruksjonene i testplanen (se kap. 7.4.2.3).

7.4.3.4 Tilgjengelighet

I Norge har vi en forskrift som sier at:

«Nettløsninger skal minst utformes i samsvar med standard Web Content Accessibility Guidelines 2.0 (WCAG 2.0)/NS/ISO/IEC 40500:2012, på nivå A og AA med unntak for suksesskriteriene 1.2.3, 1.2.4 og 1.2.5, eller tilsvarende denne standard.» (Forskrift om universell utforming av IKT-løsninger, 2013, § 1-11).

Forskriften trådte i kraft 1. Juli 2013, men det var først 1. Januar 2021 at alle nye og eksisterende nettløsninger ble pliktig til å følge denne forskriften. Videre er det ifølge tilsynet for universell utforming av IKT stor grunn til å tro at også WCAG 2.1 vil bli en del av det norske regelverket (Uutilsynet, u.å).

Selv om bachelorprosjektet ikke skulle publiseres ble det bestemt at applikasjonen skulle oppfylle de gjeldende kravene til den norske forskriften som et minimumsmål. Det ble samtidig jobbet mot å oppfylle så mange krav som mulig utover disse. Når hovedmålsettingen med prosjektet var å vise hvordan en CSH kan gjøre hjelpen mer tilgjengelig, var det å følge WCAG sine retningslinjer fundamentalt. Det ville for eksempel vært krevende å nå målsettingen dersom en tastatur-bruker ikke kunne tabbet seg inn i CSH'en.

7.5. Resultat

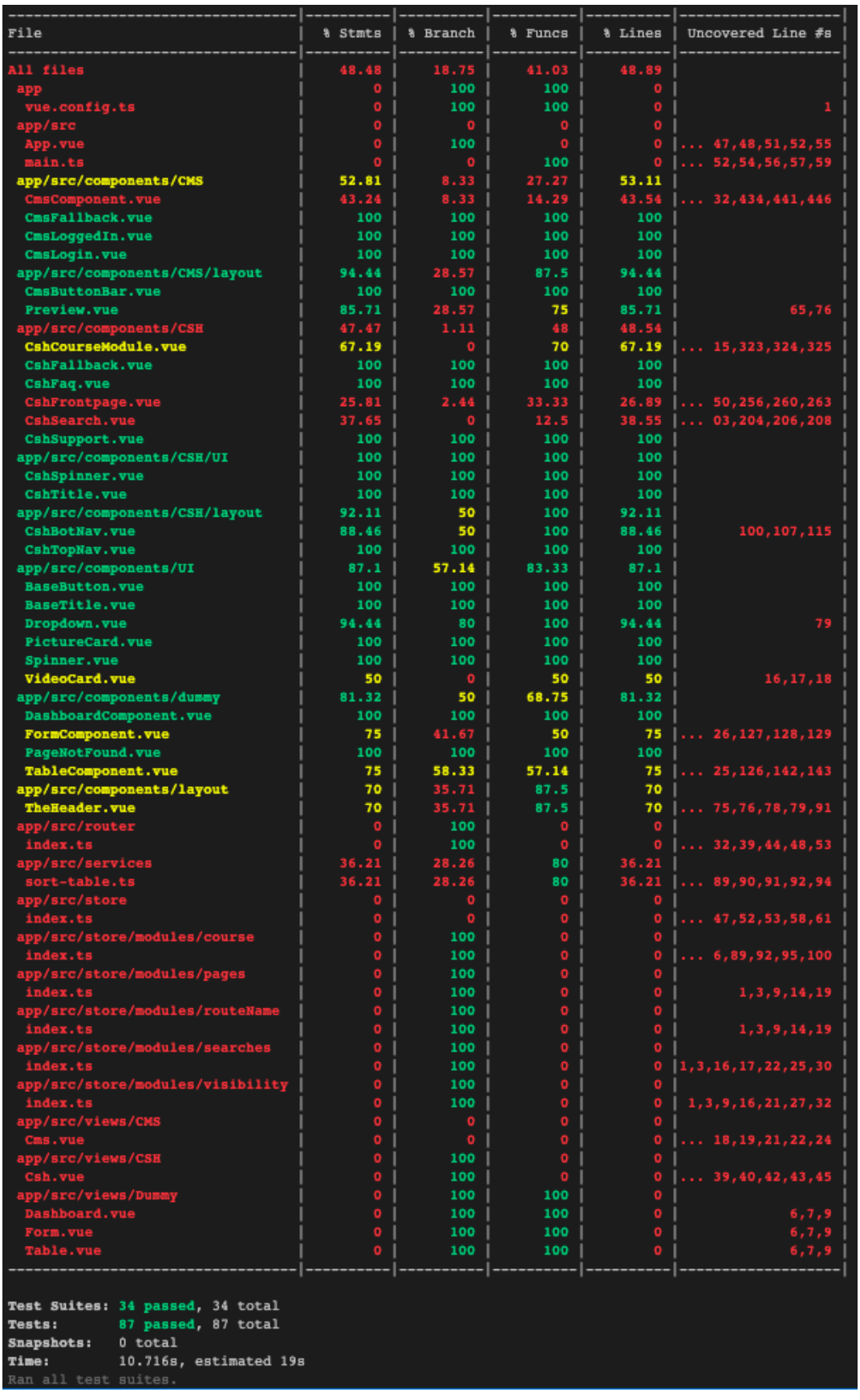
Innledningsvis i testdokumentasjonen var vi inne på målet med testing, som var å verifisere at funksjonaliteten i applikasjonen fungerte. Testen kunne være med på å belyse eksisterende feil, men den kunne ikke utelukke at feil fantes. Som et resultat av testingen hadde vi verifisert at funksjonaliteten som var testet fungerte.

Videre presenteres resultatene for henholdsvis enhetstest, systemtest, brukertest og tilgjengelighet test. For enhetstest og systemtest, foreligger det et par eksempler på hvordan testene ble utført og hvordan resultatet ble presentert.

7.5.1 Enhetstest

7.5.1.1 Oversikt

Kriteriet for grundighet i enhetstesten (se 7.4.3.1) var en linjedekning på minst 50%. Resultatet av testen viste oss en linjedekning på 48,89% (se figur 59). Siden Vuex testene ikke er knyttet til en spesifikk komponent, vil ikke resultatet av disse testene inkluderes i linjedekningen. Prosenten vil derfor ikke reflektere den reelle dekningsgraden. Normalt ville det vært å forvente en høyere prosent linjedekning på en enhetstest. I vårt tilfelle var ikke det mulig ettersom deler av komponentene var skjult bak implementasjon, som ikke var støttet av testverktøyet. Enhetstestene ble kjørt underveis i utviklingen og feilene som ble oppdaget ble derfor rettet fortløpende. Da utviklingen var ferdig, kjørte alle testene uten feil, og oversikt over feil foreligger derfor ikke.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	48.48	18.75	41.03	48.89	
app	0	100	100	0	
vue.config.ts	0	100	100	0	1
app/src	0	0	0	0	
App.vue	0	100	0	0	... 47,48,51,52,55
main.ts	0	0	100	0	... 52,54,56,57,59
app/src/components/CMS	52.81	8.33	27.27	53.11	
CmsComponent.vue	43.24	8.33	14.29	43.54	... 32,434,441,446
CmsFallback.vue	100	100	100	100	
CmsLoggedIn.vue	100	100	100	100	
CmsLogin.vue	100	100	100	100	
app/src/components/CMS/layout	94.44	28.57	87.5	94.44	
CmsButtonBar.vue	100	100	100	100	
Preview.vue	85.71	28.57	75	85.71	65,76
app/src/components/CSH	47.47	1.11	48	48.54	
CshCourseModule.vue	67.19	0	70	67.19	... 15,323,324,325
CshFallback.vue	100	100	100	100	
CshFaq.vue	100	100	100	100	
CshFrontpage.vue	25.81	2.44	33.33	26.89	... 50,256,260,263
CshSearch.vue	37.65	0	12.5	38.55	... 03,204,206,208
CshSupport.vue	100	100	100	100	
app/src/components/CSH/UI	100	100	100	100	
CshSpinner.vue	100	100	100	100	
CshTitle.vue	100	100	100	100	
app/src/components/CSH/layout	92.11	50	100	92.11	
CshBotNav.vue	88.46	50	100	88.46	100,107,115
CshTopNav.vue	100	100	100	100	
app/src/components/UI	87.1	57.14	83.33	87.1	
BaseButton.vue	100	100	100	100	
BaseTitle.vue	100	100	100	100	
Dropdown.vue	94.44	80	100	94.44	79
PictureCard.vue	100	100	100	100	
Spinner.vue	100	100	100	100	
VideoCard.vue	50	0	50	50	16,17,18
app/src/components/dummy	81.32	50	68.75	81.32	
DashboardComponent.vue	100	100	100	100	
FormComponent.vue	75	41.67	50	75	... 26,127,128,129
PageNotFound.vue	100	100	100	100	
TableComponent.vue	75	58.33	57.14	75	... 25,126,142,143
app/src/components/layout	70	35.71	87.5	70	
TheHeader.vue	70	35.71	87.5	70	... 75,76,78,79,91
app/src/router	0	100	0	0	
index.ts	0	100	0	0	... 32,39,44,48,53
app/src/services	36.21	28.26	80	36.21	
sort-table.ts	36.21	28.26	80	36.21	... 89,90,91,92,94
app/src/store	0	0	0	0	
index.ts	0	0	0	0	... 47,52,53,58,61
app/src/store/modules/course	0	100	0	0	
index.ts	0	100	0	0	... 6,89,92,95,100
app/src/store/modules/pages	0	100	0	0	
index.ts	0	100	0	0	1,3,9,14,19
app/src/store/modules/routeName	0	100	0	0	
index.ts	0	100	0	0	1,3,9,14,19
app/src/store/modules/searches	0	100	0	0	
index.ts	0	100	0	0	1,3,16,17,22,25,30
app/src/store/modules/visibility	0	100	0	0	
index.ts	0	100	0	0	1,3,9,16,21,27,32
app/src/views/CMS	0	0	0	0	
Cms.vue	0	0	0	0	... 18,19,21,22,24
app/src/views/CSH	0	100	0	0	
Csh.vue	0	100	0	0	... 39,40,42,43,45
app/src/views/Dummy	0	100	100	0	
Dashboard.vue	0	100	100	0	6,7,9
Form.vue	0	100	100	0	6,7,9
Table.vue	0	100	100	0	6,7,9

Test Suites: 34 passed, 34 total
Tests: 87 passed, 87 total
Snapshots: 0 total
Time: 10.716s, estimated 19s
Ran all test suites.

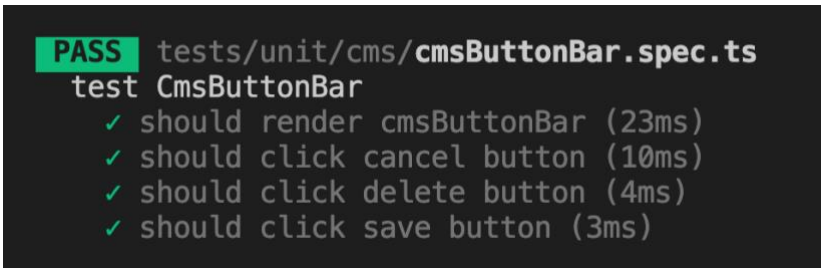
Figur 59 - Oversikt over linjedekning

7.5.1.2 Komponent eksempel

I utvalgt eksempel kan en se test av komponent CmsButtonBar (se figur 60). Den er trukket frem da den inneholder flere forskjellige tester som generelt ble brukt gjennom enhetstestingen. I CmsButtonBar komponenten var det ønskelig å teste om knappene eksisterte, og deretter sjekke om hver av knappene kunne trykkes på. Når knappen ble trykket, ble det i tillegg sjekket om den gjør kall basert på riktig verdi. Alle testene kjørte som ønsket (se figur 61).

```
import { config, mount } from "@vue/test-utils";
import CmsButtonBar from "@/components/CMS/layout/CmsButtonBar.vue";
import {
  VueRouterMock,
  createRouterMock,
  injectRouterMock
} from "vue-router-mock";

config.plugins.VueWrapper.install(VueRouterMock);

describe("test CmsButtonBar", () => {
  const router = createRouterMock();

  beforeEach(() => {
    injectRouterMock(router);
  });

  test("should render cmsButtonBar", () => {
    const wrapper = mount(CmsButtonBar);
    const cancelButton = wrapper.find('[data-test="cancelButton"]');
    const deleteButton = wrapper.find('[data-test="deleteButton"]');
    const saveButton = wrapper.find('[data-test="saveButton"]');

    expect(cancelButton.exists()).toBe(true);
    expect(deleteButton.exists()).toBe(true);
    expect(saveButton.exists()).toBe(true);
  });

  test("should click cancel button", async () => {
    const wrapper = mount(CmsButtonBar);
    const button = wrapper.find('[data-test="cancelButton"]');
    expect(button.text()).toMatch("Avbryt");
    await button.trigger("click");
    expect(wrapper.emitted()).toHaveProperty("cancel");
  });

  test("should click delete button", async () => {
    const wrapper = mount(CmsButtonBar);
    const button = wrapper.find('[data-test="deleteButton"]');
    expect(button.text()).toMatch("Slett");
    await button.trigger("click");
    expect(wrapper.emitted()).toHaveProperty("delete-data");
  });

  test("should click save button", async () => {
    const wrapper = mount(CmsButtonBar);
    const button = wrapper.find('[data-test="saveButton"]');
    expect(button.text()).toMatch("Lagre");
    await button.trigger("click");
    expect(wrapper.emitted()).toHaveProperty("save-object");
  });
});
```

