

OSLOMET

Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR

2024-15-47

TILGJENGELIGHET

Åpen

BACHELORPROSJEKT

Telefon: 22 45 32 00

HOVEDPROSJEKTETS TITTEL Brannhjelp	Date 24.05.2024
	ANTALL SIDER / BILAG 81/5
OPPDRAKGIVER Ignist AS	KONTAKTPERSON Joakim Folkesson
INTERN VEILEDER	Michael Tarlton
PROSJEKTDELTAKERE	
Hamid Hamrah	Studentnr: s362052
Hanne Larsen Fjeldaas	Studentnr: s320960
Karine Sørhus Johannessen	Studentnr: s364726
Zulfeqar Shirzadeh	Studentnr: s351922
Johan Sereba	Studentnr: s336180
SAMMENDRAG	
Ignist ser et behov for en plattform hvor de kan lagre informasjon og kundene deres kan hente ut og søke etter denne informasjonen, et slags bibliotek. De ønsket en webapplikasjon med søkefunksjon for å fylle denne rollen. På sikt skal det være mulig å integrere betalingsløsning, slik at kunder må betale for å få tilgang. Løsningen hadde de gitt navnet Brannhjelp. Det skulle også være mulig å hente informasjon fra Brannhjelp til deres eksisterende programvare, STRGI, via API.	
Vi har utviklet en webapplikasjon for Ignist, med mulighet for å logge inn som administrator eller bruker. Det har også blitt satt opp en løsning for brukere som har glemt sitt passord til webapplikasjonen. Administrator-rollen kan opprette, lese, redigere og slette artikler og underartikler, samt drive brukeradministrering. Bruker kan søke på og lese artikler og underartikler. UX-designet har blitt utviklet i iterasjoner i tett samarbeid med veileder fra Ignist, med fokus på brukervennlighet. Det har blitt utført testing av koden og retting av feil funnet i sammenheng med dette.	

3 STIKKORD	Webapplikasjon
	Informasjonsbibliotek
	Systemutvikling

Antall ord: 17813

Innleveringsfrist: 24.05.2024

OSLO METROPOLITAN UNIVERSITY
STORBYUNIVERSITETET

Forord

Prosjektrapporten dokumenterer arbeidet med bachelorprosjektet vi har utført for Ignist. I rapporten beskrives studentenes arbeid med å løse oppdraget fra Ignist, samt selve sluttproduktet som studentene har utviklet.

Studentgruppen kom i kontakt med oppdragsgiver ved at Ignist presenterte et prosjektforslag til OsloMet som var tilgjengelig for studenter å søke på som bachelorprosjekt. Dette prosjektet var allerede tatt av en annen gruppe da studentene tok kontakt med Ignist, men de hadde et forslag til et annet prosjekt som passet for gruppen.

Hensikten bak prosjektet var å utarbeide en webapplikasjon. Ignist ønsket en brukervennlig webapplikasjon som kan benyttes av deres kunder som et kunnskapsbibliotek. Ignist ønsker å publisere veiledere på webapplikasjonen, som kan leses av deres kunder. I prosjektarbeidet har studentgruppen jobbet med å utvikle denne webapplikasjonen, kalt "Brannhjelp". Arbeidet har bestått av utvikling av frontend og backend, prototyping, prosjektarbeid, testing med mer.

Vi ønsker å takke Ignist for å ha gitt oss muligheten til å arbeide for dem. Prosjektet har vært en unik mulighet for oss til å benytte og utvikle våre kunnskaper innen prosjektarbeid og utvikling. Ignist og ekstern veileder, Joakim Folkesson, har gitt nytte tilbakemeldinger som har veiledet studentene gjennom hele prosjektet.

Videre vil vi takke vår interne veileder fra OsloMet, Michael Tarlton. Han har bidratt i strukturering av prosjektet, i tillegg til omfattende hjelp i form av korrekturlesing og tilbakemelding til prosjektrapporten. Dette har vært en viktig forutsetning for at prosjektet ble godt strukturert og dokumentert.

Innhold

1 Innledning	6
1.1 Prosjektgruppen	6
1.2 Oppdragsgiver	7
1.3 Veiledere	7
2 Planlegging og metode	8
2.1 Planlegging	8
2.2 Verktøy	9
2.3 Rammeverk og biblioteker	10
2.4 Arbeidsmetode	12
2.4.1 Smidig og plandrevet utvikling	12
2.4.2 Tilbakemeldinger fra oppdragsgiver og veileder	13
3 Design	14
3.1 Designprosessen	14
3.1.1 Inspirasjon	14
3.1.2 Iterasjoner	14
3.2 Universell utforming	15
3.3 Farge	16
3.4 Ikoner	17
3.5 Font	17
3.6 UX-designet vs kode i løsningen	18
4 Utviklingsprosessen	19
4.1 Kravspesifikasjon	19
4.1.1 Utgangspunkt	19
4.1.2 Funksjonelle krav	19
4.1.3 Ikke-funksjonelle krav	20
4.1.4 Endringer i kravspesifikasjonen	20
4.2 Utforskning av rammeverk	20
4.2.1 Rammeverk for backend	20
4.2.2 Rammeverk for frontend	21
4.3 Kvalitetssikring	23
4.3.1 Rutiner for Git	23
4.4 Databasemodellering	23
5 Testing	25
5.1 Enhetstest	25
5.1.1 Gjennomføring av enhetstester	25
5.1.2 Resultater fra enhetstester	27
5.2 Systemtest	28
5.2.1 Gjennomføring av systemtest	28
5.2.2 Resultater fra testrunde 1	29
5.2.3 Resultater fra testrunde 2	31
5.3 Tilgjengelighetstest	32
5.4 Akseptansestest	32
5.5 Samsvar med testplanen	32
5.5.1 Manglende testdekning	33
5.5.2 Ikke utført integrasjonstest	33
5.5.3 Automatisering av systemtester	33
5.6 Innsikt etter endt testing	33
6 Produktdokumentasjon	35
6.1 Introduksjon av løsningen	35
6.2 Løsningens arkitektur	35

6.3	Frontend	36
6.3.1	Single Page Application	36
6.3.2	Strukturering av frontend-kode	36
6.4	Backend	39
6.4.1	Kontrollere(<i>Controllers</i>)	39
6.4.2	Oppbevaringssteder(<i.repositories< i="">)</i.repositories<>	41
6.4.3	Modeller(<i>Models</i>)	42
6.4.4	Tjenester(<i>Services</i>)	44
6.4.5	Loggføring (<i>Logs</i>)	45
6.4.6	Verktøy(<i>Utilities</i>)	46
6.5	Sentrale datastrukturer	47
6.6	Hoveddeler i løsningen	47
6.7	Samsvar mellom produkt og kravspesifikasjon	48
7	Brukerveiledning	51
7.1	Landingsside	51
7.2	Innloggingsside	51
7.3	Registreringsside	52
7.4	Glemt passord-side	53
7.5	Administrators muligheter	53
7.5.1	Opprette artikler og underartikler	53
7.5.2	Redigere og slette	54
8	Konklusjon	56
8.1	Læringsutbytte	56
8.2	Brannhjelps bruks- og nytteverdi	56
8.3	Veien videre	57
8.4	Oppsummering og konklusjon	57
A	Iterasjoner av UX-design	62
B	Testplan	69
C	Brukertilhistorier til systemtest	72
D	Test cases og resultater fra systemtester	73
E	Attest fra oppdragsgiver	81

Figurer

1	Fremdriftsplan og Gantt-skjema	8
2	En av de første iterasjonene laget i Figma. Denne i blåtoner.	15
3	Femte og siste iterasjon laget i Figma	16
4	Fargepalett valgt av Ignist.	17
5	Landsingsside modell laget i Figma.	17
6	Det mest brukte rammeverket (Stack Overflow, 2023)	22
7	Klassediagram	24
8	Testobjekter for enhetstester	26
9	Testmetode for å teste gyldig registrering av ny bruker	27
10	Kjøring av alle enhetstester	27
11	Test cases	28
12	Feil 5 - knapp (pilen) som ikke utfører noen handling	30
13	Alert som ber om bekreftelse før sletting	30
14	Bekreftende melding om at artikkel er opprettet	30
15	Ikke alle publikasjoner er synlige på siden	31
16	Alle publikasjoner vises dersom man zoomer ut på siden	31
17	Arkitekturen til frontend-koden	36
18	Authentication-mappen	37
19	Endringsformidling i et komponenttre uten og med kontekst (Sharmaa, 2023)	37
20	Componenets-mappen	38
21	Pages-mappen	39
22	Kommunikasjon mellom frontend- og backend-komponenten	40
23	Kontroller-mappen som definerer API-endepunktene	40
24	Repository-klassene som forenkler spørninger mellom controllere og databasen	41
25	AuthenticationRepository styrer databaseoperasjoner for AuthController i Cosmos DB	42
26	Kodeoversikt over PublicationsRepository, som styrer databaseoperasjoner for publikasjoner i Cosmos DB	42
27	Oversikt over 'Models'-mappen som viser datastrukturer brukt i backend	43
28	'User.cs'-klassen definerer brukerens datastruktur	43
29	Attributtene som finnes i 'Publication'-klassen	44
30	Oversikt over 'Services'-mappen som viser IcosmosDbService-grensesnittet	45
31	'Utilities'-mappen som inneholder hjelpetjenester og funksjonaliteter for systemet	46
32	Oversikt over oppnåelse av funksjonelle krav	48
33	Oversikt over oppnåelse av ikke-funksjonelle krav	49
34	Hjemmeside	51
35	Innloggingsside	52
36	Registreringsside	52
37	Side for å tilbakestille passord	53
38	Side for å opprette ny artikkel	54
39	Opprettelse av underkategori	54
40	Side for administrering av artikler	55
41	Første iterasjon av UX-design laget i Figma, i oransj fargepalett	62
42	Første iterasjon av UX-designet laget i Figma, med andre detaljer i oransj	63
43	Første iterasjon av UX-designet laget i Figma, med andre detaljer i blått	64
44	Andre iterasjon av UX-designet laget i Figma	65
45	Tredje iterasjon av UX-designet laget i Figma	66
46	Fjerde iterasjon av UX-designet laget i Figma	67
47	Femte iterasjon av UX-designet laget i Figma	68

1 Innledning

I dette kapittelet presenteres prosjektgruppens medlemmer, samt oppdragsgiver og prosjektgruppens to veiledere. Prosjektgruppen har blitt bistått av en intern veileder fra OsloMet og en veileder fra oppdragsgiver.

1.1 Prosjektgruppen

Prosjektgruppen består av fem studenter ved OsloMet. Tre av studentene går dataingeniør-linjen og to studenter går på linjen anvendt datatekologi. Kombinasjonen av to ulike studieretninger har gjort at prosjektgruppen er en godt sammensatt gruppe med erfaring og kunnskap innen ulike felt. For eksempel har noen av gruppe-medlemmene god kunnskap i utvikling, inkludert frontend og backend, mens noen har styrke innen UX design, testing og prosjektarbeid.

Hamid Hamrah

Bachelorprogram:
Dataingeniør



Hovedansvar for
koordinering,
arkitektur og
frontend

Hanne Larsen Fjeldaas

Bachelorprogram:
Anvendt Datateknologi



Hovedansvar for
prosjektrapport
testing og
kravspesifikasjon

Zulfeqar Shirzadeh

Bachelorprogram:
Dataingeniør



Hovedansvar for
backend og
database integration

Karine Sørhus Johannessen

Bachelorprogram:
Anvendt Datateknologi



Hovedansvar for
prototyping og
prosjektrapport

Johan Sereba

Bachelorprogram:
Dataingeniør



Hovedansvar for
prosjektrapport
poster ansvarlig

1.2 Oppdragsgiver

Ignist er et privat selskap som ble startet opp av Joakim Folkesson og Morten Iversen Berland i 2021. Ignist er et frittstående rådgivningsselskap som tilbyr brannteknisk spesialkompetanse til bygge- og offshore-bransjen. Tjenester som tilbys av Ignist er blant annet brannteknisk prosjektering, uavhengige kontroller, tilstandsvurderinger og due diligence. (Ignist AS, u.å.).