Figur 60 - CmsButtonBar.spec.ts



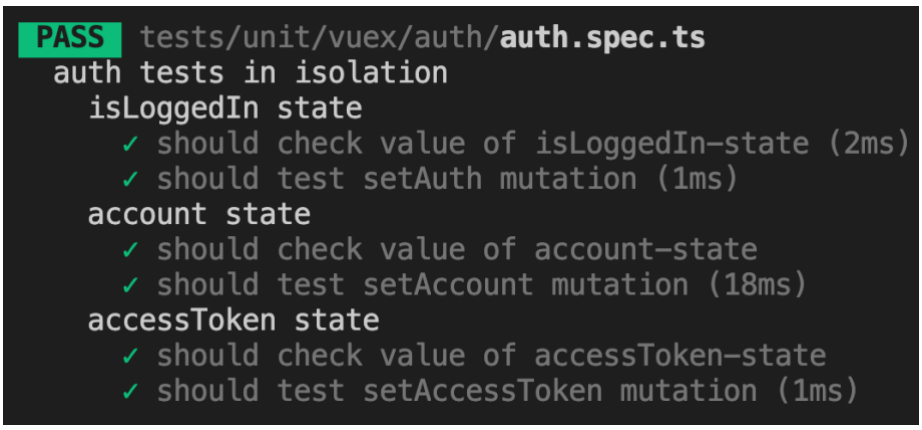
Figur 61 - Resultat cmsButtonBar.spec.ts

7.5.1.3 Vuex eksempel

I utvalgt eksempel kan en se test av autentiserings tilstandsvariabler i Vuex (se figur 62). Testen er trukket frem for å være et eksempel på en test av tilstandsvariabel i isolasjon. Det ble først testet om standard tilstandsverdi var riktig, deretter ble det testet om verdiene kunne endres ved mutasjoner. Alle testene kjørte som ønsket (se figur 63).

```
import { createStore } from "vuex";  12.3K (gzipped: 3.8K)
const store = createStore({
  state: {
    accessToken: "",
    account: "",
    isLoggedIn: false
  },
  mutations: {
    setAccessToken(state, token) {
      state.accessToken = token;
    },
    setAuth(state, payload) {
      state.isLoggedIn = payload.isAuth;
    },
    setAccount(state, payload) {
      state.account = payload.myAccount;
      console.log(state.account);
    }
  }
});
describe("auth tests in isolation", () => {
  describe("isLoggedIn state", () => {
    test("should check value of isLoggedIn-state", () => {
      expect(store.state.isLoggedIn).toBe(false);
    });

    test("should test setAuth mutation", () => {
      store.commit("setAuth", { isAuth: true });
      expect(store.state.isLoggedIn).toBe(true);
    });
  });
  describe("account state", () => {
    test("should check value of account-state", () => {
      expect(store.state.account).toBe("");
    });

    test("should test setAccount mutation", () => {
      store.commit("setAccount", { myAccount: "test" });
      expect(store.state.account).toBe("test");
    });
  });
  describe("accessToken state", () => {
    test("should check value of accessToken-state", () => {
      expect(store.state.accessToken).toBe("");
    });

    test("should test setAccessToken mutation", () => {
      store.commit("setAccessToken", "test");
      expect(store.state.accessToken).toBe("test");
    });
  });
});

```

Figur 62 - Auth.spec.ts



Figur 63 - Resultat Auth.spec.ts

7.5.2 Systemtest

7.5.2.1 Oversikt

Kriteriet for grundighet i systemtesten var at alle brukerhistoriene skulle bli testet, både for bruker-siden og admin-siden av applikasjonen. Resultatet av første testkjøring belyste 2 feil i systemet. Den første feilen ble oppdaget under implementasjon av test av forsiden i CSH'en. Dersom det var lagret et tomt objekt i databasen, forsto ikke systemet at innhold ikke var tilgjengelig, og kommuniserte derfor ikke dette til brukeren. Den andre feilen ble oppdaget under kjøring, og det var at «endre side» knappen i navigasjonsbaren ikke ble utilgjengelig på sider der den skulle det (se punkt 7.5.2.2). Feilene ble rettet opp før ny kjøring ble gjennomført og rettelsene ble fastslått.

Figur 64 og 65 viser en oversikt over alle brukerhistoriene for systemtesten. Oversikten innebærer test-caser med tilhørende ID, beskrivelse, forventet resultat, test data, utskrevet resultat, samt om testen er godkjent eller ikke. Det ble også lagt med en kommentarer for å beskrive elementære aspekter ved testen. Oversikten fungerte som en testplan og ble fylt ut underveis i prosessen.

Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
AC-0001	Som en bruker ønsker jeg å trykke på hjelp- knapp, for å finne informasjon om gjeldene side i systemet	csh åpnes og viser informasjon		csh åpnes og informasjon vises	Pass			All innhold (tittel, tekst, ordforklaring, bilde og video) må fylles ut for at at ingen tester skal feile.
AC-0002	Som en bruker ønsker jeg å ha tilgang til et kurs	kursmodul er tilgjengelig		Kursmodulen åpnes, de ulike modulene er tilgjengelige	Pass			Sjekk av diverse klikk, fra om buttons til tekst-innhold eksisterer
AC-0003	Som bruker ønsker jeg å gjennomføre et kurs, for å få opplæring i systemet	kursProsent =100%	Modul1: SaksNr: 1, AnsattId: 1, Beskrivelse: test, trykk sykdom	kurs er gjennomført og prosent er lik 100%	Pass			Går her gjennom kurs-modul i rekkefølge
AC-0004	Som en bruker ønsker jeg å kunne søke i dokumentasjonen, dette for å spare tid	resultater vises når utrykk er søkt på	Søkeord tekst: demo, søkeord tittel: Dashboard, søkeord ordforklaring: hjelp	søk funker som det skal, resultat vises	Pass			Måtte legges inn en pause/ventetid som gjøre at søk rekker å gjennomføres før assertion. Testen er avhengig av at riktig tekst ligger inne
AC-0005	Som en bruker ønsker jeg en å kunne få svar på spørsmål jeg har gjennom en FAQ	faq finnes og viser tekst		Faq finnes og viser både spørsmål og svar	Pass			Navigerer seg til faq, trykker seg deretter gjennom alle spørsmål og svar
AC-0006	Som bruker ønsker jeg og laste ned all dokumentasjon som PDF	pdf blir lastet ned		PDF blir lastet ned	Pass			Url til lastet ned fil er Cypress/downloads/CSH content
AC-0007	Som bruker ønsker jeg og kunne rapportere feil i systemet	support side vises og felter kan fylles ut	navn: test, epost: test@gmail.com, tekst: test	felter tømmes ved trykk på knapp	Pass			Rapportering kan utføres. Tiltenkt tekst legger seg til høyre i de forhåndsfylte feltene.
AC-0008	Som bruker ønsker jeg å enkelt kunne sortere tabell for bedre innsikt i informasjon	tabell kan sorteres	Sorterer på alle felt	Tabell sorteres på bakgrunn av hvilket felt man trykker på	Pass			Gjøres sortering mot alle felter. Kan trykke hvor som helst i de ulike feltene så gjøres det sortering mot click
AC-0007	Som en bruker ønsker å sjekke om Dashboard-siden eksisterer, for å kunne navigere meg dit når jeg vil	Dashboard side åpnes->sjekk mot tittel	Dashboard siden åpnes - Og tittel riktig tittel eksisterer	Dashboard og tittel fremvises	Pass			Tar bruker til front siden. Gjør deretter sjekk om ønsket tittel er på denne siden.
AC-0009	Som bruker ønsker jeg og kunne fylle ut et skjema, og trykke på knapp for tømming av skjema	fyller ut felter og tømmer på trykk	saksnummer: 1, ansattID: 1, beskrivelse: test, sjekk av sykdom	Felter fylles ut og tømmes ved trykk på slett knapp	Pass			Går til app-rooten, trykker seg så inn til skjema siden, fyller ut ønsket inn data forstå å tømme alle felter - Alert-toast med tilbakemelding når tømt

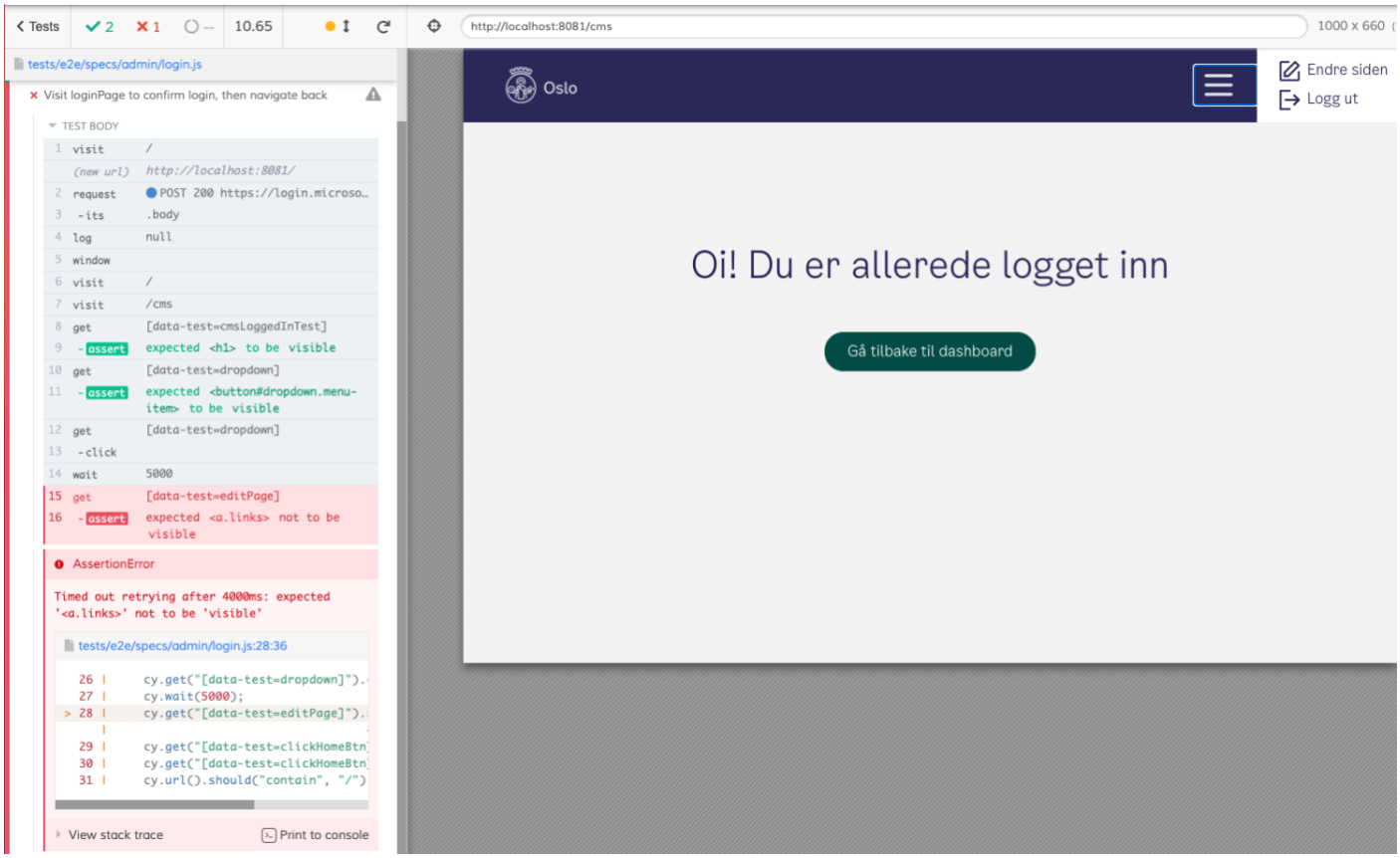
Figur 64 - Oversikt systemtest bruker-siden

Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
AC-0001	Som admin ønsker jeg å kunne logge inn i CMS	Logget inn	clientId: "b4b3d62f-43eb-4013-a40e-8e9431f558c3", tenantId: "ee07e5f1-6968-4a03-aefa-71cfca69c820", clientSecret: "FISPC_V5y~9_.QkY4v9342~Ds0GpT57pi7"	admin blir logget inn	Pass			Bruker bypass-metode for å komme forbi azure AD login, dette er standard for cypress, testdata ligger i cypress.json fil
AC-0002	Som admin ønsker jeg å bruke CMS, for å kunne gjøre endringer på innhold i CSH	viser cms		viser cms	Pass			må være logget inn for å vise cms
AC-0003	som admin ønsker jeg å legge til innhold der det ikke allerede finnes innhold	nytt innhold blir lagt inn	tittel: test, brødtekst: test, ordforklaring: test, bilde: myFixture1.png, video: myFixture2.mp4	nytt innhold blir lagt inn	Pass			Brukt Cypress-file-upload for upload av bilde og video. Laget en egen mappe kalt fixtures som kalles på for å innhente bilde og video
AC-0003	som admin ønsker jeg å avbryte en endring i CMS	cms operasjon blir avbrutt		handling ble avbrutt	Pass			Går inn på CMS siden, trykker så på avbryt. Sjekk deretter mot at endringer ikke er utført
AC-0004	som admin ønsker jeg å bruke CMS, for å slette innhold fra CSH	innhold blir slettet		innhold blir slettet	Pass			Går inn på CMS siden, trykker så på slett. Sjekk deretter mot at endringer ikke er utført
AC-0005	som admin ønsker jeg å oppdatere allerede eksisterende innhold	innhold blir oppdatert	tittel: Tabell, brødtekst: dette er en test, ordforklaring: test1, bilde:slett bilde, video: la stå	innholdet blir oppdatert	Pass			Oppdatering blir gjort ved å først lage content forså å slette tilslutt
AC-0006	som admin ønsker jeg å laste opp et bilde i CMS	bilde blir lastet opp	bilde: myFixture1.png	bilde blir lastet opp	Pass			trykk på knapp åpner en fil-velger, etter valgt fil, kan bruker sjekke preview for bekreftelse, samt ligger filnavn ved siden av knappen
AC-0007	som admin ønsker jeg å laste opp en video i CMS	video blir lastet opp	video: myFixture2.mp4	video blir lastet opp	Pass			trykk på knapp åpner en fil-velger, etter valgt fil, kan bruker sjekke preview for bekreftelse, samt ligger filnavn ved siden av knappen
AC-0008	som admin ønsker jeg å slette bilde i CMS	bilde blir slettet		bilde blir slettet	Pass			trykk på knapp sletter bilde, kan se i preview for å bekrefte, her må lagre knapp trykkes for å bekrefte sletting
AC-0009	som admin ønsker jeg å slette video i CMS	video blir slettet		video blir slettet	Pass			trykk på knapp sletter video, kan se i preview for å bekrefte, her må lagre knapp trykkes for å bekrefte sletting
AC-0010	som admin ønsker jeg å logge ut når jeg er ferdig å jobbe	Logget ut		blir tatt til microsoft logg ut side	Pass			her blir bruker tatt til logg ut siden til microsoft hvor taste trykk logger bruker helt ut

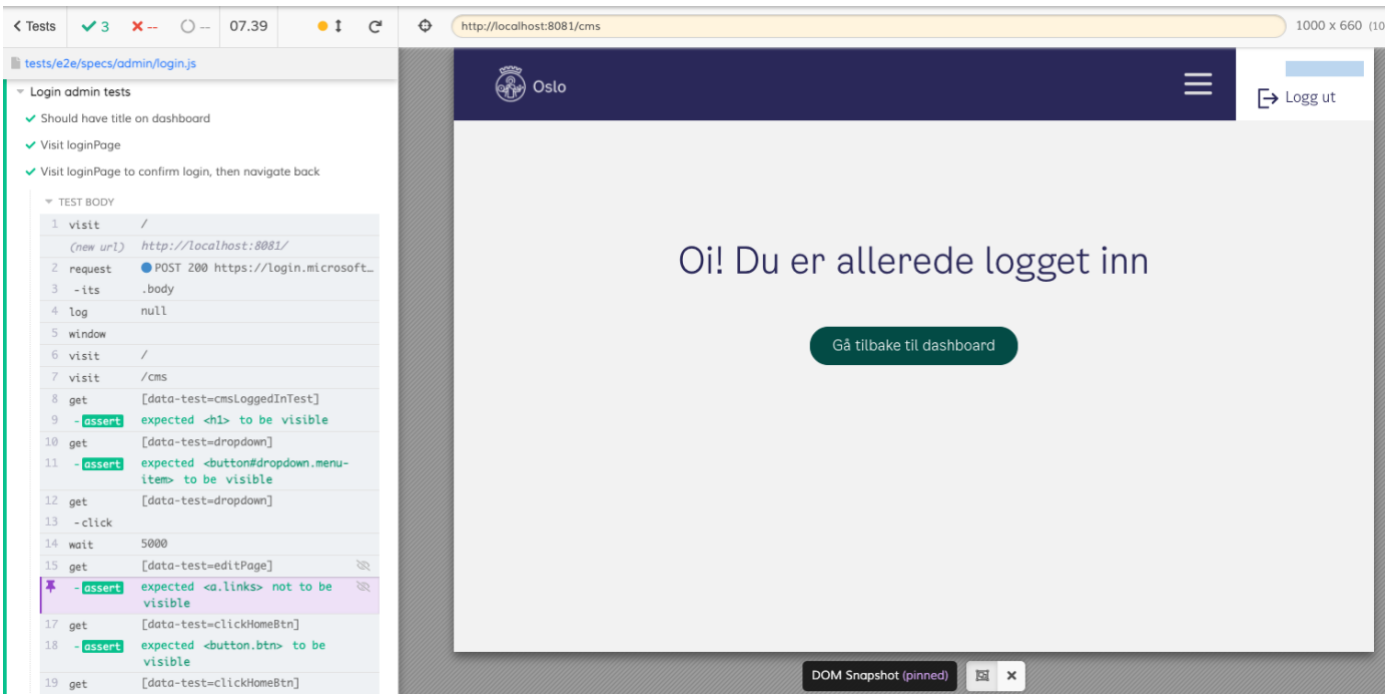
Figur 65 - Oversikt systemtest admin-siden

7.5.2.2 Nedtrekksmeny test feilet

Figur 66 viser funn av feil i nedtrekksmenyen under kjøring av systemtest. Feilen var at «endre side» knappen ikke skulle vært tilgjengelig på denne siden i applikasjonen. Etter feilretting ble ny test gjennomført, og figur 67 viser resultatet etter feilretting.



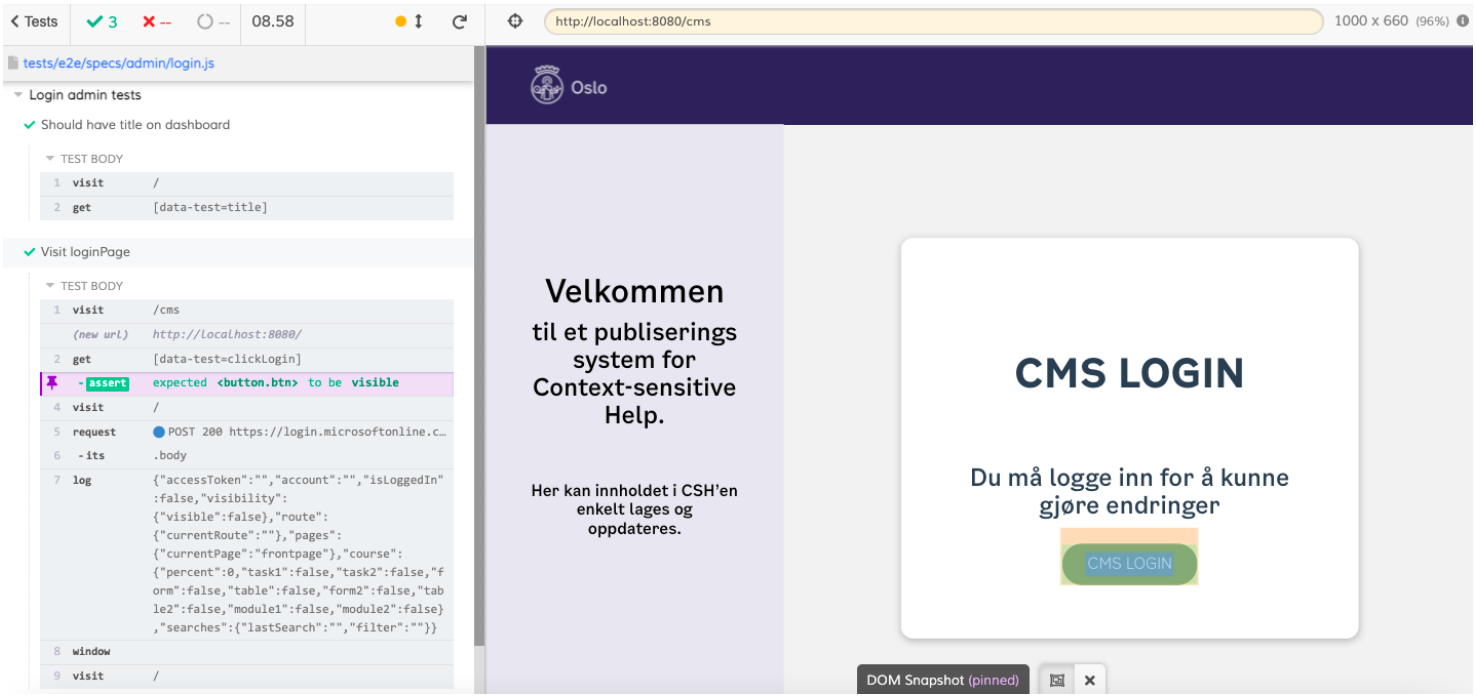
Figur 66 - Logget inn side feil



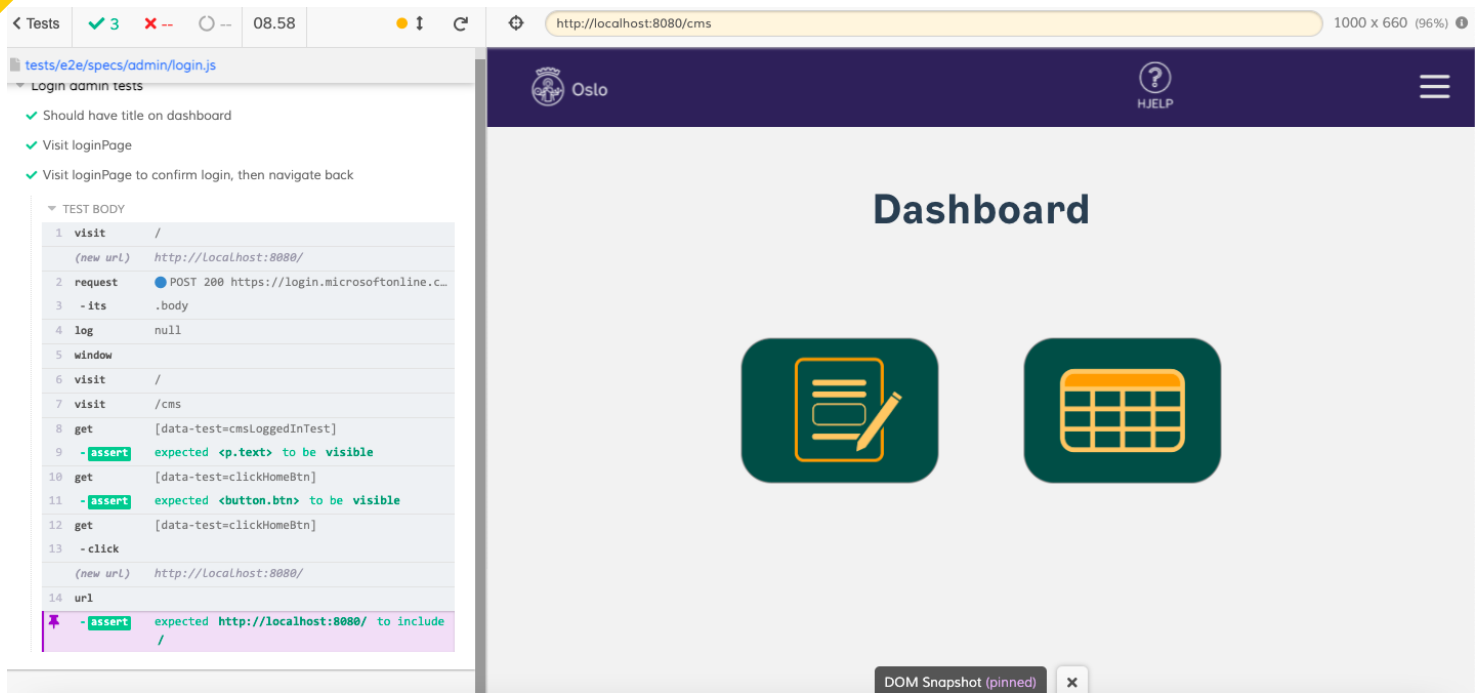
Figur 67 - Logget inn side riktig

7.5.2.3 Admin eksempel

I figur 68 og 69 er det utført en test for innlogging, og figur 70 viser oppsettet av testen. Oppsettet inkluderer brukerhistorien for den gitte testen i en kommentar, før testen beskrives og hver av testtilfellene spesifiseres. Det ble brukt en metode som etterlignet oppførselen til Azure AD (Active Directory), hvor en POST-forespørsel sendes direkte for å unngå innloggings-GUI (se figur 71). Etter at bruker er logget inn utføres det en test for å kunne trykke på hjem-knappen som skal navigere bruker til URL'en for dashboard. Testene kjørte feilfritt.



Figur 68 - AdminLogin i Cypress (del1)



Figur 69 - AdminLogin i Cypress (del2)

```
/*
Som admin ønsker jeg å kunne logge inn i CMS
*/
describe("Login admin tests", () => {
  it("Should have title on dashboard", () => {
    cy.visit("/");
    cy.get("[data-test=title]");
  });

  it("Visit loginPage", () => {
    cy.visit("/cms");
    cy.get("[data-test=clickLogin]").should("be.visible");
    /*
    Reason for not clicking button:
    click on button opens microsoft login,
    wont work in cypress because of bot blocking by microsoft
    */
    cy.login();
  });

  it("Visit loginPage to confirm login, then navigate back", () => {
    cy.login();
    cy.visit("/cms");
    cy.get("[data-test=cmsLoggedInTest]").should("be.visible");
    cy.get("[data-test=clickHomeBtn]").should("be.visible");
    cy.get("[data-test=clickHomeBtn]").click();
    cy.url().should("contain", "/");
  });
});
```

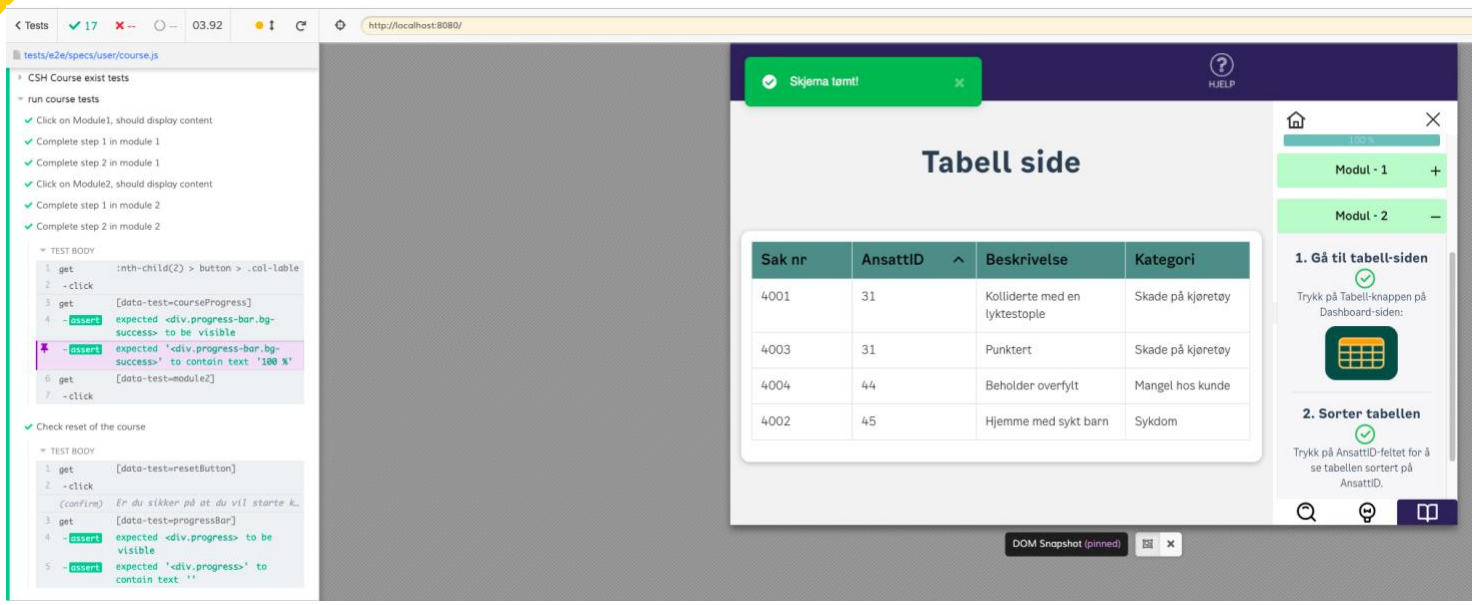
Figur 70 - AdminLogin kode

```
import "cypress-file-upload";  42.1K (gzipped: 12.9K)

Cypress.Commands.add("login", () => {
  cy.visit("/");
  cy.request({
    method: "POST",
    url: `https://login.microsoftonline.com/${Cypress.config(
      "tenantId"
    )}/oauth2/token`,
    form: true,
    body: {
      grant_type: "client_credentials",
      client_id: Cypress.config("clientId"),
      client_secret: Cypress.config("clientSecret")
    }
  })
    .its("body")
    .then(response => {
      localStorage.setItem("jwtToken", response.access_token);
      cy.log(localStorage.getItem("vuex"));
      cy.window().then(win => {
        win.__store__.commit("setAuth", { isAuth: true });
        win.__store__.commit("setAccessToken", {
          token: response.accessToken
        });
      });
    });
  cy.visit("/");
});
```

Figur 71 - Login kommando

7.5.2.4 Bruker eksempel

I figur 72 er det utført en test av kurssiden i CSH, og figur 73 viser et utsnitt av oppsettet til testen. Figuren inkluderer brukerhistorien for den gitte testen i en kommentar, før testen beskrives og hver av testtilfellene spesifiseres. Først navigeres det til kurssiden før det gjøres en sjekk om den har noe innhold. Det testes så for åpning av den første modulen, utførelse av den gitte oppgaven, at den er utført riktig og dens fremdriftslinje øker. Videre utføres tilsvarende for den andre modulen. Testene kjørte feilfritt.