1.3 Veiledere

Prosjektgruppen har hatt to veiledere, en intern veileder fra OsloMet, Michael Tarlton, og en veileder fra Ignist, Joakim Folkesson.

Studentene ble tildelt Michael Tarlton som veileder av instituttet, og han har veiledet studentgruppen gjennom hele prosjektet. Det ble tidlig satt opp ukentlige møter hvor studentene fortalte om fremdrift siden forrige møte, mål frem til neste møte, samt mulighet for å stille spørsmål til Michael ved behov. Michael har spesielt bidratt til strukturering av prosjektarbeidet, ved å gi studentene en pekepinn på hva som burde jobbes med og fokuseres på videre. Michael har også gitt tydelige oppfordringer til å kontinuerlig dokumentere prosjektarbeidet underveis, og starte tidlig med skriving av sluttrapporten. Dette har vært et viktig bidrag som har sikret at vi har kunnet utarbeide sluttrapporten på en effektiv og grundig måte. Sluttrapporten har hele tiden vært et fokus i bachelorprosjektet, og har vært jobbet med kontinuerlig fra prosjektets oppstart.

Joakim Folkesson fra Ignist har også veiledet studentene regelmessig gjennom hele prosjektet. Før oppstart av prosjektet og signering av kontrakten, presenterte Joakim prosjektet og hva Ignist ønsket at studentene skulle levere. Dette ga studentene en grunnleggende forståelse av hva som skulle utvikles og om dette var et egn prosjekt for studentene. I startfasen av prosjektet, fikk studentene en grundigere forklaring av hva Ignist ønsker å oppnå med prosjektet, hvilke krav og mål de har til løsningen, og hvilke behov Ignist ønsker å oppfylle av prosjektet. Dette ga grunnlaget til utforming av kravspesifikasjon. Joakim har også bidratt når studentene har møtt på utfordringer og skalert prosjektet etter faktiske forutsetninger, for eksempel bestemte Ignist at integrasjon av betalingsløsning ikke var oppnåelig innen prosjektets rammer og dette kravet ble bortprioritert for annen funksjonalitet.

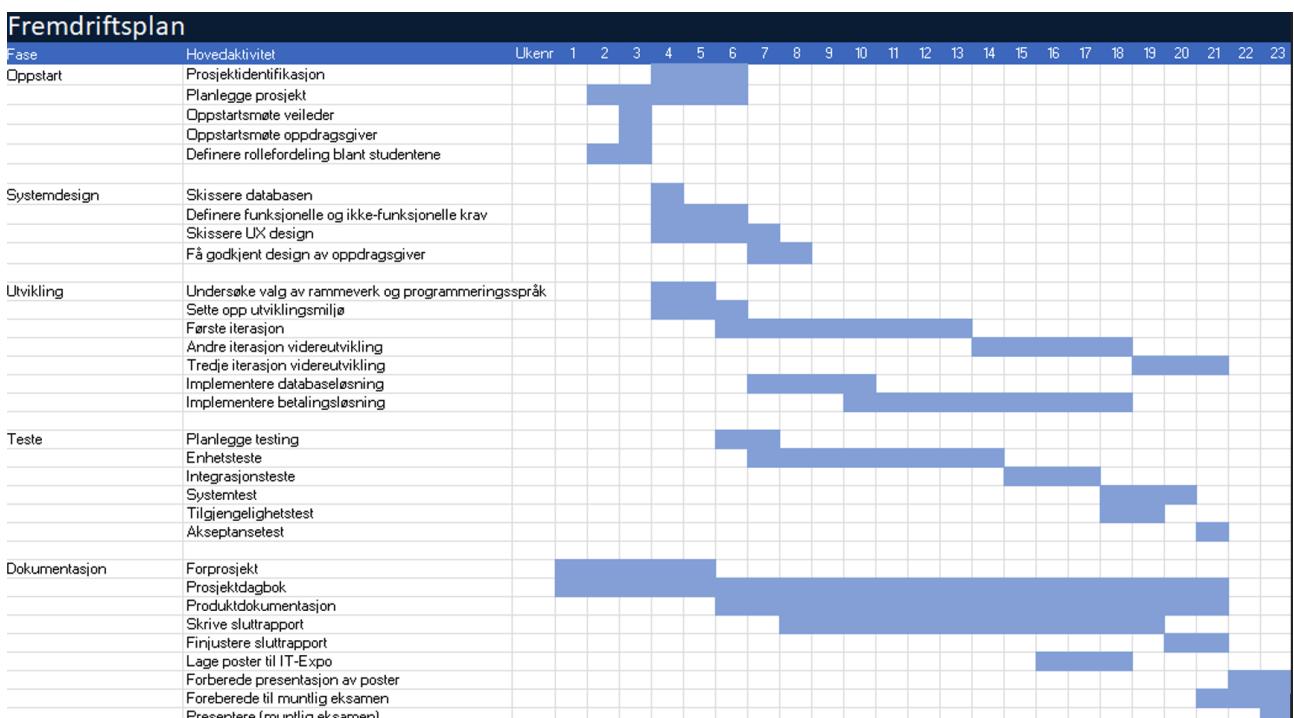
Ignist kommuniserte tidlig i prosjektet at det var viktig at webapplikasjonen er brukervennlig, ettersom de ønsker å selge tilgang til webapplikasjonen i fremtiden. I forbindelse med dette, har Joakim bidratt med å gi studentene flere eksempler på hvordan de ønsket at systemet skulle se ut. Studentene har presentert forslag til design, og Joakim har vært behjelplig med konstruktive tilbakemeldinger for å heve kvaliteten på systemet. På denne måten har Joakim skapt bedre forutsetninger for at oppdragsgiver blir fornøyde med sluttproduktet og for å sikre at studentene har vært på rett spor underveis i utviklingsprosjektet.

2 Planlegging og metode

I dette kapittelet beskrives hvordan studentene planla prosjektarbeidet og hvordan planlegging fungerte underveis i prosjektet. Deretter listes en oversikt over verktøy, rammeverk og biblioteker som er benyttet i prosjektet. Til slutt drøftes valg av arbeidsmetode og en beskrivelse av hvordan studentgruppen arbeidet og fikk tilbakemeldinger fra veiledere.

2.1 Planlegging

I forbindelse med innlevering av forprosjektrapport, utformet studentene en overordnet plan over prosjektet. I denne rapporten lagde studentene blant annet en fremdriftsplan i form av et Gantt-skjema, se figur 1, som viste en tidsplan over de ulike fasene og aktivitetene i prosjektet. Studentene planla å dele utviklingen inn i iterasjoner og utføre testing parallelt med utviklingen. Denne arbeidsmetoden var ønsket for å sikre at hver enkelt student alltid hadde noe å jobbe med, slik at studentgruppen fikk maksimal utnyttelse av den enkeltes ressurser. Dokumentering av prosjektet og utforming av prosjektrapport var også planlagt å skje underveis i hele prosjektet, for å sikre at prosjektet ble tilstrekkelig dokumentert i sanntid.



Figur 1: Fremdriftsplan og Gantt-skjema

I og med at Gantt-skjemaet ble utarbeidet før selve prosjektarbeidet sattes i gang, var gruppen enige om at dette ble en pekepinn på tidsrammen og at det ville forekomme avvik fra tidsplanen. Gantt-skjemaet var allikevel en viktig oversikt som bidro til å sikre en kontinuerlig fremdrift i prosjektet, og studentene forsøkte å overholde tidsplanen så godt som mulig underveis i prosjektet. Noen aktiviteter tok lenger tid enn forespeilet, som for eksempel utfordringer med databasemodellering, som blir beskrevet i delkapittel 4.4.

Som planlegging underveis i prosjektet, gjennomførte studentgruppen ukentlige stand-up møter hvor studentene ble enige om mål og planer for den kommende uken. På neste stand-up gjennomgikk studentene hvorvidt disse målene var nådd eller ikke. Hver uke ble det også holdt et stand-up møte med intern veileder fra OsloMet og et

med veileder fra Ignist. Disse møtene og kontinuerlig målsetting og sammenligning av progresjon opp mot målene, har vært nyttige for å få en tydelig formening om hva hver enkelt student skal jobbe med videre, og hvordan gruppen ligger an i prosjektarbeidet. Disse møtene har også vært en naturlig arena for å lufte utfordringer og problemer som har dukket opp fortløpende, både studentene seg imellom, men også med oppdragsgiver og veileder fra OsloMet. På denne måten har studentene kunnet håndtere problemer så fort som mulig, og dermed bidratt til å minimere risiko for at disse gir ringvirkninger i andre faser av prosjektarbeidet. Takhøyde for å diskutere utfordringer hyppig har også gitt verdifull input fra andre gruppemedlemmer og veiledere.

2.2 Verktøy

Dette kapitelet presenterer verktøyene som har blitt anvendt for å løse oppgaven ved en generell forklaring av verktøyet, med fokus på aspektene ved verktøyet som er tatt i bruk for å løse oppgaven.

Discord

Discord er en populær kommunikasjonsplattform som i utgangspunktet ble laget for folk som spiller videofspill. Discord er ikke et spill i seg selv, men brukes i stor grad for å kommunisere via tekst eller lydsamtaler mens en spiller. Plattformen tilbyr også muligheten for å sette opp servere med kanaler for stemmekommunikasjon eller tekstkommunikasjon. Discord brukes i dag av forskjellige sammensetninger av grupper, f.eks. utviklere eller vennegrupper (Medietilsynet, 2022). Vi har brukt Discord for enkel uformell kommunikasjon og deling av filer i forbindelse med prosjektet.

Figma

Figma er et skybasert design- og prototypingverktøy for digitale prosjekter. Det gjør det mulig å samarbeide på tvers av lokasjoner, slik at man kan arbeide sammen uansett hvor man befinner seg. Figma har likheter med andre designverktøy, men har den unike fordelen at det støtter teamarbeid. (Figma, u.å.). Vi har benyttet Figma til å utforme prototyper av UX-designet. Tidligere erfaring med Figma og muligheten til å enkelt dele og arbeide på filer sammen gjorde at vi valgte Figma som arbeidsverktøy.

Git

Git er et versjonshåndteringssystem som opererer lokalt på brukerens datamaskin og holder oversikt over endringer i filer, som f.eks. kildekode. Det er et distribuert versjonshåndteringssystem, noe som innebærer at hver bidragsyter beholder en personlig kopi av hele koden og dens endringshistorikk (Kinsta, 2023a). Dette gjør det enkelt å håndtere endringer fra flere brukere. Når en bruker ønsker å dele sine endringer, publiseres de til et sentralt kodearkiv, som for eksempel GitHub. Git er spesielt egnet for å følge med på historikken til tekstbaserte filer, som programmeringskoder, hvor det er nyttig å sammenligne fileversjoner over tid.

GitHub

GitHub er et kommersielt foretak som tilbyr lagring av Git-repositories i skyen. Denne tjenesten forenkler bruken av Git for versjonskontroll og samarbeid i prosjekter (Kinsta, 2023b). Ved bruk av Github har vi dokumentert nøkkelinformasjon om endringer for hvert endringssett, eller commit, som ble gjort. Slik har vi også sikret at ingenting ble overført til det delte repositoriet dersom koden ikke kjørte slik som tiltenkt.

Microsoft OneDrive

OneDrive er en Microsoft-skytjeneste som kan brukes til lagring av filer. Verktøyet sikrer filene og gir mulighet til å dele filer med andre brukere. OneDrive kan også integreres med Microsoft Word, slik at flere personer kan arbeide med det samme dokumentet samtidig (Microsoft, u.å.-a). Gjennom prosjektet har vi jevnlig brukt OneDrive for å holde vår dokumentasjon oppdatert og tilgjengelig for alle studentene som deltok i prosjektet.