Figur 72 - kursmodul i Cypress

```
/*
Som en bruker ønsker jeg å ha tilgang til et kurs
*/
describe("CSH Course exist tests", () => {…
});

/*
Som bruker ønsker jeg å gjennomføre et kurs,
for å få opplæring i systemet
 */
describe("run course tests", () => {
  //Modul 1
  it("Click on Module1, should display content", () => {
    cy.get("[data-test=module1]").click();
    cy.get("[data-test=subModule1]")
      .should("be.visible")
      .and("contain.text", "1. Gå til skjema-siden");
    cy.get("[data-test=round]").should("be.visible");
    cy.get("[data-test=mod1Img]").should("be.visible");
  });

  it("Complete step 1 in module 1", () => {
    cy.get("[data-test=clickForm]").click();
    cy.url().should("include", "/form");
    cy.get("[data-test=check]").should("be.visible");
    cy.get("[data-test=courseProgress]")
      .should("be.visible")
      .and("contain.text", "25 %");
  });

  it("Complete step 2 in module 1", () => {
    cy.get("[data-test=caseNr]").should("be.visible");
    cy.get("[data-test=caseNr]").type("1");
    cy.get("[data-test=empId]").should("be.visible");
    cy.get("[data-test=empId]").type("1");
    cy.get("[data-test=description]").should("be.visible");
    cy.get("[data-test=description]").type("test");
    cy.get("[data-test=category]").should("be.visible");
    cy.get(":nth-child(6) > [data-test=category]").click();
    cy.get("[data-test=knapp]").should("be.visible");
    cy.get("[data-test=knapp]").click();
    cy.get("[data-test=courseProgress]")
      .should("be.visible")
      .and("contain.text", "50 %");
    cy.get("[data-test=module1]").click();
  });
```

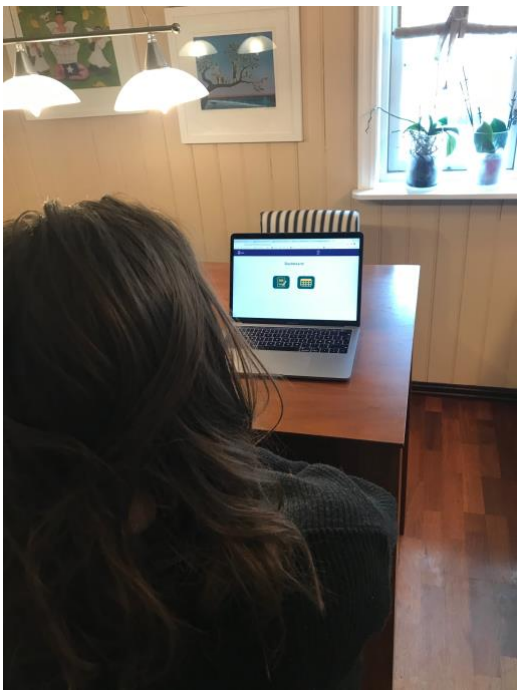
Figur 73 - Utsnitt av kurs test-kode

7.5.3 Brukertest

Etter å ha gjennomført brukertesting på til sammen 6 personer i ulik alder og kjønn, sto vi igjen med et generelt positivt resultat. Problemer med forståelse var i stor grad knyttet til bruken av ulike ikoner og dummy-innhold. Den generelle forståelse av hvordan CSH'en fungerte var god, og det kom flere tilbakemeldinger på at testpersonene ønsket denne typen hjelp i flere applikasjoner.

Testperson 1 (mann, 25 år):
Testpersonen hadde ingen problemer med å navigere seg rundt i dummy-applikasjonen. I CSH'en hadde han derimot problemer med å forstå at tilbakeknappen kun gikk tilbake til forsiden. Dette var knyttet til at knappen var utformet som en pil, lik den man finner i nettlesere osv. Testpersonen forventet at knappen førte han tilbake til den siden han sist var på. Han forsto heller ikke at knappen ikke fungerte når den var grået ut. I CMS'en hadde testpersonen først problemer med å forstå hvordan den fungerte og sammenhengen mellom input feltene og forhåndsvisningen. Dette løste seg likevel så fort han gjorde en endring i et felt.

Testperson 2 (kvinne, 24 år)
Testpersonen hadde ingen problemer med å navigere seg rundt i dummy-applikasjonen. I CSH'en skjønte hun ikke hva lyspære-ikonet skulle bety. Tannhjul-ikonet forbandt hun med klassiske innstillinger, og innholdet bak gav derfor ikke mening i hennes øyne. Hun foreslo videre at det ville vært enklere å forstå ikonene dersom en tekstlig forklaring var tilgjengelig. CMS'en hadde hun ingen problemer med å forstå.



Figur 74 - Testperson 2 under brukertesten

Testperson 3 (mann, 20 år)

Testpersonen hadde ingen problemer med å navigere seg rundt i dummy-applikasjonen. I CSH'en hadde han problemer med å forstå FAQ'en ettersom denne på dette tidspunktet ikke inneholdt relevant innhold. Han hadde også problemer med å forstå sammenhengen mellom CSH'en og dummy-applikasjonen av samme grunn. I likhet med Testperson 1, trodde han tilbake-knappen førte til siden han sist hadde vært på. I likhet med Testperson 2, hadde han problemer med å forstå tannhjul-ikonet. Resten av CMS'en hadde han ingen problemer med å forstå.

Testperson 4 (mann, 62 år)

Testpersonen hadde hørt om applikasjonen på forhånd, og det er å anta at dette bidro til et positivt resultat. Han hadde en god forståelse av applikasjonen og gjennomførte alle oppgavene uten problemer.

Testperson 5 (kvinne, 45 år)

Testpersonen hadde ingen problemer med å navigere seg rundt i dummy-applikasjonen. Hun skjønte ikke hvilken funksjon tannhjulet hadde, frem til hun trykket seg til de ulike komponent sidene. Hun hadde problemer med å forstå logoen til

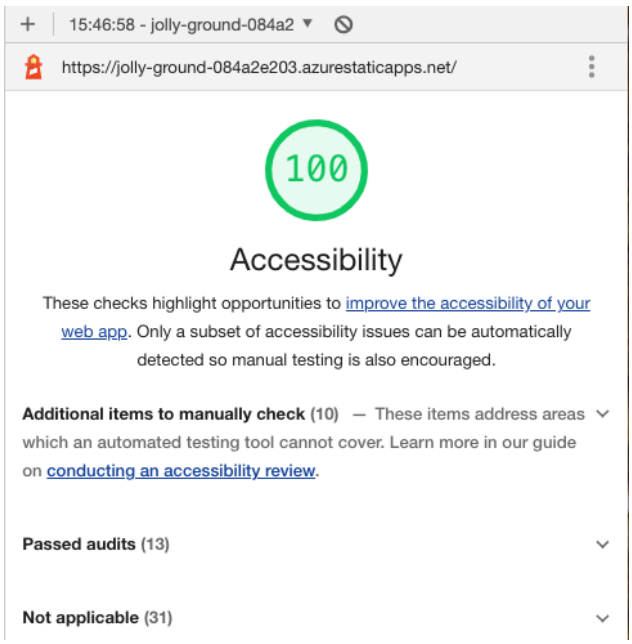
kursmodulen, men skjønte raskt da hun åpnet siden. Kursmodulen hjalp dermed veldig på forståelsen av dummy-sidene og arkitekturen som var tiltenkt. Hun likte også det at det var fremvist en forhåndsvisning i CMS'en. Dette primært på bakgrunn av at det systemet hun bruker på jobb i dag har to forskjellige brukergrensesnitt for relativ lik funksjonalitet.

Testperson 6 (kvinne 49 år)
Testpersonen hadde ingen problemer med å navigere seg rundt i dummy-applikasjonen. Hun skjønte at burger-ikonet betydde nedtrekksmeny. Testperson slet med å finne FAQ, det grunnet ikonbruk, men forstod det da hjelpeteksten dukket opp. Denne funksjonaliteten likte hun, og utnyttet senere i brukertesten. Brukeren likte generelt CSH løsningen som en helhet, spesielt etter forståelse av ikoner som blir brukt. Eneste kommentar på CMS siden var at hun likte forhåndsvisningen av hvordan CSH ville se ut. Som nevnt var det kun oppgaven med å finne FAQ-siden som tok lengre tid, ellers ingen vanskeligheter.

7.5.4 Tilgjengelighet

Testing av tilgjengelighet ved hjelp av skjermleser avslørte ingen feil i applikasjonen. Til vår overraskelse fanget skjermleseren opp mer enn forventet. Skjermleseren fortsatte å lese varslinger selv etter tidsbegrensningen var gått ut. Vi kan med det konstatere at applikasjonen kan brukes av de som benytter skjermleser.

Ved hjelp av Lighthouse fikk vi automatisk testet en mengde tilgjengelighets punkter. Testen avslørte ingen feil (se figur 75), men som rapporten også poengterer dekket ikke denne testen i nærheten av alle punktene. Rapporten hadde heller ingen henvisninger til WCAG, noe som gjorde det vanskelig å få oversikt over hvilke krav som var testet og hvilke krav som manglet. Applikasjonen ble derfor testet manuelt, og feil som ble oppdaget ble rettet. Oversikten over punktene i WCAG ble oppdatert og endelig resultat viste at applikasjonen tilfredsstiller nivå A og AA med unntak av punkt 2.2.1 og 2.4.5 (se figur 76). Mer om de ulike punktene og begrunnelser for valgene som er gjort kan leses under kapittel 6 om design.



Figur 75 - Lighthouse rapport

Punkt	Nivå	Ferdig?
1.1.1	A	Ja
1.2.1	A	Irrelevant
1.2.2	A	Irrelevant
1.2.3	A	Irrelevant
1.2.4	AA	Irrelevant
1.2.5	AA	Irrelevant
1.2.6	AAA	Irrelevant
1.2.7	AAA	Irrelevant
1.2.8	AAA	Irrelevant
1.2.9	AAA	Irrelevant
1.3.1	A	Ja
1.3.2	A	Ja
1.3.3	A	Ja
1.3.4	AA	Ja
1.3.5	AA	Ja
1.3.6	AAA	Ja
1.4.1	A	Ja
1.4.2	A	Irrelevant
1.4.3	AA	Ja
1.4.4	AA	Ja
1.4.5	AA	Ja
1.4.6	AAA	Ja
1.4.7	AAA	Ja
1.4.8	AAA	Nei
1.4.9	AAA	Nei
1.4.10	AA	Ja
1.4.11	AA	Ja
1.4.12	AA	Ja
1.4.13	AA	Ja
2.1.1	A	Ja
2.1.2	A	Ja
2.1.3	AAA	Ja
2.1.4	A	Irrelevant
2.2.1	A	Nei
2.2.2	A	Irrelevant
2.2.3	AAA	Nei
2.2.4	AAA	Nei
2.2.5	AAA	Nei
2.2.6	AAA	Nei
2.3.1	A	Irrelevant
2.3.2	AAA	Irrelevant
2.3.3	AAA	Irrelevant
2.4.1	A	Ja
2.4.2	A	Ja
2.4.3	A	Ja
2.4.4	A	Ja
2.4.5	AA	Nei
2.4.6	AA	Ja
2.4.7	AA	Ja
2.4.8	AAA	Nei
2.4.9	AAA	Nei
2.4.10	AAA	Nei
2.5.1	A	Irrelevant
2.5.2	A	Ja
2.5.3	A	Ja
2.5.4	A	Irrelevant
2.5.5	AAA	Irrelevant
2.5.6	AAA	Ja
3.1.1	A	Ja
3.1.2	AA	Irrelevant
3.1.3	AAA	Nei
3.1.4	AAA	Nei
3.1.5	AAA	Nei
3.1.6	AAA	Nei
3.2.1	A	Ja
3.2.2	A	Ja
3.2.3	AA	Ja
3.2.4	AA	Ja
3.2.5	AAA	Ja
3.3.1	A	Irrelevant
3.3.2	A	Ja
3.3.3	AA	Irrelevant
3.3.4	AA	Irrelevant
3.3.5	AAA	Ja
3.3.6	AAA	Irrelevant
4.1.1	A	Ja
4.1.2	A	Ja
4.1.3	AA	Ja

Figur 76 - WCAG oversikt

8. Kjente feil og mangler

8.1 Deaktivering av knapper

Gjennom å deaktivere knapper når de ikke har noen funksjon, kan man unngå at brukerne opplever at knapper ikke fungerer. «Start kurs på nytt» knappen på kurs-siden i CSH'en er et eksempel på en slik knapp. Når det ikke er progresjon i kurset, vil ikke trykk på knappen utgjøre noen forskjell.