Visual Studio

Visual Studio er et integrert utviklingsmiljø (IDE) utviklet av Microsoft, som tilbyr omfattende verktøy og funksjoner for programvareutvikling, feilsøking og versjonskontroll. Det støtter en rekke programmeringsspråk og rammeverk, for eksempel C#, .NET, JavaScript, Python og mange flere, som gjør det til et fleksibelt verktøy for utviklere som arbeider på tvers av forskjellige teknologistakker (Microsoft, u.å.-c).

I prosjektet har Visual Studio spilt en sentral rolle både i utviklingen av backend- og frontendkoden. For backend, som er skrevet i C# og benytter .NET 8.0, har Visual Studio gitt oss ett miljø for å utvikle, teste og feilsøke vår ASP.NET Core Web API. Vi har også benyttet Visual Studios støtte for Azure integrasjon til å koble til Azure Cosmos DB, som har vært kritisk for å kunne lagre og håndtere data for applikasjonen. Visual Studios debugging-verktøy har også gjort det enklere å identifisere og løse problemer raskt, som har bidratt til å opprettholde kodekvalitet.

For utvikling av frontend, som hovedsakelig er basert på React, har Visual Studio med utvidelser, som Visual Studio Code, gitt en effektiv utviklingsprosess med støtte for moderne webteknologier og rammeverk. Verktøyene for kildekontroll som er integrert i Visual Studio har gjort det mulig for teamet vårt å samarbeide effektivt, med enkel tilgang til versjonskontroll-funksjoner som “Branching” og “Merging”, noe som har vært viktig for å håndtere ulike funksjoner og endringer i prosjektet.

Slack

Slack er en kommunikasjonsplattform designet for bedrifter, som legger til rette for et mer samarbeidsorientert arbeidsmiljø ved å tilby et sammenkoblet, fleksibel og inkluderende arbeidsmiljø. Plattformen muliggjør effektiv kommunikasjon og samarbeid gjennom organisering av dialog i kanaler. Plattformen støtter asynkront arbeid som gjør informasjon tilgjengelig på tvers av ulike tidssoner og lokasjoner, slik at team medlemmer har tilgang til den samme informasjonen for raskere beslutninger (Slack, u.å.). Vi har brukt Slack for mer uformell kommunikasjon med Ignist utenom de ukentlige møtene.

Microsoft Word

Microsoft Word er et tekstbehandlingsprogram som tilbyr funksjonalitet for å opprette, redigere, og formtere tekstdokumenter. Word er utviklet for både individuelt og kollektivt arbeid, og støtter integrasjon med OneDrive for lagring og deling, som gjør det mulig for samhandling og deling i et team (Microsoft, u.å.-d). Vi har anvendt Word for å dokumentere alt arbeid under prosjektet. Gjennom integrasjon med OneDrive har vi lagret og delt dokumentasjonen med teamet, noe som har forsterket vårt samarbeid og produktivitet.

Microsoft Teams

Microsoft Teams er et skybasert samarbeidsverktøy fra Microsoft 365 og Office 365, designet for å kommunisere via meldinger, videomøter og fildeling (Robinson & O'Neill, 2024). Vi har benyttet Teams som møteplattform med oppdragsgiver og intern veileder for de ukentlige møtene. Ved å bruke Teams har vi enkelt kunne holde møter, dele skjermer og filer underveis for å holde hverandre og våre veiledere informert om status på prosjektet.

2.3 Rammeverk og biblioteker

Rammeverk og biblioteker spiller en viktig rolle i utviklingsprosessen av applikasjoner. En rekke rammeverk og biblioteker er mye brukt i programvareutviklingsbransjen. I dette avsnittet presenteres rammeverkene og bibliotekene som har blitt tatt i bruk for å løse denne oppgaven. Beskrivelsene er ment som en introduksjon. Senere i rapporten vil vi komme med dypere beskrivelse og detaljer om utvalgsprosessen av de ulike rammeverkene/bibliotekene som har blitt anvendt, og hvordan disse har blitt anvendt.

ASP.NET Core

ASP.NET Core er et rammeverk introdusert av Microsoft i 2016, som har oppnådd bred anerkjennelse for sin

tilgjengelighet som åpen kildekode under en Microsoft-lisens. Rammeverket åpner for integrert bruk av Microsofts omfattende verktøy og økosystemer, og er spesielt tilrettelagt for effektiv utvikling av webapplikasjoner og skrivebordsprogrammer. Det er designet for å gi utviklere muligheten til å skrive kode i C# og C++, og fremmer plattformuavhengig utvikling, noe som tillater programmer å kjøre på forskjellige operativsystemer (Microsoft, u.å.-b).

React

React er et JavaScript-bibliotek utviklet for å bygge dynamiske og interaktive brukergrensesnitt. Biblioteket gir mulighet for komponentbasert utvikling, noe som forenkler prosessen med å utforme visuelle elementer. I stedet for å manipulere Document Object Modelen (DOM) direkte, håndterer React denne prosessen virtuelt, noe som gir utviklere muligheten til å skape brukergrensesnitt på en mer intuitiv og effektiv måte. Det konseptuelle rammeverket til React tilbyr en enklere tilnærming til utvikling sammenlignet med metoder som manipulerer DOM-en. Dette tilrettelegger ikke bare for en bedre forståelse under utviklingsprosessen, men sikrer også høy ytelse. Biblioteket er utviklet av Meta Platforms.

TypeScript

TypeScript er et programmeringsspråk utviklet av Microsoft. Språket er designet for å utvide JavaScript ved å introdusere statiske typer. Dette legger til et lag av sikkerhet ved utvikling av applikasjoner, ved å tillate definering av Typer og Interfaces (TypeScript, u.å.). I vårt prosjekt har bruken av typescript bidratt til å forbedre kodekvaliteten og forenkle refaktorering og vedlikehold. Dette har også effektivisert utviklingsprosessen.

Swagger

Swagger er et åpent kildekode-verktøysett designet for å hjelpe utviklere å bygge, dokumentere og bruke REST APIer mer effektivt. Swagger baserer seg på OpenAPI, og tilbyr en rekke verktøy som f.eks. Swagger UI for interaktiv dokumentasjon, Swagger Editor for redigering av API-spesifikasjoner, og Swagger Code-gen for å generere server- og klientkode. (Swagger, u.å.). I prosjektet benyttes Swagger for å automatisk generere dokumentasjon for APIet, basert på koden. Gjennom Swagger UI får utviklere utforske og testet APIet direkte fra nettleseren, som forenkler forståelsen og bruken av APIet.

Material UI

Material UI er et åpent kildekode-bibliotek for React som tilbyr forhåndsbygde komponenter basert på Google Material Design. Det gjør det enkelt for utviklere å skape estetiske og funksjonelle webapplikasjoner, med muligheter for egne tilpasninger (Material UI, u.å.).

Gjennom bruk av komponenter som Container, Box, Textfield og Button, har vi oppnådd et strukturert layout og effektiv interaktivitet i brukergrensesnittet. Snackbar og Alert komponentene er benyttet for å gi brukervennlige tilbakemeldinger. Navigasjonen er forbedret med bruk av IconButton, Menu og MenuItem som gir en responsiv meny. I tillegg har vi brukt Typography for å oppnå konsistent og tilgjengelig typografi.

Froala

Froala editor er en høyytelse, lettvekt HTML-editor også kalt en WYSIWYG (What you see is what you get) Editor. Denne er designet for å levere en god redigeringsopplevelse for brukere og enkel integrasjon for utviklere. Froala har et intuitivt, rent design som er enkelt å bruke samtidig som det gir et bredt utvalg av redigeringsfunksjoner og tilpasningsalternativer (Froala, u.å.).

Vi har brukt React-Froala-WYSIWYG-pakken for å integrere Froala Editor i applikasjonen. Denne er brukt i komponentene Create og Update i webapplikasjonen, for å kunne opprette og oppdatere innhold. Dette tillater brukere å benytte seg av Froalas mange tekstdredigeringsfunksjoner når man arbeider med publikasjoner. Vi har inkludert plugins og verktøyalternativer for å støtte funksjoner som tekstformatering, innsetting av lenker og

mer.

2.4 Arbeidsmetode

Systemutvikling kan utføres med ulike arbeidsmetoder, henholdsvis smidig eller plandreven utviklingsprosess. Hvordan man velger å jobbe med systemutvikling, påvirker resultatet og bør avhenge av prosjektets og programvarens egenskaper og krav (Sommerville, 2016, s. 46). For å bestemme arbeidsmetode for prosjektet, ble egenskapene ved prosjektet og programvaren som skulle utvikles drøftet. Spesielt prosjektgruppens kompetanse, kommunikasjonsbehov med oppdragsgiver og prosjektets omfang og tidsfrist ble tatt i betraktning ved valg av arbeidsmetode.

Studentene ønsket å ha en overordnet prosjekt- og fremdriftsplan, men med mulighet for avvik i denne, slik at man kan tilpasse seg endringer underveis i prosjektet. På grunn av tidsfristen til prosjektet, så studentene behov for å overlappe arbeid med ulike aktiviteter underveis. Gruppemedlemmene har ulike ferdigheter og erfaringer, og så det derfor som hensiktsmessig at noen utførte tester samtidig som andre jobbet med utvikling. I tillegg skulle det arbeides parallelt med utvikling av UX-design og backend, for å spare tid og utnytte gruppens ressurser best mulig.

Det ble også bestemt at vi ønsket tett kontakt med oppdragsgiver i form av ukentlige møter. Krav og mål for webapplikasjonen ble tidlig kommunisert av oppdragsgiver, men ikke formalisert som et selvstendig dokument før en måneds tid inn i utviklingen av løsningen. Studentene var åpne for at det kunne skje endringer i kravspesifikasjonen underveis, men hadde en forventning om at dette eventuelt ville være mindre endringer, og ikke større endringer i funksjonalitet.

Studentene ønsker å levere en minimumsversjon av webapplikasjonen først, med fokus på å levere den viktigste funksjonaliteten tidligst via iterasjoner. I utgangspunktet ønsket Ignist at det skulle være betalingsløsning slik at deres kunder kunne betale for tilgang til webapplikasjonen. Underveis i prosjektet ble Ignist og studentene heller enige om at studentene i første omgang skulle utvikle en webapplikasjon med funksjonalitet for å opprette, lese, oppdatere og slette publikasjoner, kalt "CRUD", og at Ignist sine kunder kan lese uten å betale. Dette var nyttig slik at Ignist kunne ta i bruk webapplikasjonen ved prosjektets slutt, da det ble forespeilet at det mest sannsynlig ikke var nok tid til å implementere funksjonalitet for betaling innen prosjektets slutt.

Valg av arbeidsmetode falt på å inkorporere teknikker og metoder fra både smidig- og plandreven utvikling, men med klart mest metoder hentet fra smidig metodikk.

2.4.1 Smidig og plandreven utvikling

Gruppen har som nevnt brukt en blanding av smidig utvikling og plandreven utvikling som arbeidsmetode under prosjektet

Smidig utvikling, eller agile metoder, er en måte å strukturere et prosjekt på hvor man tar i bruk satte perioder med tid for å jobbe med delmål for prosjektet. Disse satte periodene kan også kalles sprinter og varer ofte i to til fire uker. Smidig utvikling har fem punkter som til sammen utgjør fokusområdene til arbeidsmetoden(e), disse omhandler: kundeinvolvering, fokus på enkelhet, fokus på folk – ikke prosesser, omfavne forandring, og inkrementell levering (Sommerville, 2016, s. 76). Med disse fokusområdene er smidig utvikling agilt og åpent for endringer under hele arbeidsprosessen. Smidig utvikling passer godt som arbeidsmetode når man jobber med små systemer som kan leveres raskt til kunder og for utvikling av kode i iterasjoner (Sommerville, 2016, s. 75). Med tanke på tidsrammen til prosjektet og kravene satt av oppdragsgiver, var det naturlig å adoptere flere aspekter av smidig utvikling for prosjektarbeidet.

Plandrevne prosesser har en klar plan på forhånd av arbeidsstart på et prosjekt og arbeidet måles etter hvorvidt målene på denne planen er nådd. Som nevnt ble et Gannt-skjema satt opp i planleggingsfasen for dette prosjektet, hvor det var satt mål for progresjonen. Målene ble satt basert på kravene fra oppdragsgiver. Ut ifra dette Gannt-skjemaet ble det naturlig å jobbe i ukes-sprinter, med de satte ukentlige møtene med oppdragsgiver som oppsummering av sprintens mål. For å komme frem til et resultat med en funksjonell webapplikasjon, var det ikke alltid mulig å begynne på en ny sprint med et nytt mål dersom et mål ikke var nådd i forrige sprint. For eksempel tok koden for å legge til token for å bli innlogget mer tid enn antatt, og derfor brukte noen av studentene enda en sprint for å komme i mål med dette.

Studentene satte seg mål fra uke til uke, hvor de ukentlige møtene med oppdragsgiver og internveileder fra OsloMet har blitt brukt som oppsummering på hvilke mål som har blitt oppnådd og hvordan den neste bolken med tid vil bli brukt.

Det har kun vært én overlevering av koden til prosjektet, denne var mot slutten av prosjektperioden. Fremgangen på prosjektet har dog blitt presentert og demonstrert for oppdragsgiver på slutten av hver sprint, som en form for overlevering ved hver sprintslutt. Slik har det blitt sikret at prosjektet ikke ble avsporet fra oppdragsgivers krav til prosjektet, og oppdragsgiver har fått innsikt i fremgangen underveis. Disse ukentlige møtene ble også en naturlig arena for oppdragsgiver å komme med tilbakemeldinger og vurderinger på arbeidet under prosessen.

2.4.2 Tilbakemeldinger fra oppdragsgiver og veileder

Stand-up møtene har vært en naturlig anledning til å få tilbakemeldinger fra oppdragsgiver og intern veileder kontinuerlig gjennom hele prosjektet. Dette har vært tilbakemeldinger både knyttet til selve produktet og til dokumentering og strukturering av prosjektet. Disse møtene var svært nyttige for å sikre at studentene var på riktig spor og at oppdragsgiver fikk sett fremdriften og kommet med tilbakemeldinger.

Utenom møtene har studentene også tatt kontakt med oppdragsgiver og intern veileder ved behov. For eksempel har vi kontaktet oppdragsgiver ved utforming av UX design, hvor en av studentene hadde jevnlig kontakt via Slack og fysiske møter for å gjennomgå og få tilbakemelding på design. Intern veileder var tilgjengelig for å gi tilbakemeldinger knyttet til prosjektrapporten.

3 Design

I dette kapitelet beskrives designprosessen og hvordan arbeidet med dette har foregått. Deretter følger en beskrivelse av designvalgene som har blitt tatt og begrunnelser for disse valgene. Til slutt diskuteres forskjellene mellom det ferdige UX-designet og koden levert til oppdragsgiver ved avslutning av prosjektet.

3.1 Designprosessen

UX står for User Experience, eller bruker opplevelse og beskrives av Benyon som, løst oversatt fra engelsk til norsk, “å lage høy-kvalitets opplevelser for mennesker som bruker interaktive systemer.” (Benyon, 2010, s. 97).

Arbeidet med UX-designet har foregått i hovedsak mellom en student og veileder fra Ignist, med innspill fra de andre studentene og representanter fra Ignist. Kommunikasjon har foregått i hovedsak på de ukentlige møtene, med supplerende kommunikasjon via Slack. Da i stor grad for deling av ressurser og oppklaring av mindre detaljer. I løpet av prosjektprosessen hadde studenten som arbeidet med designet også en arbeidsdag på kontoret til Ignist, for å komme frem til en tilnærmet ferdig versjon av UX-designet. Den siste iterasjonen av Figma modellene ble laget på tilbakemeldinger og notater gjort denne dagen.

I løpet av en sprint ble det laget en versjon av UX-designet som deretter ble presentert på de ukentlige møtene. Her kom Ignist med tilbakemeldinger og deretter ble en ny versjon laget med tilbakemeldingene som mål for den nye versjonen. UX-designet fikk en del fokus da Ignist hadde et sterkt ønske om at dette skulle være intuitivt og funksjonelt å bruke. Dette medførte at mer tid ble allokkert til arbeid med UX-designet enn hva som var tiltenkt i Gannt-skjemaet, og arbeidet med front-end koden ble noe utsatt som resultat av dette.

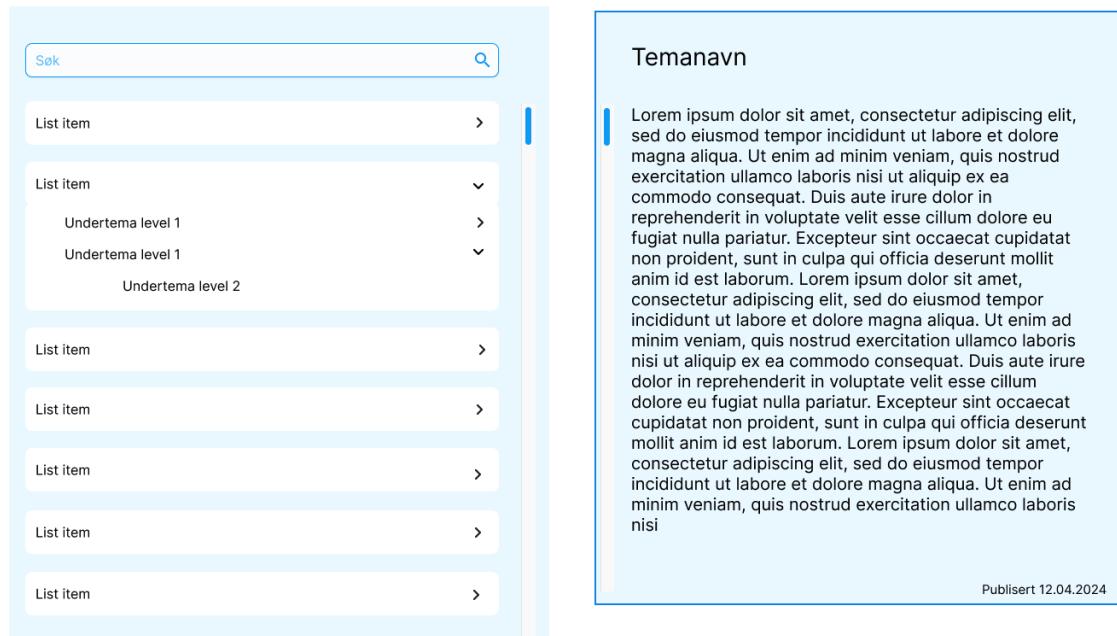
I kravspesifikasjonen var det et krav som omhandlet UX-designet, som lyder “UX-designet og brukergrensesnittet skal være intuitivt og brukervennlig, slik at webapplikasjonen gir en positiv brukeropplevelse”. I tillegg skulle webapplikasjonen bestå av elementene hovedbanner eller topptekst, kategorimeny, artikkel med overskrift og brødtekst, en søkerbar, og et banner med bunntekst.

3.1.1 Inspirasjon

Designprosessen begynte med å hente inspirasjon fra andre webapplikasjoner med noenlunde tilsvarende funksjoner som webapplikasjonen skulle ha. Ignist sendte en liste med eksempler på webapplikasjoner som de likte hele eller deler av designet til og som de fant funksjonelle og intuitive å bruke. Disse eksemplene var kontohjelp.no, airbnb.no og stonly.no. Det var aspekter ved disse eksemplene, som oppsett av elementene, fargebruk og fargevalg, som var viktige å legge merke til. I tillegg hadde Ignist en god formening om hvordan de ønsket oppsettet av temaer og artiklene med tekst.

3.1.2 Iterasjoner

I første omgang ble det laget to versjoner av UX-designet. Disse to versjonene ble også presentert med to forskjellige fargetemaer hver, for å få en god pekepinn på hvilken retning Ignist ønsket å ta med tanke på de visuelle elementene. Se Figur 2 som eksempel på en av disse. Videre fikk vi tilbakemeldinger fra Ignist, og brukte disse til å utforme neste iterasjon av UX-designet. Til sammen ble det laget fem iterasjoner av UX-designet, hvor de tre første versjonene av disse ble presentert med flere fargealternativer. Se vedlegg A for alle iterasjoner. Til den siste iterasjonen ble det også laget en versjon av en landingsside etter Ignist sitt ønske.



The screenshot shows a Figma wireframe of a web application. On the left is a sidebar with a search bar at the top. Below it is a list of items, each with a right-pointing arrow icon. Some items have a downward arrow icon next to them, indicating they are expandable. One item under "List item" has two nested levels of expandable items. The main content area on the right is titled "Temanavn" and contains a large amount of placeholder text (Lorem ipsum). At the bottom right of the content area is the text "Publisert 12.04.2024".

Figur 2: En av de første iterasjonene laget i Figma. Denne i blåtoner.

3.2 Universell utforming

Universell utforming er et viktig aspekt som må tas hensyn til ved UX-design i dag. FN-konvensjonen om rettighetene til personer med nedsatt funksjonsevne definerer universell utforming på følgende måter:

«Med ”universell utforming”menes utforming av produkter, omgivelser, programmer og tjenester på en slik måte at de kan brukes av alle mennesker, i så stor utstrekning som mulig, uten behov for tilpassing og en spesiell utforming. ”Universell utforming” skal ikke utelukke hjelpemidler for bestemte grupper av mennesker med nedsatt funksjonsevne når det er behov for det.» (FN, 2006).

For at så mange som mulig skal kunne bruke et produkt uavhengig av funksjonsnedsettelse eller andre hemninger, om det er en webapplikasjon slik som i dette prosjektet, eller et fysisk produkt som en fjernkontroll, er universell utforming et viktig aspekt ved design av et produkt. Universell utforming er et bredt tema med mange aspekter som vil variere ut ifra produktet man skal utforme. Ved et fysisk produkt vil man f.eks. måtte ta stilling til den fysiske formen til produktet, hvordan skal det holdes og kan flest mulige ha et godt grep ved denne utformingen. Ved et digitalt produkt, som f.eks. en webapplikasjon må man ta stilling til aspekter som fargevalg på tekst og bakgrunn, for god lesbarhet for så mange som mulig, og at bilder er merket med alternativ tekst slik at brukere som er avhengige av skjermleser ikke går glipp av innhold når skjermleseren ikke kan lese f.eks. et bilde.

Kravspesifikasjonen som omhandlet UX-designet hadde som nevnt tidligere fokus på brukervennlighet og intuitivt design. Med dette var det naturlig, og nødvendig, å også ha fokus på universell utforming når designvalg ble tatt. Ettersom hovedbruksområdet til webapplikasjonen vil være å meddele informasjon via tekst til Ignist

sine kunder, har vi lagt fokus spesielt på at innholdet må være lettleselig ved design. Dette reflekteres i valget av font, som vil utdypes i delkapittel 3.5 og i fargevalg som diskuteres i delkapittel 3.3.