Samme problematikken gjelder for øvrig også på knapper som sender inn skjemaer. I slike tilfeller er det vanlig å kombinere deaktiveringen med validering av input feltene, slik at knappen ikke blir tilgjengelig før skjemaet er klart til å sendes inn. I vår applikasjon er det grunnet demonstrasjonsformålet ikke brukt tid på å implementere validering av skjemaer. Validering er noe samtlige gruppemedlemmer har vært bort i gjentatte ganger gjennom studiet. Manglende læringsutbytte var derfor også med i nedprioriteringen av denne funksjonaliteten.

8.2 Kursmodul 1 oppgave 2

Oppgave 2 i kursmodul 1 består av å fylle ut et skjema og tømme det. I oppgaveteksten står det "Fyll ut alle felter og trykk deretter på 'Tøm skjema'-knappen". Dersom en bruker trykker på knappen før hen har fylt ut alle feltene i skjemaet, vil skjemaet tømmes uten at oppgaven godkjennes. Her burde brukeren få en beskjed, om at oppgaven ikke er godkjent fordi alle feltene ikke var utfylt, når hen tømte skjemaet. Når brukeren ikke får noe respons på handlingen vil hen raskt kunne tenke at oppgaven ikke fungerer slik den skal.

8.3 Avbryte endringer i CMS'en

Dersom brukeren ønsker å avbryte en endring av innhold i CMS'en, har hen en knapp med dette formålet tilgjengelig. Ved trykk på knappen må brukeren bekrefte handlingen, for å unngå at hen uten hensikt avbryter og sletter endringene. Uheldigvis kan det fremdeles skje ved at brukeren trykker på Oslo Kommune logoen i navigasjonsbaren. Her burde enten knappen vært deaktivert eller fått samme

funksjonalitet som «Avbryt» knappen, når brukeren befinner seg på endre-siden i CMS'en.

9. Referanser

Bootstrap. (u.å). *Overview.* Hentet 18. mai 2021 fra https://getbootstrap.com/docs/4.0/layout/overview/

Cypress. (u.å). *Launching Browsers.* Hentet 19. april 2021 fra https://docs.cypress.io/guides/guides/launching-browsers#Browsers

Forskrift om universell utforming av IKT-løsninger. (2013). *Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger* (FOR-2013-06-21-732). Lovdata. https://lovdata.no/dokument/SF/forskrift/2013-06-21-732

Full scale. (2020, 28. august). The best CSS unit for responsive Web design. *Full scale.* https://fullscale.io/blog/best-css-unit-for-responsive-web-design/

GitHub. (2021). *About billing for GitHub Actions.* GitHub Docs hentet 9. mai 2021 fra https://docs.github.com/en/github/setting-up-and-managing-billing-and-payments-on-github/managing-billing-for-github-actions/about-billing-for-github-actions

Henry, S. (2021, 29.april). *Web Content Accessibility Guidelines (WCAG) Overview.* W3C. https://www.w3.org/WAI/standards-guidelines/wcag/#intro

Jansen, A. (2018, 16. november). *Kravdefinisjon og kravspesifikasjon.* ndla https://ndla.no/subject:25/topic:1:193104/topic:1:123578/resource:1:123591?filters=urn:filter:9d6d3241-014d-4a5f-b0bc-ae0f83d1cd71

Microsoft. (2019, 27. november). *Supported languages in Azure Functions*. https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages

Microsoft. (2020, 20. november). *Introduction to Azure Functions*. https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview

Microsoft. (2020, 5. juni). *What is Azure Active Directory*. https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis

Microsoft. (2021, 9.april). *Github Actions workflows for Azure Static Web Apps*. https://docs.microsoft.com/en-us/azure/static-web-apps/github-actions-workflow

Microsoft. (2021, 1. april). *What is Azure Static Web Apps*. https://docs.microsoft.com/en-us/azure/static-web-apps/overview

Mozilla. (u.å). *Using media queries.* Hentet 18. mai 2021 fra https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

Npm. (2021, 12. mai). *Microsoft Authentication Library for JavaScript (MSAL.js) 2.0 for Browser-Based Single-Page Applications.* https://www.npmjs.com/package/@azure/msal-browser

Oslo kommune. (u.å). *Farger* [bilde]. Hentet fra https://designmanual.oslo.kommune.no/farger

Oslo kommune. (u.å). *Font* [bilde]. Hentet fra https://designmanual.oslo.kommune.no/skrifttype

Sommerville, I. (2018). *Software Engineering* (10. utg.). Pearson.

STF. (2020, 17. september). *Black Box Testing*. Software Testing Fundamentals. https://softwaretestingfundamentals.com/black-box-testing/

STF. (2020, 21. september). *System testing.* Software Testing Fundamentals. https://softwaretestingfundamentals.com/system-testing/

STF. (2020, 13. september). *Unit testing.* Software Testing Fundamentals. https://softwaretestingfundamentals.com/unit-testing/

STF. (2020, 6. september). *Usability testing.* Software Testing Fundamentals. https://softwaretestingfundamentals.com/usability-testing/

Uutilsynet. (u.å). *EUs webdirektiv (WAD).* Hentet 11. mai 2021 fra https://www.uutilsynet.no/webdirektivet-wad/eus-webdirektiv-wad/265#forholdet_til_wcag_21

Vuex. (2021, 10. mai). *What is Vuex*. https://next.vuex.vuejs.org/

W3C. (2019, 4. oktober). *How to Meet WCAG (Quick Reference).* https://www.w3.org/WAI/WCAG21/quickref/

W3schools. (u.å). *HTML Responsive Web Design.* Hentet 17. mai 2021 fra https://www.w3schools.com/html/html_responsive.asp

Ødegaard, O. (2019, 4. april). Kravspesifisering. *Spire consulting.* https://blogg.spireconsulting.no/?p=570

BRUKERDOKUMENTASJON

Gruppe 8

OSLOMET - BACHELOR 2020 / 2021

Forord

Denne brukerdokumentasjonen gjelder for brukere av CSH demo applikasjon. Applikasjonen er laget av bachelorgruppe 8, i samarbeid med Norconsult Informasjonssystemer (NoIS). Applikasjonen består av tre deler: En dummy-applikasjon, en kontekstsensitiv hjelp (CSH) og et publiseringssystem (CMS).

Bruksområde vil i første omgang være under innsalg av kontekstsensitiv hjelp til kunder av NoIS. I tillegg til demonstrasjonsformålet har applikasjonen primært to typer sluttbrukere: Innholdsskaper og standard bruker. Brukermanualen er strukturert av hensyn til disse to sluttbrukerene. Bruk av dummy-applikasjonen og den kontekstsensitive hjelpen blir gjennomgått i manualen for bruker, mens publiseringssystemet blir gjennomgått i manualen for innholdsskaper.

I utgangspunktet er det ikke nødvendig med forkunnskaper før bruk av applikasjonen. Vi anbefaler likevel at nye brukere leser gjennom brukermanualen. Etter én gjennomlesing bør bruker ha innsikt i applikasjonens grunnleggende funksjonaliteter.

Teknisk beskrivelse av programmet ligger under Produktdokumentasjon.

Innhold

1. Notasjoner / Terminologi

- CSH = Context-sensitive help (på norsk: Kontekstsensitiv hjelp)
- CMS = Content management system (på norsk: publiseringssystem)
- Innholdsskaper = Personen(e) som skal lage innhold til CSH
- FAQ = Frequently asked questions (på norsk: Ofte stilte spørsmål)
- URL = Uniform Resource Locator (på norsk: Nettadresse)

2. Innledning

CSH demo er en nettbasert applikasjon som er laget for desktop og nettbrett. Applikasjonen er optimalisert for Google Chrome, men fungerer også greit på Mozilla Firefox, Safari og Microsoft Edge. Applikasjonen kan aksesseres via lenken under.

CSH Demo

Applikasjonen i sin helhet fungerer som en demonstrasjon for hvordan en CSH kan gjøre hjelpen mer tilgjengelig for brukere av et system, samt hvordan innhold kan skreddersys på en enklere måte gjennom å bruke en CMS. Forhåpentligvis vil demoen kunne bidra til at flere kunder av NoIS vil se behovet for denne typen brukerhjelp. Det vil igjen kunne bidra til å minske mengden henvendelser til support hos kunden.

Dummy-applikasjonen består av en forside (Dashboard) og to undersider; skjema-siden og tabell-siden. På skjema-siden kan man fylle ut og tømme et skjema og på tabell-siden kan man sortere kolonner i en tabell. Noe videre funksjonalitet er ikke lagt inn ettersom denne delen kun er laget for å vise hvordan en CSH kan fungere.

I navigasjonsbaren på toppen av siden finner man hjelp-knappen som åpner CSH'en. Forsiden i CSH'en tilbyr hjelp til siden man befinner seg på, samt en knapp for å laste ned innholdet i CSH'en i PDF-format. Videre har CSH'en en søkefunksjon, som lar deg søke gjennom den tekstlige informasjonen i CSH'en. Den har en side med ofte stilte spørsmål, en side med kurs og en side der man kan sende inn feilrapportering dersom man finner feil i systemet. På sistnevnte side er ikke funksjonaliteten lagt inn ettersom den bare er laget for å vise at man også kan bruke CSH'en til dette.

Ved å skrive */cms* bak base URL'en blir man sendt til innloggingen til publiseringssystemet (CMS). Etter å ha logget seg inn, vil innholdsskaperen kunne lage nytt innhold eller endre allerede eksisterende innhold i forsiden til CSH'en. Dette inkluderer tittel, brødtekst, ordforklaringer, bilde og video.

Applikasjonen har innebygd to former for tilbakemeldinger til brukere. Dersom feil

oppstår slik at applikasjonen ikke klarer å hente innholdet til CSH'en eller CMS'en, vil en feil-skjerm vises. Tilbakemeldinger som omhandler brukers interaksjon, blir gitt i form av varslinger øverst i venstre hjørne av applikasjonen.

3. Brukermanual

3.1 Manual for bruker

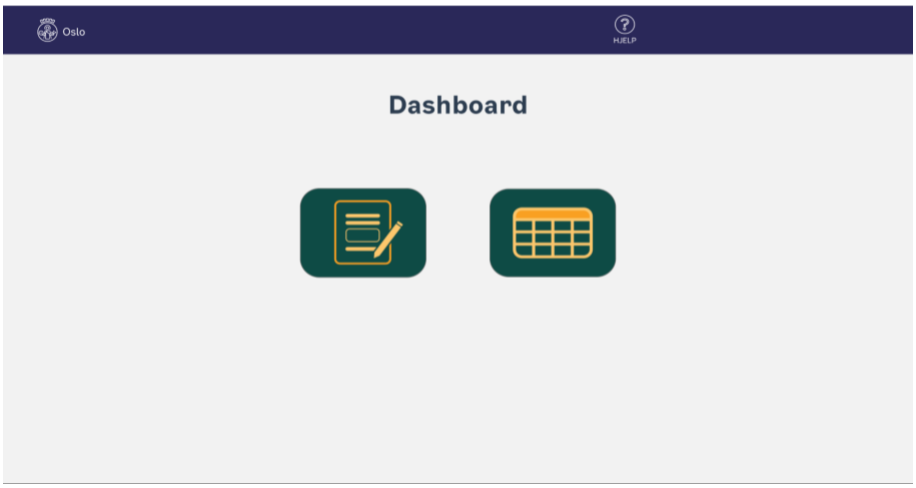
3.1.1 Navigasjonsbar

Navigasjonsbaren er plassert i toppen av skjermen og inneholder to knapper. Den ene er Oslo kommune sin logo, som er en hjem-knapp. Et *trykk* på knappen, fører alltid til Dashboard-siden (se kap. 3.1.2). Den andre knappen inneholder et "?" ikon med teksten "Hjelp". *Trykk* på knappen for å åpne CSH. *Trykk* en gang til for å lukke CSH.

Figur 1 - Navigasjonsbaren

3.1.2 Dashboard (Dummy applikasjon)

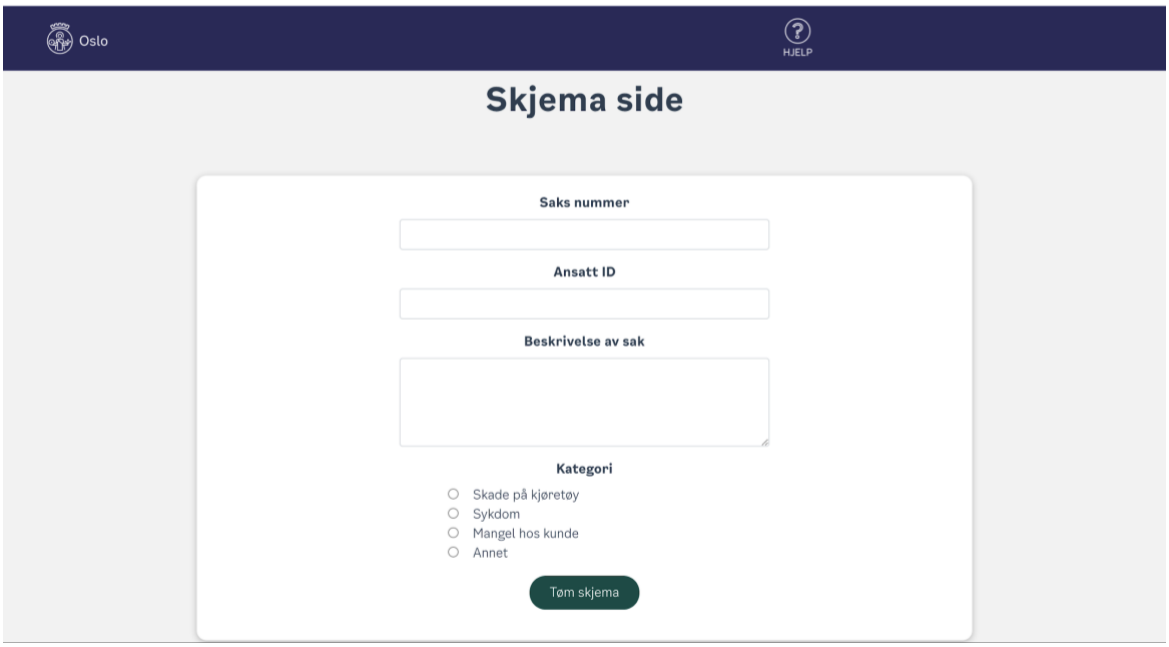
Når du åpner applikasjonen, er dette den første siden du ser (se figur 2). Her ser du to knapper og en overskrift. Overskriften indikerer hvilken side du befinner deg på. Knappene fører til hver sin underside. Knappen til venstre fører til skjema-siden (se kap. 3.1.3) og knappen til høyre fører til tabell-siden (se kap. 3.1.4). Dersom du er usikker på hvilken knapp som fører hvor, hold musepekeren over knappen for å se hjelpetekst.