Main content

[Materialer og produkters](#) ▾
 [Barn](#) >
 [Overflatekrav & brann](#) ▾
 [Overflatekrav](#)
 [Kledningskrav](#)
 [Barnebarn](#)
 [Tekniske installasjoner](#) >
 [Bæreevne og stabilitet](#) >
 [Sikkerhet ved eksplosjon](#) >
 [Generelle krav til røminn](#) >
 [Tiltak for å påvirke røminn](#) >
 [Forelder](#) >
 [Forelder](#) >
 [Forelder](#) >
 [Forelder](#) >

Overflatekrav og Brannbeskyttende kledning +
Publisert 12.04.2024
Overflatekrav for vegg og tak. Ved prøving observeres tid til overtenning, varmeavgivelse, røykproduksjon, brannutbredelse og brennende dråper eller deler. Minste tid til overtenning er 10 minutter for klasse D, 12 minutter for klasse C og 20 minutter for klass A og B. Overflaten kan være brennbar [D], men skal ikke produsere brennende dråper [d0] og avgive begrenset mengde røyk [s2].
Med kledning menes en byggevarer som benyttes innvendig eller utvendig på en vegg eller på undersiden av en etasjeskiller. Kledningsklassen angir kledningens evne til å beskytte sin egen bakside og bakenforliggende materiale mot antennelse. Klassen K210 betyr beskyttelse mot antennelse i 10 minutter [klassene K1-A, K1 og K2]

Overflatekrav +
Vi bruker Euroklassene for å fastsette kravene til overflater som benyttes på vegg og tak. Med overflate menes her det ytterste laget av en bygningsdel (det du kan ta på), for eksempel overflatesjikt som dannes av maling, tapet og tilsvarende. Overflate må ses i sammenheng med underlaget som overflaten er på, som sponplate, gipsplate, isolasjonsmateriale og lignende. Klassifiseringen gjelder derfor det endelige produktet, det vil si overflaten på det aktuelle underlaget.
Overflater skal dokumenteres og testes etter xxxx

Kledningskrav +

Av Ignist
Personværsklæring og annen info

Figur 3: Femte og siste iterasjon laget i Figma

3.3 Farge

Fargepaletten ble eksperimentert med i de første iterasjonene. Fargene oransje, blå og lilla ble testet ut som fargefamilie på designet, med hvit som bakgrunnsfarge eller kontrastfarge og sort som tekstfarge.

Spesielt tekstfarge og bakgrunnsfarge, og kontrasten mellom disse to har vært et fokus under designprosessen, da teksten er en såpass viktig del av webapplikasjons oppgave. Fra den første iterasjonen har det gjennomgående vært sort eller svært mørk tekstfarge og en lys, eller helt hvit bakgrunn. Slik sikres en god kontrast mellom bakgrunnen og teksten og skaper gode leseforhold for flest mulig brukere. Figur 3 illustrerer dette.

Til den tredje iterasjonen hadde Ignist valgt en fargepalett som er monokromatisk og denne har preget de senere iterasjonene av UX-designet. Se figur 4. Monokromatiske fargeharmonier er basert på samme fargetone, der variasjon skapes ved å variere metning og lysgrad (Sandnes, 2022, s. 129). Fargepaletten ble basen for fargevalgene i de følgende og den siste iterasjonen av UX-designet. Ved å holde fargevalgene til en monokromatisk palett blir helhetsuttrykket behagelig å se på og sammenhengende i forhold til om man hadde valgt å trekke inn forskjellige farger.

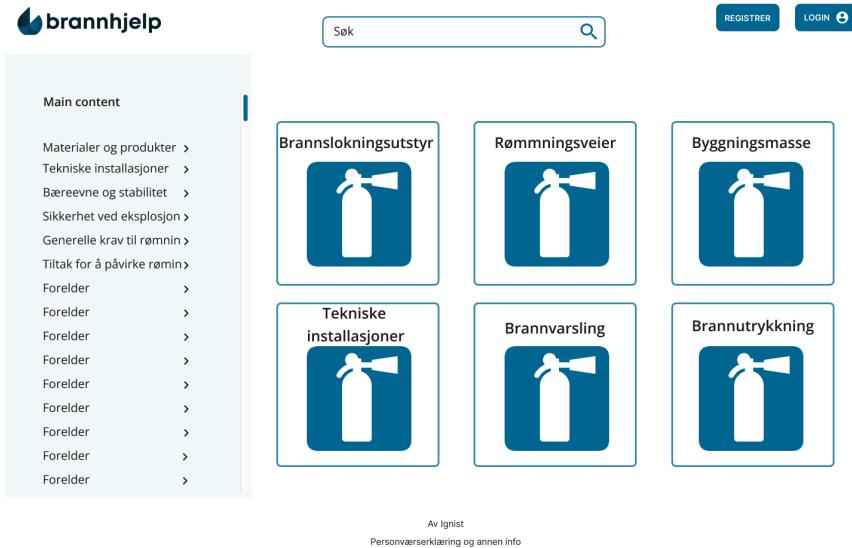
Color	Hex	RGB
	#00b8ff	(0,184,255)
	#009bd6	(0,155,214)
	#00719c	(0,113,156)
	#00415a	(0,65,90)
	#001f2b	(0,31,43)

Figur 4: Fargepalett valgt av Ignist.

Fra fargepalletten i Figur 4 var det den midtre fargen, med hexkoden 00719c som vi landet på som kontrastfarge i den siste og ferdige iterasjonen av UX-designet. Andre blåtoner brukt i siste iterasjon er blåtoner med denne som base og tilført mer hvitt slik at de ser lysere ut, som f.eks. blåtonen i bakgrunnen av kategorimenyen i Figur 3.

3.4 Ikoner

Ikonene brukt i illustrasjonene er hentet fra Iconify.design, som er en open source plug-in i Figma. Ignist eier rettighetene til egne ikoner som de vil bruke når tar i bruk webapplikasjonen for kunder. Spesifikt vil de bytte ut ikonene brukt på landingssiden, se figur 5. Derfor er ikonene brukt i Figma-modellene kun et eksempel på hvordan ikoner kan bli brukt i webapplikasjonen.



Figur 5: Landsingsside modell laget i Figma.

3.5 Font

Med hovedoppgaven å meddele informasjon via tekst for webapplikasjonen, var det spesielt viktig å tenke på fontvalg i tillegg til fargevalgene. Både administratorer som skal holde informasjonen relevant og riktig, og brukere som skal innhente informasjon er avhengige av god lesbarhet. Det må være god lesbarhet for så mange

som mulig, uavhengig av funksjonshemninger. På grunn av sin lesbarhet, var det konsensus om at en sans-serif font ville være et godt valg. En sans-serif font er en enkel font med ren visuell struktur (Sandnes, 2022, s. 90). Mer spesifikt ble fonten Open Sans valgt. Denne skriftypen ble designet for digitale flater og har god lesbarhet i bokstavdesignet (Google, u.å.). Denne fonten er gjennomgående i UX-designet til Brannhjelp, se Figur 3 og Figur 5. med unntak av logoen som Ignist selv sto for designet på.

3.6 UX-designet vs kode i løsningen

Det har blitt noen forskjeller på UX-designet i den faktiske koden levert til oppdragsgiver ved prosjektslutt sammenlignet med UX-designet utformet i Figma i samarbeid med oppdragsgiver.

En av disse forskjellene er font-typen. Som nevnt i delkapittel 3.5 valgte vi ut fonten Open Sans, og denne ble brukt gjennomgående i siste iterasjon av Figma modellen. I filene som ble overlevert til Ignist var fontene brukt font-familien apple-system, system-ui, BlinkMacSystemFont. Denne inneholder fontene Segoe UI, Roboto, Oxygen, Ubuntu, Cantarell, Fira Sans, Droid Sans og Helvetica Neue. Det som er spesielt med denne løsningen for font er at denne løsningen er kodet slik at avhengig av brukerens operativsystem, vil fonten som er standard for dette operativsystemet bli brukt for visningen av teksten i webapplikasjonen. Slik spares det på ressurser for operativsystemet. Alle fontene i font-familien er dog en sans serif font, og er derfor enkle å lese.

Fargene varierer også noe mellom den siste Figma-iterasjonen av webapplikasjonen og koden i det ferdige prosjektet. Bakgrunnen i den leverte koden er også hvit, og teksten svart. Aksentfargen her har hex-kodene 8cbfd3 og f2f8fa. Forskjellene i fargekoden skyldes noe dårlig kommunikasjon mellom studentene når den siste iterasjonen av UX-designet var ferdigstilt, og dette skulle inkorporeres i frontend koden. Bedre kommunikasjon i denne fasen av prosjektet ville sikret en ferdigstilt webapplikasjon som samsvarer med UX-designet som ble laget i samarbeid med Ignist.

Presentasjonen av artiklene som vises er også noe annerledes mellom UX-designet og den faktiske løsningen. I UX-designet er artiklene med brødtekst under samme forelder synlige når forelderen har blitt åpnet. Dette var et ønske Ignist presenterte sent i designprosessen, og slik koden allerede hadde blitt bygget med API-kall for hver artikkel var dette vanskelig å endre på, med tiden som da var til disposisjon. Det var først når den siste iterasjonen av UX-designet var under arbeid at Ignist kom med ønske om å kunne se mer enn én hel artikkel om gangen. Derfor ble dette ikke realisert i det ferdige prosjektet.

4 Utviklingsprosessen

Prosjektet har utviklet seg gjennom flere faser som demonstrert i Gantt-diagrammet i avsnitt 2.1, selv om hver fase har vært distinkt. Hver fase har vært inndelt i aktiviteter, hvorav noen aktiviteter har strukket seg over flere faser. Prosjektet har pågått over 21 uker. I dette avsnittet diskuteres prosessen i utviklingsfasen.

4.1 Kravspesifikasjon

Kravspesifikasjonen ble formalisert som et selvstendig dokument en måneds tid inn i prosjektarbeidet. Inntil da, ble det regnet med at studentene og oppdragsgiver hadde en felles forståelse av hva som skulle leveres, basert på e-postkorrespondanse, muntlige samtaler og Ignist sin initielle prosjektbeskrivelse. For å ytterligere styrke en felles forståelse av hva prosjektgruppen skulle levere og for å samle alle kravene på ett sted, utarbeidet studentene kravspesifikasjonen. Studentene utformet et forslag basert på tidligere kommuniserte krav og sendte denne til oppdragsgiver for innspill. Kravspesifikasjonen ble så redigert og godkjent av både studentene og Ignist.

Kravspesifikasjonen la grunnlaget for utviklingen av webapplikasjonen og har etter sin opprettelse vært et sentralt dokument gjennom hele utviklingsprosessen. En kravspesifikasjon er et såkalt levende dokument, som kan endres underveis i prosjektet dersom studentene eller oppdragsgiver ser at noe kan løses på en annen måte en først tiltenkt (Sommerville, 2016, s. 131). Kravspesifikasjonen i dette kapitelet er den siste versjonen. Endringer i kravspesifikasjonen er beskrevet i delkapittel 4.1.4.

Kravspesifikasjonen er inndelt i funksjonelle- og ikke-funksjonelle krav. Funksjonelle krav beskriver hva systemet skal gjøre, og sier altså noe om hvilke funksjoner et system skal ha. Funksjonelle krav kan også beskrive hvordan systemet skal oppføre seg i ulike scenarier og hvordan det skal reagere på input. Ikke-funksjonelle krav beskriver ofte systemet i sin helhet og kan blant annet definere systemets krav til brukervennlighet, sikkerhet og ytelse (Sommerville, 2016, s. 105-109).

4.1.1 Utgangspunkt

Ignist ønsker en brukervennlig webapplikasjon som kan fungere som et bibliotek av informasjon for deres kunder. Webapplikasjonen, kalt "Brannhjelp", skal bestå av en administrator- og en kunde-del. I dag har Ignist ingen eksisterende løsning for publisering av info med innloggingsfunksjon for deres kunder. Ignist ønsker at systemet utvikles med tanke på en fremtidig integrasjon av betalingsløsning. De ønsker også at informasjonen fra Brannhjelp skal kunne integreres inn i eksisterende programvare de har utviklet, kalt STRGI, via API.

4.1.2 Funksjonelle krav

- Nettsiden skal ha en søkefunksjon, som brukerne kan benytte til å finne innhold.
- Det skal være mulig for bruker å logge inn med to ulike roller, disse er administrator og kunde.
- Administrator-roljen skal ha tilgang til å redigere, opprette og slette innhold og kategorier.
- Administrator-roljen skal kunne administrere brukertilganger.
- Kunde-roljen skal ha lese-tilgang. Kunden skal ikke kunne redigere, opprette og slette innhold og kategorier. Kunde-roljen skal ikke kunne administrere brukertilganger.
- Dersom kunden ikke har et gyldig abonnement, skal kunden fortsatt ha lesertilgang, men miste denne etter en gitt tid.