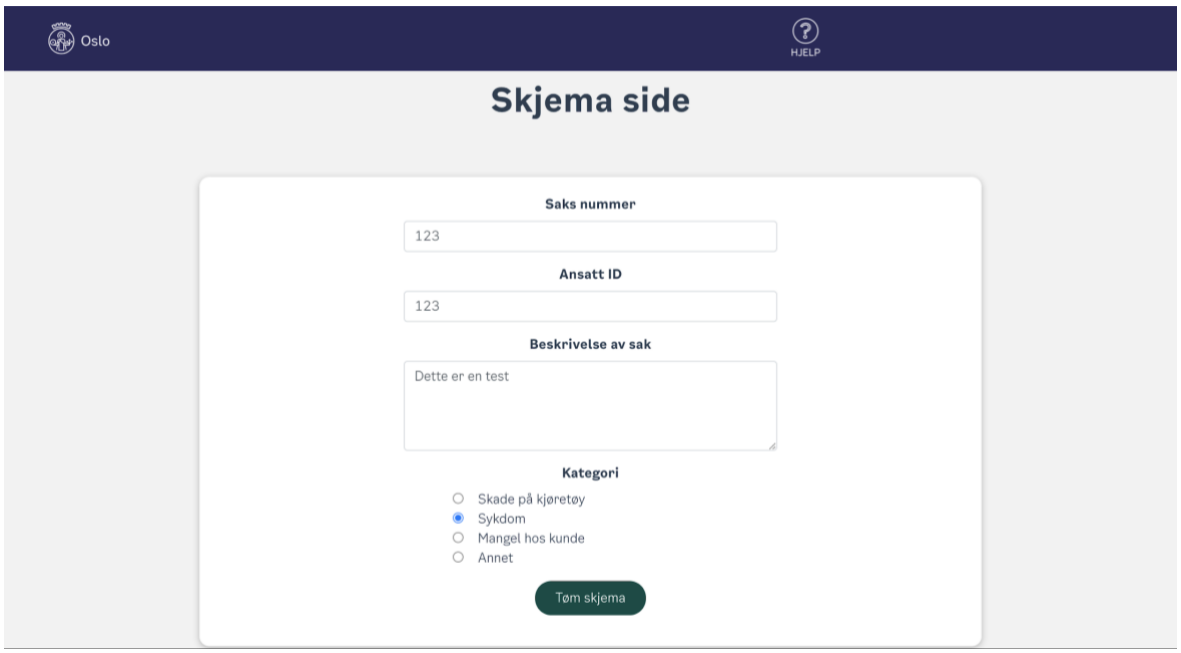
Figur 2 - Forsiden i applikasjonen

3.1.3 Skjema side (Dummy applikasjon)

Du kommer til denne siden ved å *trykke* på skjema-knappen på dashbordet (se kap. 3.1.2). Her kan du fylle ut et skjema og tømme det (se figur 3 og 4). For å fylle ut feltene, *trykk* på feltene og skriv ønsket tekst. Under kategori kan du velge en kategori ved å *trykke* på ett av gitte alternativer. *Trykk* til slutt på knappen for å tømme skjema.



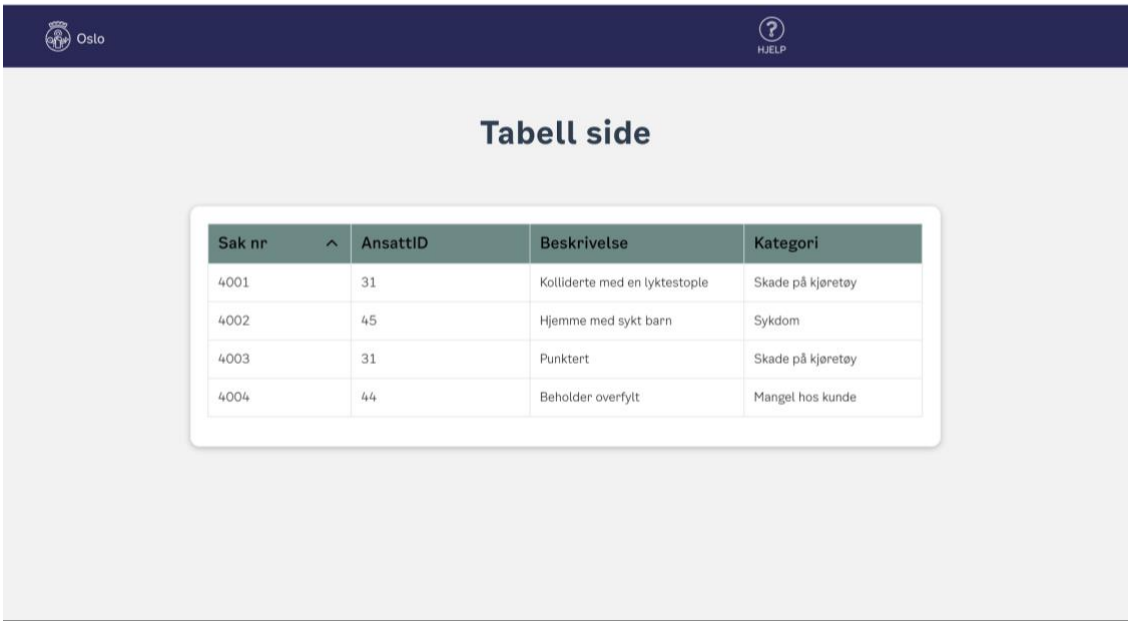
Figur 3 - Skjema-side



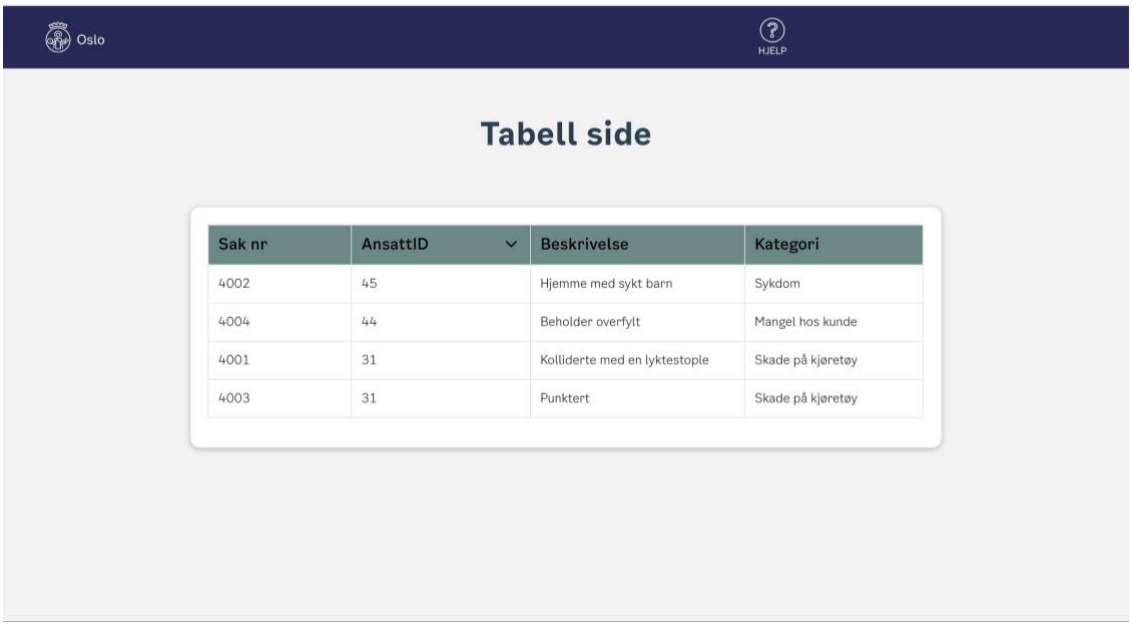
Figur 4 - Skjema utfylt

3.1.4 Tabell-side (Dummy applikasjon)

Du kommer til denne siden ved å *trykke* på tabell-knapp på dashbordet (se kap. 3.1.2). Her kan du se en tabell (se figur 5). Tabellen kan sorteres ved at du *trykker* på ønsket kolonneoverskrift (se figur 6). Ikonet til høyre i kolonneoverskriften indikerer om kolonnen er sortert, samt rekkefølge. Dersom pilen peker opp er kolonnen sortert stigende, og ved pil ned er den synkende.



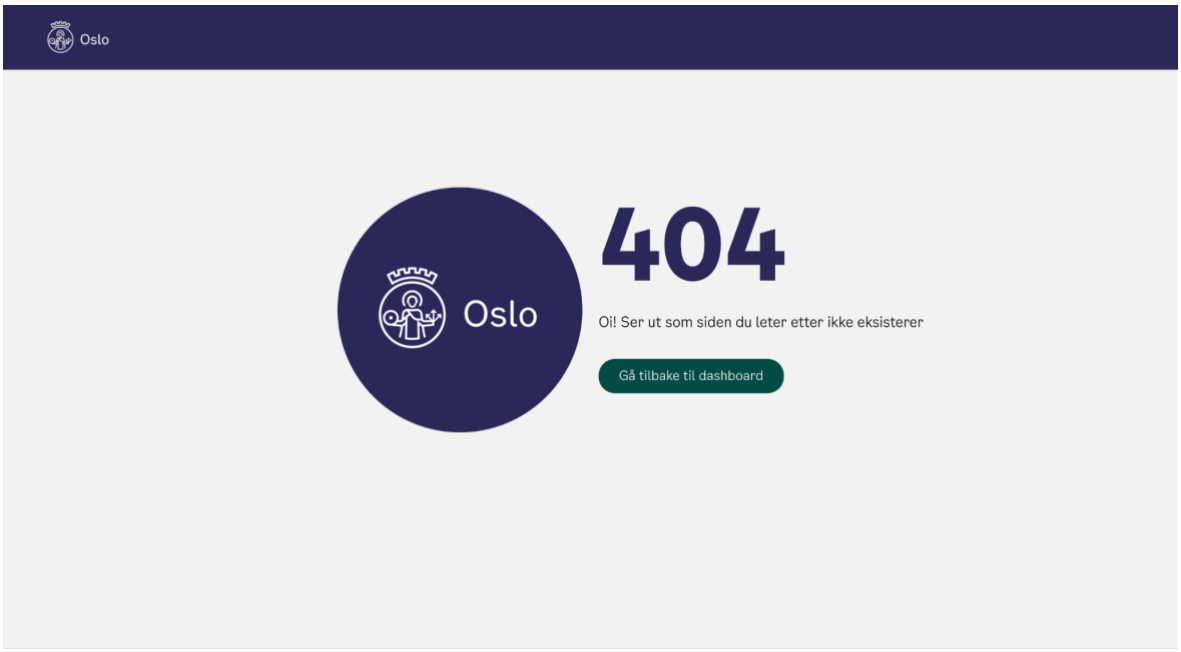
Figur 5 - Tabell-side



Figur 6 - Tabell sortert synkende på Ansatt ID

3.1.5 Feil-side (Dummy applikasjon)

Feil-siden i programmet vises hvis du prøver å navigere deg til en ugyldig URL. Siden inneholder tekst som forklarer problemet og en knapp. *Trykk* på «Gå til dashboard» knapp for å navigere hjem til dashboard (se kap. 3.1.2).



Figur 7 - Feil-side i dummy-applikasjonen

3.1.6 Navigasjon i CSH

Når CSH er åpen, kan du navigere deg rundt ved hjelp av topp- og bunn-navigasjon.

Topp-navigasjonen inneholder alltid et kryss, som ved *trykk* lukker CSH'en. Når du befinner deg på CSH'en sin forside (se kap. 3.1.7), er dette eneste knappen. Befinner du deg i søk-siden (se kap. 3.1.8), på FAQ-siden (se kap. 3.1.9) eller kurs-siden (se kap. 3.1.10), vil topp-navigasjonen også ha en hjem-knapp. Ved *trykk* på hjem-knappen navigeres du tilbake til forsiden. Befinner du deg på feilrapportering-siden (se kap. 3.1.11) vil topp-navigasjonen ha en tilbake-knapp i stedet for hjem-knapp. Ved *trykk* på tilbake-knappen navigeres du tilbake til FAQ-siden.

Figur 8 - Topp-navigasjon med hjem-knapp

Figur 9 - Topp-navigasjon med tilbake-knapp

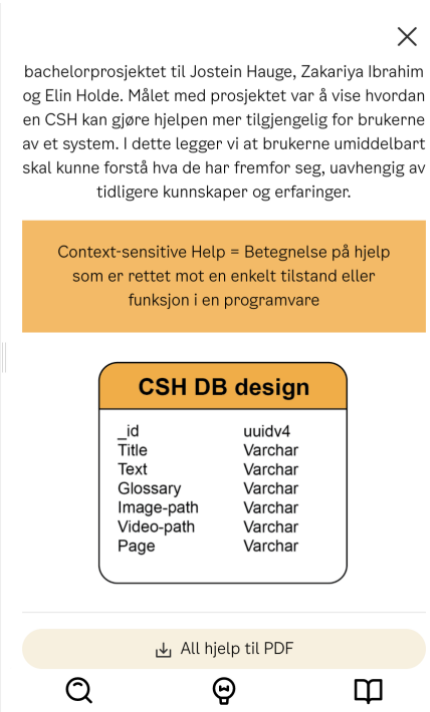
Bunn-navigasjonen inneholder alltid tre knapper med følgende ikoner; lupe-ikon, lyspære-ikon og bok-ikon. Ved *trykk* på lupe-ikonet blir du navigert til søk-siden. Ved *trykk* på lyspære-ikonet blir du navigert til FAQ-siden. Ved *trykk* på bok-ikonet blir du navigert til kurs-siden.