- Personer som ikke er kunder (ikke logget på) skal kunne lese innhold, men innhold skal blokkeres etter kort tid, f.eks. etter 10 sekunder. Hensikten er å gi eksempel på innhold. Se kontohjelp.no.
- Det er ønskelig at systemet på sikt har en integrasjon mot betalingsløsning som brukere kan benytte for å betale for tilgang. Integrasjonen er ikke en del av leveransen, men webapplikasjonen skal være utviklet med tanke på integrasjon i fremtiden.
- Det må være mulig å hente data fra Brannhjelp til STRGI via et API.
- Det er ønskelig å kunne tilrettelegge for potensielle underartikler til hver hovedartikkkel, dersom det er mulig innenfor prosjektets rammer.

4.1.3 Ikke-funksjonelle krav

- UX-design og brukergrensesnittet skal være intuitivt og brukervennlig, slik at webapplikasjonen gir en positiv brukeropplevelse.
- Databasen som brukes skal være Microsoft Azure Cosmos.
- Nettsiden skal støtte publisering av tekst. Dersom det er mulig innenfor prosjektets rammer, er det ønskelig at det er mulig å publisere bilder og tabeller.

4.1.4 Endringer i kravspesifikasjonen

Kravspesifikasjonen gjennomgikk ingen skriftlige endringer etter at den ble formalisert som et eget dokument. Imidlertid var det behov for å avklare visse punkter i kravspesifikasjonen, noe som ble gjort gjennom muntlige diskusjoner uten nedtegnede endringer. Til tross for fraværet av formelle endringer, vil det her presenteres to vesentlige endringer i kravene.

Da samarbeidet med Ignist startet, var det enighet om at systemet skulle ha en integrasjon mot betalingsløsning. Imidlertid ble det etter hvert tydelig at dette ikke var mulig innenfor prosjektets rammer. I den første formelle kravspesifikasjonen ble kravet om integrasjon mot betalingsløsning derfor endret. I stedet for å tilby en betalingsløsning direkte, ble det krav om å utvikle systemet med tanke på en slik integrasjon i fremtiden.

To av de funksjonelle kravene nevner ordet "kategorier", men det var misforståelser rundt hva dette betyddet. Under utviklingen var det usikkert om kategoriene refererte til "tags" knyttet til publikasjoner, for eksempel å kategorisere en artikkkel om rømningsveier med kategorien "risikotiltak". I et møte med Ignist ble det imidlertid presisert at med "kategorier" mentes det informasjonskategorier som vises på webapplikasjonen med under-kategorier. For eksempel, kategorien "Materialer og produkters egenskaper ved brann" med underkategorien "Overflatekrav og brannbeskyttende kledning".

4.2 Utforskning av rammeverk

Miljø for utvikling av et produkt har stor betydning. Valgene man gjør her kommer med begrensinger og muligheter. I dette delkapittelet vil vi redegjøre for valgene som har blitt tatt når det kommer til rammeverk og hvorfor vi har tatt disse valgene.

4.2.1 Rammeverk for backend

Backend er den skjulte delen av en applikasjon, og er delen som driver applikasjonen. Den inkluderer server-logikken, databasesystemer, API-er og mer. Det er viktig å velge et rammeverk som er brukervennlig og som

gjør overtagelse av eksisterende prosjekter enkelt. Spesielt i et prosjekt som dette, hvor oppdragsgiver ønsker å videreutvikle applikasjonen etter overlevering av prosjektet.

For dette prosjektet har vi valgt å bruke .NET sammen med C# for backend-utvikling. Dette valget er basert på flere viktige aspekter som samsvarer med prosjektets mål og krav. .NET er et kraftig, åpent kildekode-rammeverk utviklet av Microsoft. Det er designet for å lage effektive webapplikasjoner, API-er og mikrotjenester. Denne kombinasjonen tilbyr et sterkt miljø som kan håndtere store mengder trafikk, samtidig som det opprettholder hastighet og sikkerhet. Dette er viktig å ta hensyn til i dagens digitale marked.

Vi har valgt C# som programmeringsspråk for backend-logikk i denne webapplikasjonen. C# er kjent for å være et kraftig og fleksibelt programmeringsspråk, og dets objektorienterte natur gjør koden lesbar og enkel å vedlikeholde. C# har et omfangsrikt standardbibliotek som forenkler komplekse oppgaver som databaseoperasjoner og nettverkskommunikasjon. Språket støtter også moderne programmeringsparadigmer som asynkron programmering, noe som er viktig for å bygge responsive og effektive applikasjoner.

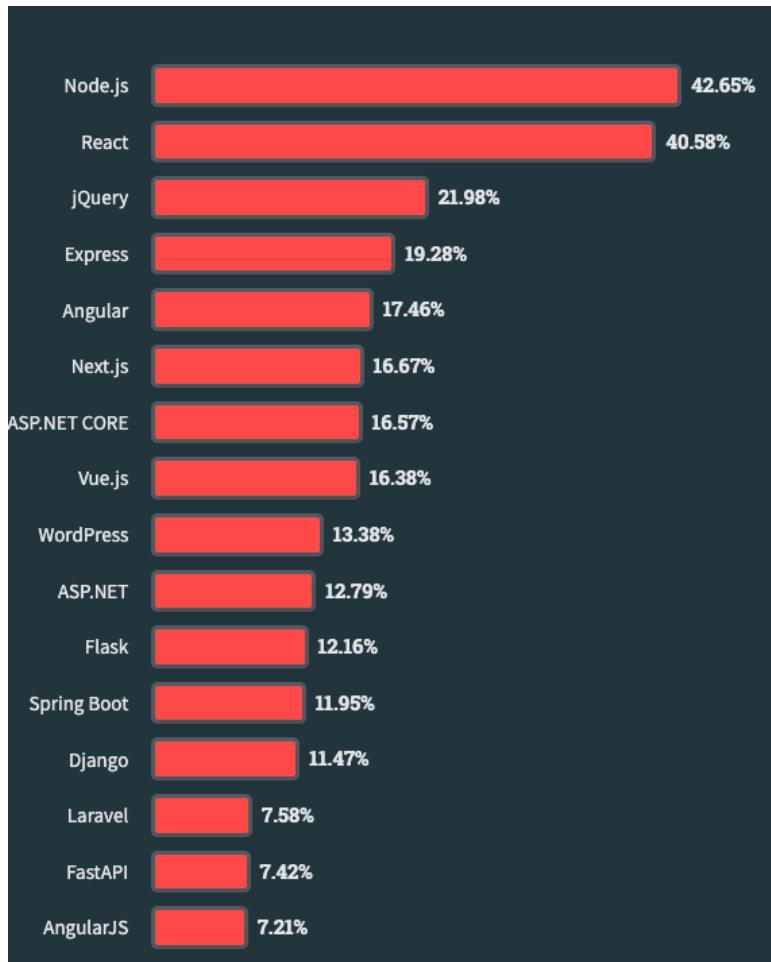
ASP.NET Core er en viktig del av .NET-rammeverket og fungerer godt i kombinasjon med C#. ASP.NET Core har innebygd støtte for Dependency Injection, som gjør det enkelt å legge til nødvendige komponenter direkte i klasser. Dette gir en løs kobling og gjør det lettere å teste hver del uavhengig. Samtidig tillater ASP.NET Core enkel og rask utvikling av komplekse applikasjoner med ren og enkel vedlikeholdbar kode. C# sitt omfattende standardbibliotek, kombinert med ASP.NET Core sine mange funksjoner, som støtter for Dependency Injection, utvikling av RESTful API-er og avanserte sikkerhetsfunksjoner, gjør det mulig å lage webapplikasjoner som er sikre, skalarbare og av høy kvalitet. I tillegg støtter ASP.NET Core utvikling på tvers av plattformer, dette gjør det fleksibelt for et team av utviklere å jobbe sammen.

4.2.2 Rammeverk for frontend

I dagens digitale verden er det avgjørende å skape interaktive og brukervennlige nettsider. Frontend-rammeverk spiller en sentral rolle i å forenkle denne prosessen. Disse rammeverkene kan tilby forhåndsdefinerte strukturer og verktøy, som utviklere kan bruke for å bygge komplekse webapplikasjoner på en mer effektiv måte. Blant de mest populære frontend-rammeverkene har man React, Angular og Vue.js.

Bruken av frontend-rammeverk gir flere fordeler i webutvikling. Det gjør det lettere å organisere og strukturere kode, noe som fører til enklere vedlikehold og skalarbarhet. Rammeverkene kommer også med en rekke innebygde funksjoner og verktøy som reduserer behovet for å skrive kode fra bunnen av. Dette sparar tid og reduserer risikoen for feil. Videre bidrar rammeverkene til å sikre kontinuitet i brukeropplevelsen, uavhengig av utviklerne som jobber på prosjektet.

Det finnes flere frontend-rammeverk som er populære blant utviklere i dag. Ifølge den årlig utviklerundersøkelsen utført av Stack Overflow er rammeverkene vist på figur 6 er de mest populære. React, utviklet av Facebook, er kjent for sin enkelhet og fleksibilitet. Dette rammeverket bruker en komponentbasert tilnærming som gjør det enkelt å gjenbruke kode. Angular, fra Google, er et omfattende rammeverk som tilbyr en fullverdig løsning for både små og store applikasjoner. Mens Vue.js, skapt av Evan You, er kjent for gode ytelse og intuitive API, noe som kan gjøre denne til et godt valg for nybegynnere.



Figur 6: Det mest brukte rammeverket (Stack Overflow, 2023)

Vi har valgt å bruke React som rammeverk i prosjektet. Dette valget diskuteres i de kommende delkapitlene.

4.2.2.1 Enkelt å forstå

React er et enkelt og lettforstårlig frontend-rammeverk sammenlignet med andre. React's læringsvennlige natur var særlig tiltalende for oss med begrenset erfaring med komplekse rammeverk fra tidligere av, noe som gjorde valget åpenbart. Dette valget resulterte i redusert tid brukt på feilsøking og økt produktivitet.

4.2.2.2 Raskere utvikling av applikasjoner

React er et rammeverk som er bygget opp rundt konseptet komponentbasert arkitektur, hvilket innebærer at React muliggjør optimalisering gjennom gjenbruk av komponenter. Dette komponentbaserte perspektivet gjør det mulig for utviklere å effektivt overvåke og spore prosjektets fremgang, og fremmer dermed en mer modulisert og strømlinjeformet utviklingsprosess.

4.2.2.3 Forbedret langsiktig applikasjonsstabilitet

Bærekraft og stabilitet i applikasjonen er avgjørende for effektiviteten av webapplikasjoner. En webapplikasjons levetid avhenger av et rammeverk eller en plattform som kan utvikles og oppdateres for å integrere ny funksjonalitet over tid. React står i en særstilling til å unngå foreldelse. Dens evne til å understøtte vedvarende

utvikling av brukergrensesnitt for webapplikasjoner gjør at mange foretrekker React fremfor andre plattformer for sin lange levetid.

4.3 Kvalitetssikring

Kvalitetssikring (QA, fra engelsk Quality Assurance) var et sentralt område i vårt prosjekt. For å sikre at vi oppfyller de definerte kravene i kravspesifikasjonen hadde vi satt oss rutiner som ville sikre utvikling og kvaliteten av webapplikasjonen på sikt. Fra første uke da vi startet med å planlegge prosjektet, hadde studentene avtalt ukentlige møter og dokumentering av disse. For utviklingens del ble GitHub Project benyttet som verktøy, tilbuddt av GitHub. Dette verktøyet kan brukes som en tavle hvor man legger inn aktiviteter som er i ulike faser av prosjektet. Denne aktiviteten kan direkte knyttes til selve repositoriet i GitHub. En aktivitet kan skrives med tanke på om den er i startfase eller om man jobber aktivt med den.

4.3.1 Rutiner for Git

For å sikre konsekvent og effektiv bruk av Git under utviklingen, har vi tatt i bruk følgende rutiner. Disse ble brukt for å støtte en smidig utviklingsprosess og for å fremme kvalitet og sikkerhet i det ferdige produktet.