Figur 10 - Bunn-navigasjon

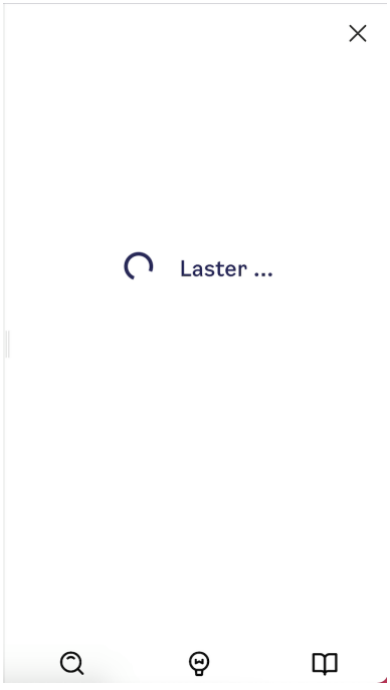
3.1.7 Forside (CSH)

På forsiden i CSH kan du få hjelp til siden du befinner deg på i dummy-applikasjonen. Forsiden kan inneholde følgende: En overskrift, tekst, ordforklaringer, bilde, video og en knapp (se figur 11 og 12). Ved *trykk* på knappen lastes all hjelp ned i form av PDF. Når CSH'en henter hjelp til deg, vises en laster-skjerm (se figur 13). Dersom det ikke finnes hjelp til siden du befinner deg på i dummy-applikasjonen, vil en feil-skjerm fortelle deg at: "Det er dessverre ikke lagt inn hjelp til denne siden enda" (se figur 14). Hvis noe går galt under lasting, vil du få beskjed på feil-skjermen.

Figur 11 - CSH-forside (Del 1)



Figur 12 - CSH-forside (Del 2)



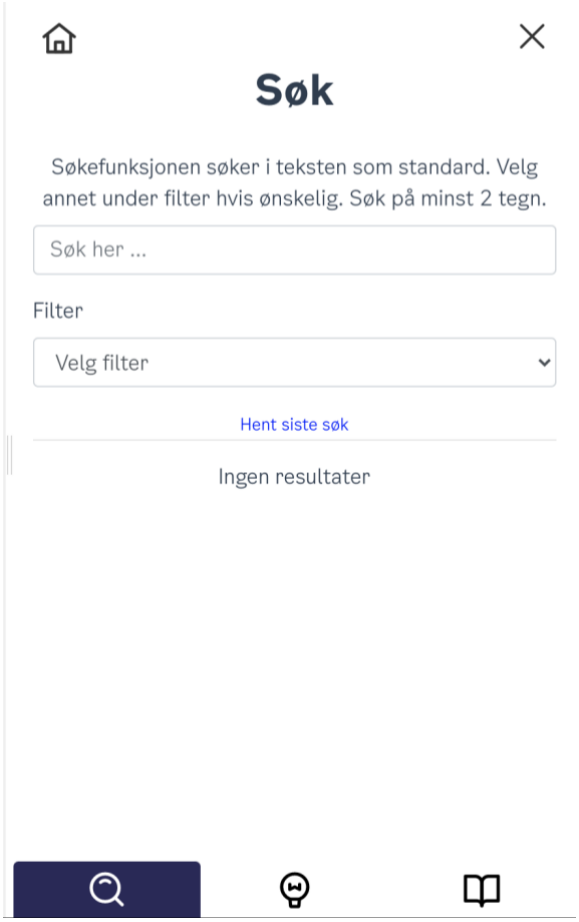
Figur 13 - CSH laster

Figur 14 - CSH feil-skjerm

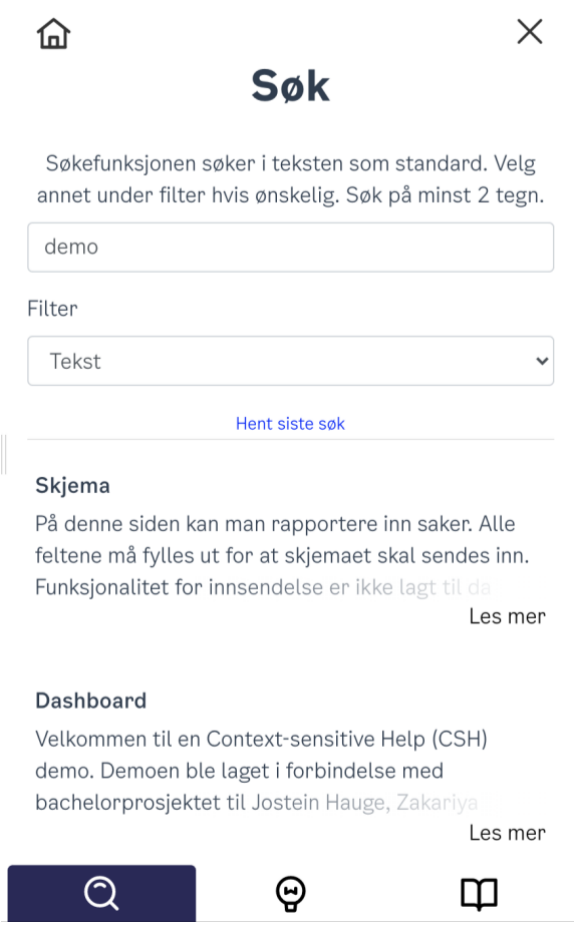
3.1.8 Søk-siden (CSH)

På søkesiden kan du søke gjennom hjelpeteksten fra forsiden i CSH. For å søke i hjelpeteksten skriver du i søkefeltet (se figur 15 og 16). Ønsker du å filtrere søket ditt, velger du et filter i nedtrekksmenyen under søkefeltet. Velg mellom tekst, ordforklaringer og tittel. Som standard er «tekst» valgt.

Når du har gjort et søk, kan du se en liste over resultater. I resultatlisten kan du lese et lite utdrag fra resultatet. For å lese mer kan du *trykke* på «les mer» for å bli navigert til riktig forside. På søkesiden kan du også *trykke* på knappen som sier «hent siste søk», for å hente det siste søket du gjorde.



Figur 15 - Søkefunksjon før søk



Figur 16 - Søkefunksjon etter søk

3.1.9 FAQ-side (CSH)

På FAQ-siden kan du se ofte stilte spørsmål og en knapp (se figur 17). Ved klikk på et av spørsmålene vil svaret vises. Ved *trykk* på «Rapporter feil» vil du navigeres til side for rapportering av feil (se kap. 3.1.10).

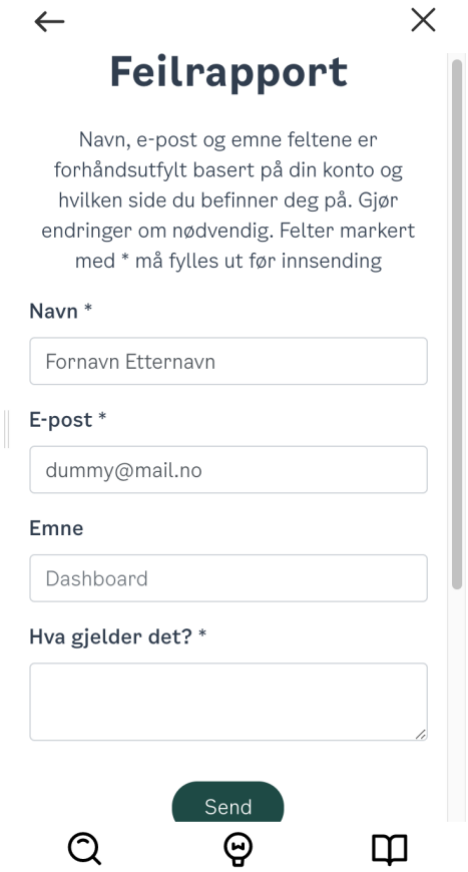
Figur 17 - FAQ, med første spørsmål åpen

3.1.10 Feilrapportering (CSH)

Når du *trykker* på knappen «rapporter feil» på FAQ-siden, kommer du til side for rapportering av feil. Her kan du fylle ut en feilrapport (se figur 18 og 19).

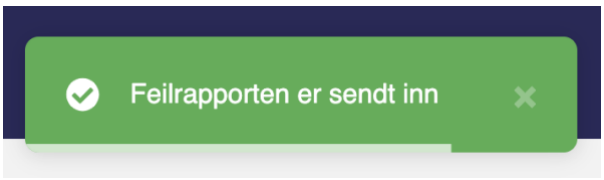
Ettersom applikasjonen skal brukes til demonstrasjonsformål fungerer ikke skjemaet slik som det ellers ville gjort. Feltene «Navn» og «E-post» ville vært forhåndsutfylt med informasjon hentet fra en brukerprofil. Feltene er derfor forhåndsutfylt med “Fornavn Etternavn” og “dummy@mail.no” for å etterligne denne funksjonaliteten, men du kan

gjøre endringer hvis ønskelig. Emne-feltet er forhåndsutfylt med informasjon om siden du befinner deg på i dummy-applikasjonen. Dersom du ønsker å sende en feilrapport som ikke har med denne siden å gjøre, naviger deg til riktig side eller overskriv med ønsket tekst. Til slutt kan du *trykke* på «Send» knappen for å etterligne en innsending av feilrapporten. Feltene vil tømmes og du vil få et varsel om at handlingen er vellykket (se figur 20). Varselet kan pauses ved å holde musepekeren over, eller lukkes før tiden har gått ut ved å *trykke* på krysset til høyre eller sveipe den til høyre eller venstre.



Figur 18 - Feilrapportering (Del 1)

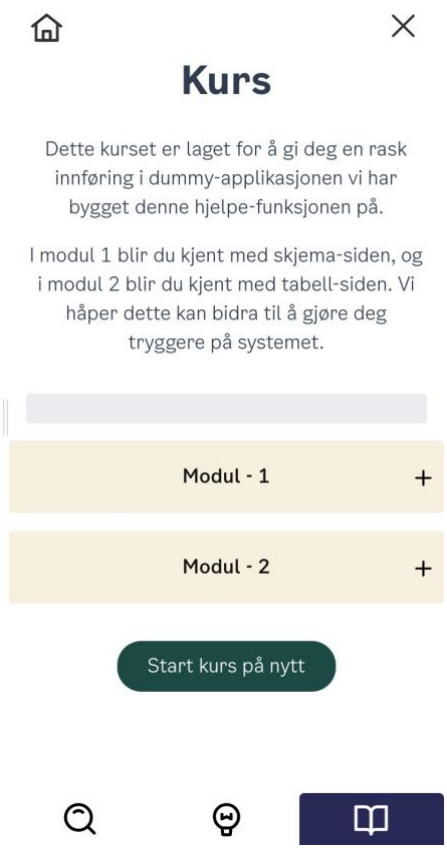
Figur 19 - Feilrapportering (Del 2)



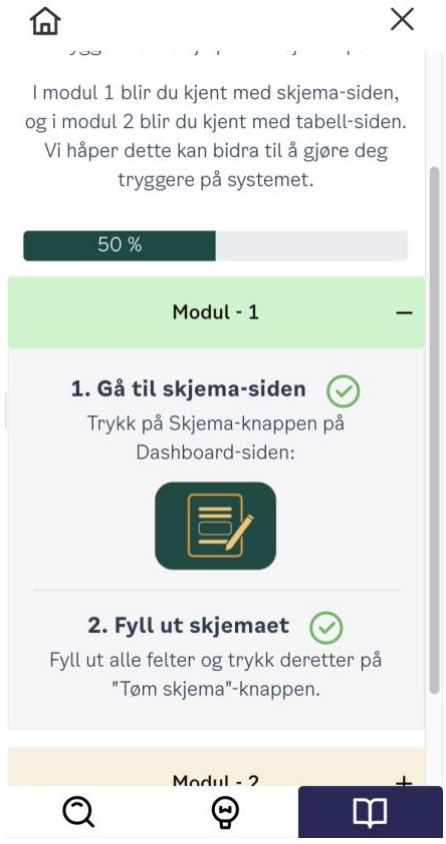
Figur 20 - Varsel når feilrapport er sendt inn

3.1.11 Kurs-side (CSH)

På kurssiden kan du se en progresjonslinje som går fra 0% til 100%, to moduler og en «Start kurs på nytt» knapp (se figur 21). Hver modul kan åpnes og inneholder steg som skal gjøres for å gjennomføre modulen. Når du fullfører en oppgave, vil prosenten på progresjonslinjen øke (se figur 22). Ønsker du å gjennomføre kurset flere ganger, kan du *trykke* på «Start kurs på nytt» knappen nederst på siden. Da vil all progresjonen som er gjort hittil i kurset nullstilles.



Figur 21 - Kurs-moduler lukket



Figur 22 - Modul-1 åpen og kurs 50% gjennomført

3.2 Manual for innholdsskaper:

Denne manualen er for innholdsskapere, og gjelder ikke for den vanlige bruker. Hvis du ikke har fått beskjed om at du skal lage innhold til CSH'en, kan du se bort fra denne manualen.

Systemet vil se likt ut for innholdsskaper og vanlige brukere, men innholdsskapere har tilgang til flere funksjoner gjennom publiseringssystemet (CMS). For å få tilgang til CMS'en, må du først logge inn med gyldig brukernavn og passord.