- Repository Oppsett:

Vi valgte å skille frontend og backend komponenten i to forskjellige GitHub-repositories fra starten av utviklingen. Dette gjorde det enklere for teamet å fokusere på spesifikke deler av prosjektet uten forstyrrelser fra den andre delen. Dette bidro også til klarere ansvarsfordeling og færre konflikter underveis i koden.

- Branching Strategi:

Master Branch: Hovedgrenen for utviklingen, og all kode som merges inn i denne skulle være grundig testet og verifisert.

Utvikling Branch: Utviklerne committet først endringene til egen branch. Denne personlige branchen ble brukt til å utvikle og teste nye funksjoner eller rettelser. Når koden var klar og godkjent av de andre i gruppen ble den deretter merget til Master Branchen.

- Commit Praksis:

I tilfeller der det oppsto merge-konflikter, var det ansvaret til den som utførte mergen å koordinere med studentene som hadde ansvar for den opprinnelige koden for å finne en løsning.

- Sikkerhet:

Tilgang til repositoriene var begrenset til autoriserte teammedlemmer. For å ytterligere sikre våre data, ble repositoriene kontinuerlig sikkerhetskopiert.

Ved å følge disse Git-rutinene, sikret vi en strukturert og effektiv prosess i utviklingen. Dette har bidratt til å styrke samarbeid, redusere feil og økt kvaliteten i prosjektet. Til sammen har dette sikret at det leveres en pålitelig og sikker programvareløsning.

4.4 Databasemodellering

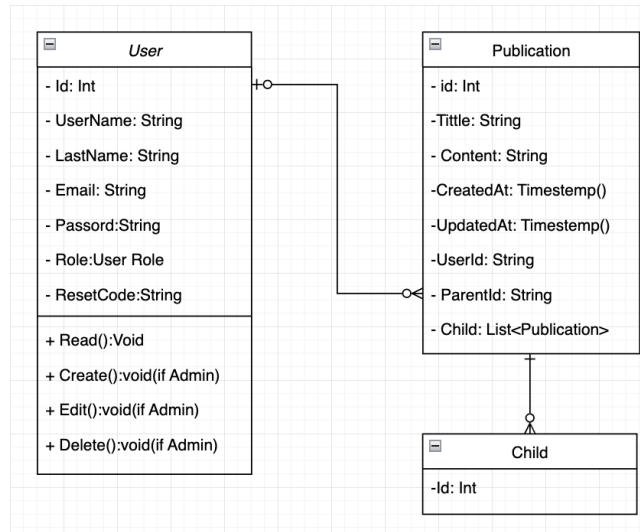
Oppdragsgiver presiserte at Cosmos DB skulle brukes som database i utviklingen. Dette presenterte en utfordring, ettersom det var mangel på tidligere erfaring med denne teknologien i teamet. Oppdragsgivers begrunnelse for valget av Cosmos DB skyldtes at det er en robust og effektiv databasemodell, med evnen til å imøtekommе både de nåværende og framtidige behovene til webapplikasjonen. Målet med databasemodelleringen var å etab-

lere en strukturert lagringsmekanisme som ikke bare beskytter og gjør data umiddelbart tilgjengelig, men også forbedrer håndtering av forespørsler for å øke ytelsen til applikasjonen.

Databasen er konstruert for å fungere som hjertet i applikasjonen, en sentral reporistory hvor all essensiell data lagres. I den inngående analysen prioriterte vi etableringen av tydelige og sammenhengende relasjoner mellom de forskjellige datasegmentene. Dette tiltaket ikke bare forenkler håndtering av data, men legger også til rette for skalering og utvidelse av databasen i takt med behov.

Gjennom en analytisk prosess i samarbeid med oppdragsgiver, ble dataspesifikasjonene for Brannhjelp klargjort. Denne prosedyren garanterer at alle kritiske datakomponenter ble inkludert for å underbygge applikasjonens funksjoner og de overordnede målene.

For å illustrere relasjonen mellom de ulike entitetene i databasen, har det blitt laget et entitets-relasjonsmodell (ERD), som er presentert i figur 7. Dette diagrammet visualiserer datamodellens struktur og hvordan de forskjellige klassene kommuniserer effektivt. En av de fundamentale relasjonene som er kartlagt i ERD, er den hierarkiske sammenkoblingen mellom “Publication”-entiteter (som fungerer som “forelder”) og potensielle “Child”-entiteter, som er underartikler.



Figur 7: Klassediagram

I denne modellen utgjør “Publication” primærentiteten og representerer en distinkt enhet i applikasjonen. Denne entiteten inneholder essensielle attributter, herunder en unik identifikator “Id”, som fungerer som primærnøkkelen. “Child” er definert som en avledet entitet som er forbundet med, og arver attributter fra primærentiteten.

Relasjonen mellom artikkel og underartikkel er en en-til-mange kobling. Dette innebærer at en artikkkel kan assosieres med et vilkårlig antall underartikler, men hver underartikkel er kun tilknyttet én enkelt artikkkel. For å administrere denne sammenhengen anvendes “Id” fra Publication”-entiteten som en fremmednøkkel i “Child”-entiteten.

5 Testing

I dette kapittelet beskrives de ulike testene som er utført, hvordan testene ble gjennomført og resultater fra testene. Testene som har blitt utført og beskrives, er enhetstest, systemtest, tilgjengelighetstest og akseptansetest. Videre følger delkapitler som tar for seg hva slags innsikt de ulike testene har gitt, og samsvar mellom mål fra testplanen og de faktisk utførte testene.

Testing er nødvendig for å forbedre kvaliteten på systemet og redusere risiko for at feil oppstår når systemet er i drift (Norwegian Testing Board & International Software Testing Qualifications Board [NTB & ISTQB], 2019, s. 13). I prosjektet har det blitt utført tester på tre ulike testnivåer, disse er enhetstest, systemtest og akseptansetest. Testnivåene har ulike mål, for eksempel skal enhetstest finne feil i de minste komponentene i koden, systemtest benyttes til å validere at systemet fungerer som forventet og akseptansetest skal si noe om systemet er godkjent av oppdragsgiver (NTB & ISTQB, 2019, s. 28-33). Ved å utføre disse ulike testene, har vi fått gjort både funksjonelle- og ikke-funksjonelle tester. Funksjonelle tester, tester hva systemet skal gjøre, mens ikke-funksjonelle tester sier noe om hvordan systemet virker (NTB & ISTQB, 2019, s. 37).

I begynnelsen av utviklingsfasen, ble det utformet en testplan som er vedlagt som vedlegg B. En testplan beskriver systemet som skal testes, hvilke type tester som skal utføres og hvorfor, mål for testene og hvordan testene skal gjennomføres (NTB & ISTQB, 2019, s. 64). Testplanen ble justert ved behov for endringer. For eksempel ble en tilgjengelighetstest inkludert i testplanen etter hvert, ettersom studentene så behov for å teste om webapplikasjonen er universelt utformet.

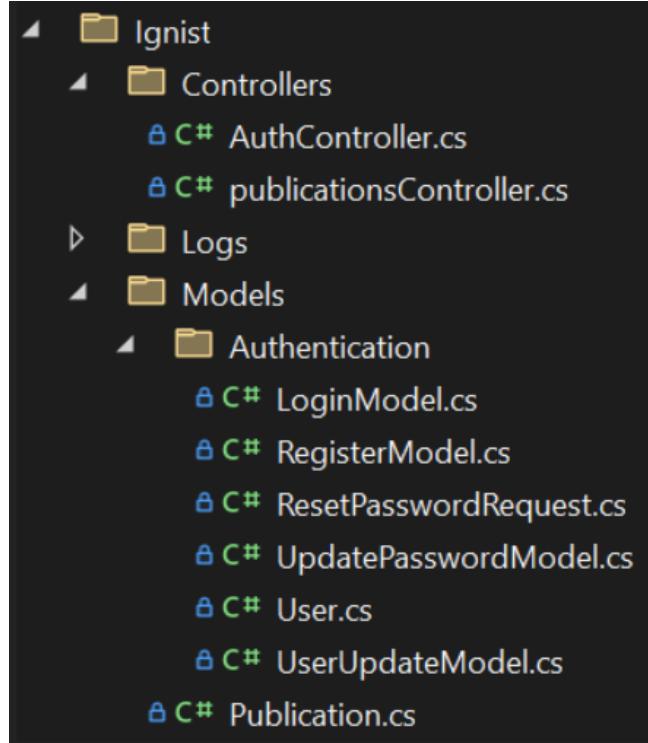
5.1 Enhetstest

Enhetstestene tester de minste komponentene i koden, og sjekker at metoder oppfører seg som tiltenkt (NTB & ISTQB, 2019, s. 28-29). Målet med enhetstestene var å oppdage så mange feil i koden som mulig, tidlig i utviklingsprosessen.

Enhetstestene består av både negative og positive tester. Negative tester simulerer scenarioer hvor noe går galt, og sjekker dermed hvordan metoder håndterer feilsituasjoner. Positive tester sjekker at metoden gjør det den skal når den får korrekt input. Et eksempel kan være en metode for å registrere en bruker. I en positiv test kan man da forsøke å registrere en bruker med gyldige brukerdata og se at metoden faktisk gjennomfører dette. I en negativ test kan man fremprovosere en feil ved å registrere en kunde uten et påkrevd felt, for eksempel en epostadresse. Deretter testes det at metoden håndterer denne feilsituasjonen ved å for eksempel gi en feilmelding og at registrering av kunden ikke gjennomføres.

5.1.1 Gjennomføring av enhetstester

Enhetstestene ble utført løpende under utviklingsprosessen, og det ble skrevet kode for enhetstester etter hvert som metoder og funksjonalitet var på plass. Ved oppdagelse av feil, ble koden rettet og re-testet frem til testene passerte. Testobjektene, som vist i figur 8, var systemets to kontrollere, “AuthController” og “PublicationController”, samt modellene “LoginModel”, “RegisterModel”, “ResetPasswordRequest”, “UpdatePasswordModel”, “User”, “UserUpdateModel” og “Publication”.



Figur 8: Testobjekter for enhetstester

Testene ble utviklet i et eget prosjekt i Visual Studio med en referanse til Brannhjelp-prosjektet. Testprosjektet ble opprettet ved å benytte “xUnit Test Project”-malen, og er kompatibelt med .NET. For å få ekstra funksjonalitet, ble det benyttet en NuGet-pakke kalt “Moq.EntityFrameworkCore”. Dette biblioteket gir metoder for å opprette simulerte versjoner av databasen og kontrollere, kalt “mocks”. Mocks tillater oss å etterligne adferden til databasen og kontrollere. Testene kjøres altså mot en simulert versjon av databasen og kontrollerne. Dette muliggjør isolering av oppførselen til database og kontrollerne, slik at logikk i koden kan testes uavhengig av den faktiske databasen og kontrolleren.

Hver enkelt test skrives som en metode, se figur 9 for et eksempel på en testmetode. Denne testen simulerer et positivt scenario hvor en ny bruker blir registrert, og testen sjekker at registreringsmetoden fungerer som tiltenkt når den får gyldige inputverdier. Hver testmetode deles inn i en “arrange”, “act” og “assert”-del. I arrange-delen av testene blir miljøet for testen forhåndsdefinert. Nødvendige objekter og verdier for testen opprettes, i dette tilfellet en ny bruker som skal registreres. På linje 42 mockes databasen. På linje 43-44 settes mock-versjonen av databasen opp til å utføre metoden vi ønsker å teste, med resultatet vi ønsker å fremprovosere. På linje 45 mockes kontrolleren som testes. Deretter følger act-delen av testene, hvor registrerings-metoden utføres og resultatet av den lagres i variabelen “result”. Til slutt er assert-delen, hvor det bekreftes at registreringen fungerte som forventet.