3.2.1 Logg inn i CMS

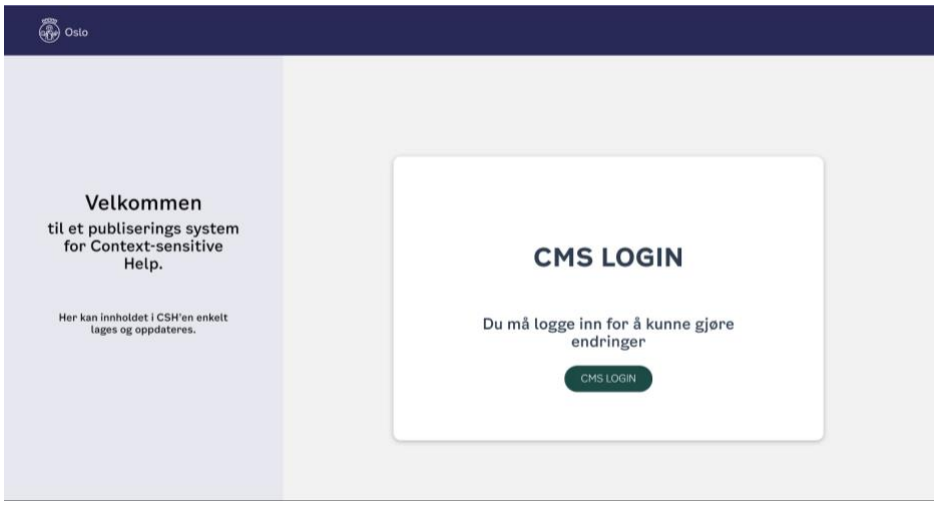
Du kommer til denne siden ved å skrive */cms* bak base URL'en eller *trykk* på lenken under.

Logg inn i CMS her

Innloggingsinformasjon:

- Brukernavn: testuser@oslobachelor2021gmail.onmicrosoft.com
- Passord: BachelorGruppe8!
- Trykk «Hopp over nå (14 dager til dette er nødvendig)»

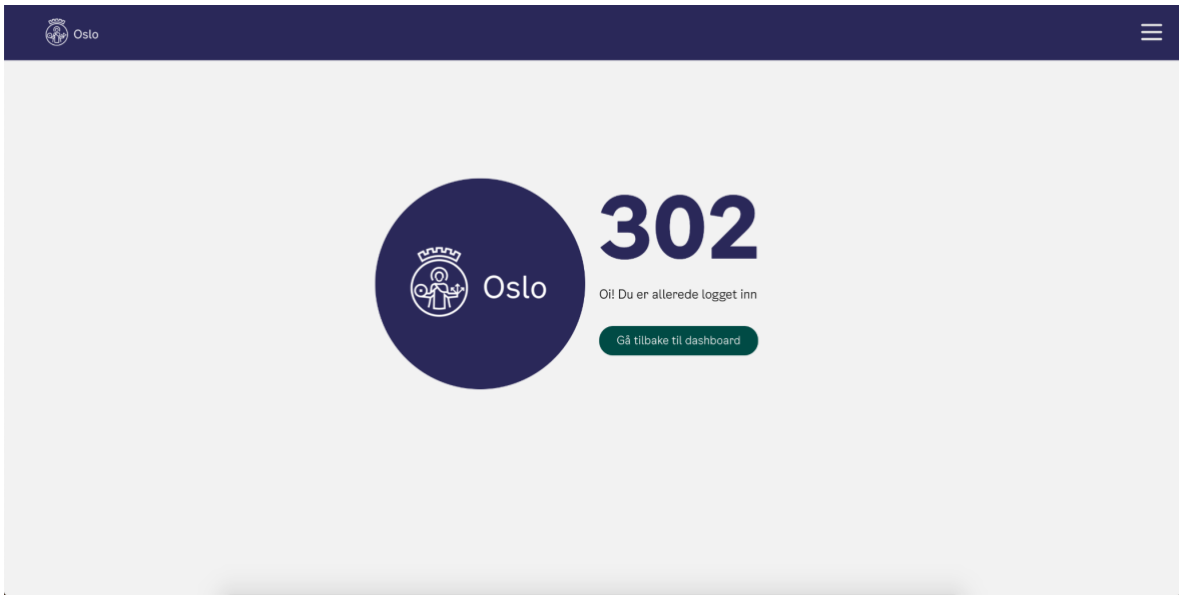
På denne siden vil du se infotekst og en knapp (se figur 23). Når du *trykker* på knappen, vil du få mulighet til å logge inn gjennom Microsoft. Etter å ha logget inn returneres du til dashbordet (se kap. 3.1.2).



Figur 23 - Logg inn i CMS

3.2.2 Allerede logget inn i CMS

Hvis du skriver */cms* bak base URL'en når du allerede er logget inn, vil du bli omdirigert til "allerede logget inn" siden (se figur 24). Her kan du se en tekst som forklarer problemet og en knapp som sier «Gå tilbake til dashboard». *Trykker* du på den, vil du navigeres tilbake til dashboard (se kap. 3.1.2).



Figur 24 -Allerede logget inn i CMS

3.2.3 Nedtrekksmeny (CMS)

Nedtrekksmenyen ligger øverst til høyre i navigasjonsbaren (se kap. 3.1.1) og er symbolisert med et hamburger-ikon (se figur 25). *Trykk* på ikonet for å åpne menyen. Her har du to valg:

Valg 1: *Trykk* på "endre-siden" (se 3.2.4) for å endre innholdet i CSH'en.
Valg 2: *Trykk* på "logg ut" (se figur 26) for å logge ut av CMS'en.

Valg 1 er bare tilgjengelig på dashboard (se kap. 3.1.2), skjema-siden (se kap. 3.1.3) og tabell-siden (se kap. 3.1.4).

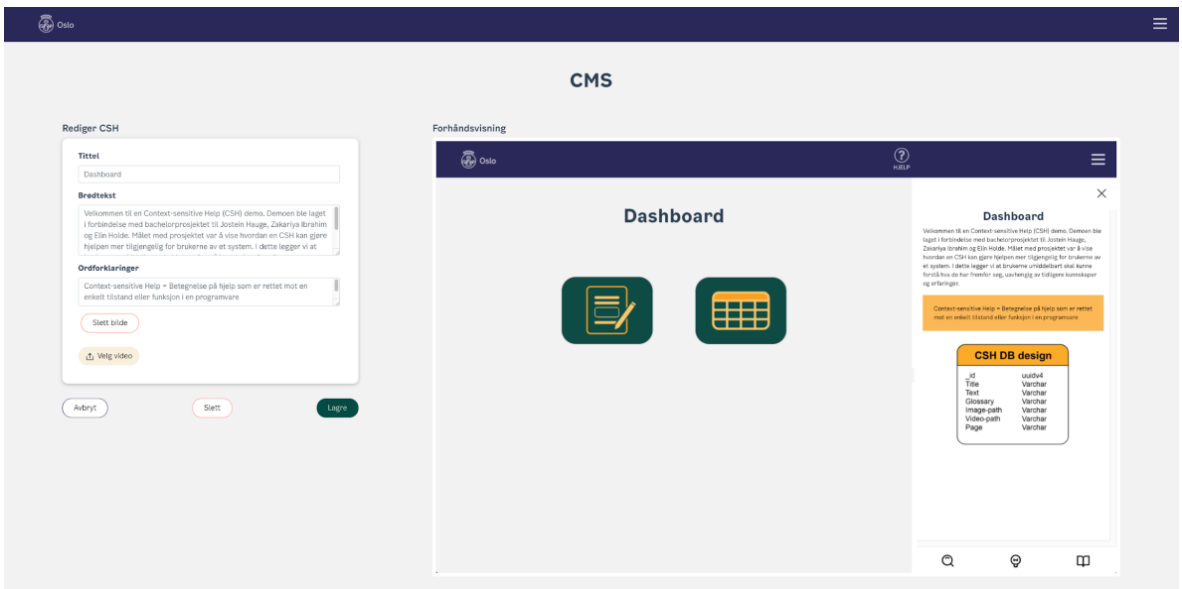
Figur 25 - Nedtrekksmeny lukket

Figur 26 - Nedtrekksmeny åpnet

3.2.4 Endre-side (CMS)

På endre-siden kan du legge til, endre eller slette innholdet som vises i CSH (se figur 27). Siden inneholder to deler:

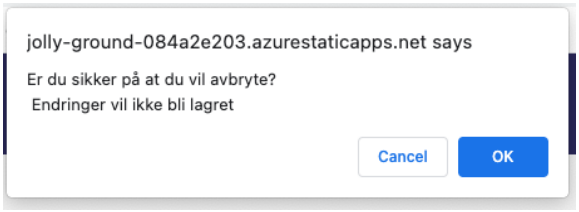
Del 1: Rediger CSH, består av input-felter der du kan skrive inn innhold, og laste opp bilde og video. Etter du har lastet opp bilde eller video, vil du få tilbakemelding om opplastingen er vellykket. Tilbakemelding gis i form av et varsel (se figur 32 under kap. 3.2.4.1).
Del 2: Forhåndsvisning, viser hvordan innholdet i CSH'en vil se ut. Her vil du kunne se endringer som legges inn i del 1 fortløpende.



Figur 27 - Endre-side

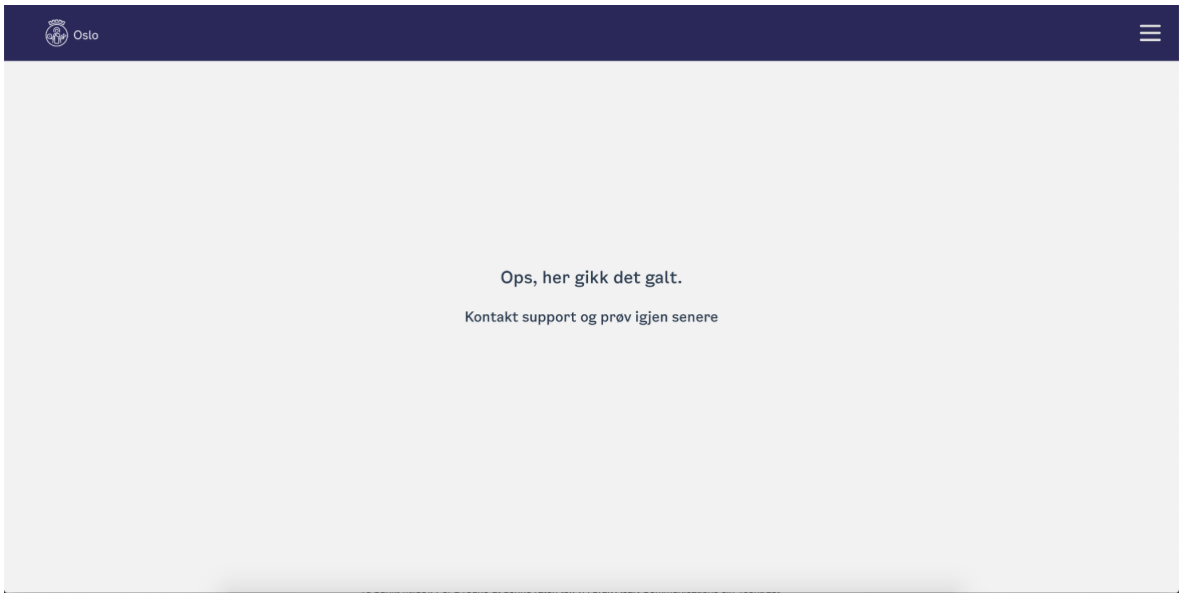
Endre-siden inneholder også tre knapper: Avbryt, slett og lagre. Ved *trykk* på «slett» knappen vises en varslingsboks som ber om bekreftelse på handlingen du skal utføre (se figur 28). Dersom «OK»-knappen *trykkes*, blir alt innholdet for gjeldende side slettet og du får et varsel når handlingen er utført (se figur 33). Ved *trykk* på «avbryt» knappen får du også varslingsboks hvor du må bekrefte handlingen du skal utføre (se figur 29). Hvis du *trykker* «OK»-knappen blir du navigert til dashboard (se kap. 3.1.2). Ved *trykk* på «lagre» knappen, blir dine endringer lagret. Du får et varsel om at handlingen er utført (se figur 34 under kap. 3.2.4.1). Deretter blir du navigert tilbake til dashboard.

Figur 28 - Slett varslingsboks



Figur 29 - Avbryt varslingsboks

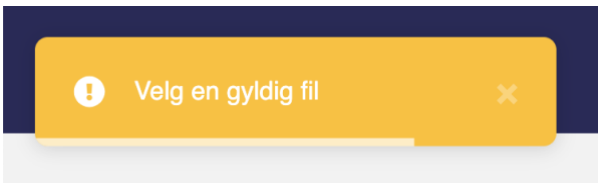
Dersom innholdet ikke er tilgjengelig for endringer, vil du få beskjed gjennom en feil-skjerm (se figur 30).



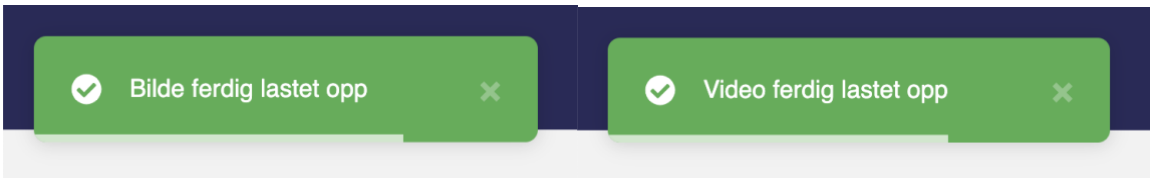
Figur 30 - Feil-skjerm i CMS

3.2.4.1 Varslinger i CMS

Publiseringssystemet (CMS) kan gi deg en rekke tilbakemeldinger i form av varslinger (se figur 31 til 34). Varslingene dukker opp i venstre hjørnet av applikasjonen og har varierende varighet avhengig av lengden på teksten. Et varsel kan pauses ved å holde musepekeren over, eller lukkes før tiden har gått ut ved å *trykke* på krysset til høyre eller sveipe den til høyre eller venstre.

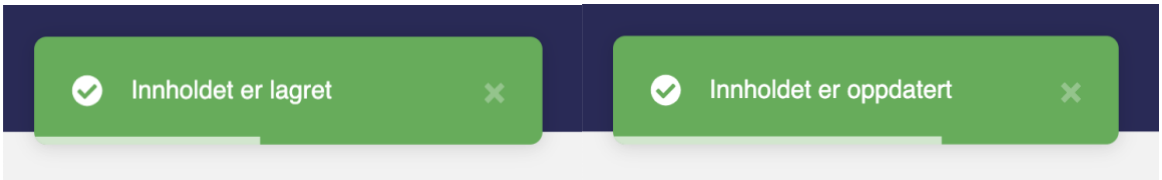


Figur 31 - Advarsel om ugyldig fil



Figur 32 - Vellykket opplasting av bilde/video

Figur 33 - Vellykket sletting av innholdet



Figur 34 - Vellykket lagring/oppdatering