Resultatet fra act-delen sammenlignes med resultatet man forventer å få. I dette tilfellet ønsker vi å se at metoden returnerer et resultat av typen “OkObjectResult” og gir meldingen “User registered.”, som forventet.

```

33 |     [Fact]
34 |     public async Task TestRegister_Positive()
35 |     {
36 |         //arrange
37 |         var registerModel = new RegisterModel()
38 |         {
39 |             UserName = "hanne",
40 |             LastName = "Larsen",
41 |             Email = "hanne-lf@hotmail.com",
42 |             Password = "password123";
43 |         };
44 |         var mockCosmosDbService = new Mock<ICosmosDbService>();
45 |         mockCosmosDbService.Setup(repo => repo.RegisterUserAsync(registerModel))
46 |             .ReturnsAsync("User registered.");
47 |         var authController = new AuthController(mockCosmosDbService.Object);
48 |
49 |         //act
50 |         var result = await authController.Register(registerModel);
51 |
52 |         //assert
53 |         var okResult = Assert.IsType<OkObjectResult>(result);
54 |         Assert.NotNull(okResult);
55 |         Assert.Equal("User registered.", okResult.Value);
56 |

```

Figur 9: Testmetode for å teste gyldig registrering av ny bruker

Ved igangsettelse av enhetstestene, oppstod det problemer med mocking av databasen. Prosjektet hadde på dette tidspunktet kun én såkalt “Data access layer”-fil (DAL) med navn DataContext.cs. Kjøring av enhetstest feilet med feilmelding om at Moq ikke kunne instansiere en mock-versjon av DataContext direkte. For å løse dette problemet, måtte testprosjektet endres til å teste et grensesnitt av DataContext i stedet for å teste direkte mot DataContext. For å kunne enhetsteste med mocks, måtte det derfor opprettes slike grensesnitt for alle kontrollerne. På dette tidspunktet var det kun utviklet en kontroller for publikasjon, så det var kun dette datagrunnlaget som det måtte opprettes grensesnitt for databasetilgang for. I videre utvikling av prosjektet, ble utviklingen gjort slik at kontrollere fikk egne DAL fra start.

5.1.2 Resultater fra enhetstester

Ved endt enhetstesting var det totalt skrevet 54 tester, hvor alle testene er bestått, se figur 10. Ved identifisering av feil, ble tilbakemelding om dette gitt til utviklingsteamet. Utviklingsteamet rettet opp i feil fortløpende og testene ble kjørt på nytt.

Test run finished: 54 Tests (54 Passed, 0 Failed, 0 Skipped) run in 1,3 sec	
Test	Duration
UnitTests-Ignist (54)	412 ms
UnitTests_Ignist.Controllers (40)	395 ms
AuthControllerTests (29)	192 ms
PublicationControllerTests (11)	203 ms
UnitTests_Ignist.Models (2)	1 ms
PublicationModelTests (2)	1 ms
UnitTests_Ignist.Models.Authentication (12)	16 ms
LoginModelTests (1)	6 ms
RegisterModelTests (4)	2 ms
ResetPasswordRequestTest (6)	2 ms
UpdatePasswordModelTests (1)	6 ms

Figur 10: Kjøring av alle enhetstester

Enhetstestene identifiserte en metode som ikke bestod sine tester. Det ble ikke tatt skjermbilde av denne hendelsen, men feilen ble rettet av utviklerne. Den identifiserte feilen var en metode som manglet feilhåndtering. Metoden “getAllPublications()”, returnerte et resultat som indikerte at alt var veldigkvet, selv om testen var skrevet for å fremprovosere metoden til å feile. Metoden ble rettet til å returnere en feilmelding i feilsituasjoner, og testen passerte etter dette. I senere iterasjoner av koden, ble imidlertid denne metoden slettet, og tilhørende enhetstester ble da også fjernet. Selv om vi ikke har dokumentasjon av feilet test i form av skjermbilder, viser

dette allikevel at enhetstestene var nyttig for å identifisere og rette feil i koden.

5.2 Systemtest

Systemtest tester systemets funksjonalitet og ytelse, og sjekker at webapplikasjonen samsvarer med kravspesifikasjonen. Systemtest er nødvendig for å undersøke brukervennlighet, om systemet oppfører seg som forventet og for å forbedre kvaliteten til systemet ved å finne og rette feil (NTB & ISTQB, 2019, s. 32).

5.2.1 Gjennomføring av systemtest

Systemtestene ble utført som manuelle tester. Som forberedelse til gjennomføring av systemtest, ble først kravspesifikasjonen nøye gjennomgått for å få oversikt over hva webapplikasjonen skulle gjøre og dermed hva som skulle testes. Deretter ble det skrevet brukerhistorier som skulle ta for seg kravene i kravspesifikasjonen, alle brukerhistoriene er vedlagt i vedlegg C. Det ble identifisert forskjellige scenarioer fra brukerhistoriene, hvor hvert scenario fikk én test case. Test casene er vedlagt som vedlegg D. Hver test case har en unik identifikator, og inneholder følgende informasjon: en beskrivelse av test scenarioet, forutsetninger som må være møtt for at testen kan utføres, stegene i testen, resultat og om testen er bestått eller ikke. Eksempel på en brukerhistorie og tilhørende test cases er:

Brukerhistorie: "Som person uten allerede opprettet konto, vil jeg ha mulighet til å registrere meg som kunde på Brannhjelp og deretter kunne logge inn med denne brukeren."

ID	Beskrivelse av testscenario	Forutsetninger	Steg	Resultat	Godkjent/Feilet
1.1	Man skal kunne registrere seg som kunde på Brannhjelp	Personen har ikke en registrert bruker og har tilgang til registreringssiden	Gå til Brannhjelpp nettsiden Trykk på "Don't have an account? Sign up" Fyll inn navn, epost og passord Trykk på "Sign up"	Brukere ble opprettet	Godkjent
1.2	Kunde skal kunne logge inn	Kunden har allerede en registrert bruker og forsøker å logge inn med korrekt epost og passord	Gå til Brannhjelpp nettsiden Trykk på "Login" Fyll inn epost og passord Trykk på "Log in"	Ble logget inn og videresendt til forsiden	Godkjent

Figur 11: Test cases

Systemtest ble utført i to runder. I første runde ble alle test casene gjennomgått. Testene som ikke bestod denne, ble videreført til neste runde. Utviklingsteamet rettet opp i feil som ble oppdaget i runde en, før testrunde to ble utført. Det ble vurdert å gjennomføre en tredje testrunde hvor alle test casene ble gjennomgått en siste gang, for å undersøke om rettingen som ble gjort mellom runde en og to introduserte nye feil eller forårsaket uventede problemer. Dette var det dog ikke nok tid til å gjennomføre.

Et av kravene i kravspesifikasjonen er at Brannhjelp skal være brukervennlig. Det er utformet flere test cases for å forsøke å dekke over dette kravet. For eksempel er det en test case som tar for seg hvorvidt administrator kan

se rollene til hver enkelt bruker, som vil være nyttig informasjon for en administrator når hen skal drive bruker-administrering. Ellers ble det også lagt fokus på å notere og se etter om systemet gir brukeren tilbakemelding på utførte handlinger, og om systemet gir feilmeldinger når noe har gått galt.

5.2.2 Resultater fra testrunde 1

Fra første testrunde var det 4 av 21 test cases som feilet og ble videreført til neste runde. I tillegg til disse fire feilene testene, ble det oppdaget en knapp som ikke utførte noen handling. Alle disse fem feilene ble kommunisert til utviklerne for retting. En komplett liste over test cases og deres resultater fra begge testrundene er vedlagt som vedlegg D.

Feilene som ble identifisert i første testrunde var:

1. Ikke alle brukere vises under «Manage users»
2. Ikke alle publikasjoner vises under «All publications»
3. Administrator kan ikke se hvilken rolle en bruker har
4. Man kan endre e-postadressen sin til å være en ugyldig e-post
5. En knapp uten funksjon

På siden for administrasjon av brukere, var det ikke alle brukere som var synlig. Dette kan føre til at administratorer ikke kunne se eller administrere alle registrerte brukere. Det samme gjelder på siden for alle publikasjoner. Dette kan resultere i at publikasjoner ikke er synlige for administrator, og gir utfordringer ved administrering, endring og sletting av publikasjoner. Det var usikkert hvorfor ikke alle brukere og publikasjoner ble vist, men det ble mistenkt at det var en feil ved henting av data fra databasen. Disse feilene diskuteres videre i neste delkapittel, 5.2.3.

Administrator skal kunne administrere brukertilganger ved å bestemme hvilken rolle en bruker skal ha. Da er det hensiktsmessig at administrator kan se hvilken rolle en bruker har, men dette var det ingen muligheter for. Dette kunne gjøre det utfordrende for administrator å skille mellom ulike brukere og deres tilgangsnivå. Dette ble løst ved å legge til en kolonne i oversikten over brukere, som viser hvilken rolle en bruker har.

Dersom brukeren gikk inn på sin egen profil, kunne brukeren endre sin e-postadresse til å ha en ugyldig verdi. Det ble testet med en epost uten alfakrøll, og denne endringen ble lagret. Dette kunne føre til at brukere ble registrert med ugyldig kontaktinformasjon, samt skape problemer for resetting av passord da man må være registrert med en gyldig e-post for å få tilbakestilt passordet sitt. Problemet ble løst ved å endre inputvalidering i koden til å ikke godta ugyldige e-postverdier. I tillegg ble det implementert informerende feilmelding til brukeren om at e-posten er ugyldig.

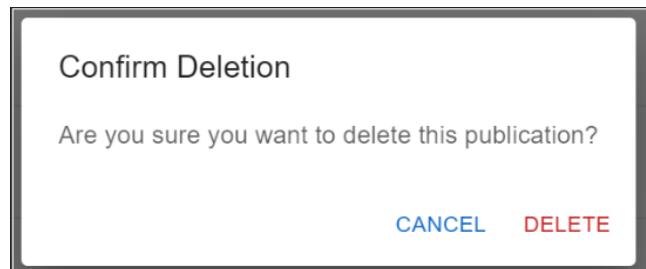
Det ble også identifisert en knapp uten funksjon. Dette var en pil som kunne trykkes på, se figur 12, som indikerte at man kan sortere publikasjoner på id. Pilen utførte derimot ingen handling ved klick på den. Sorteringspilen ble i utgangspunktet implementert i koden mens id ble tildelt publikasjoner i stigende rekkefølge. I ettertid har tildelingen av id blitt endret til tilfeldige verdier, og sorteringen fungerer dermed ikke lenger. Problemet ble løst ved å slette knappen.

Ellers kom det frem av testrunden at alle handlinger som ble testet, ga tilbakemeldinger til brukeren. Enten i form av melding om at en endring ble utført, eller feilmelding dersom noe ikke kan utføres eller om noe gikk galt. Et eksempel er når administrator skal opprette en ny publikasjon, hvor man da får en bekrefteelse på skjermen

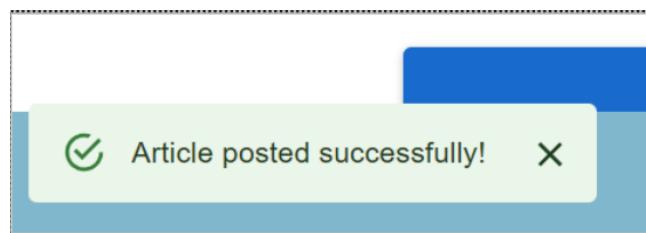
ID ↑	Title
4647	Materialer og produkter

Figur 12: Feil 5 - knapp (pilen) som ikke utfører noen handling

om at publikasjonen ble opprettet, se figur 13. Man må også bekrefte sletting dersom man forsøker å slette noe, se figur 14. Dette minimerer sjansen for å slette noe ved en feiltagelse.



Figur 13: Alert som ber om bekreftelse før sletting



Figur 14: Bekreftende melding om at artikkel er opprettet

5.2.3 Resultater fra testrunde 2

I runde to av testingen, ble alle testene som feilet i første runde kjørt på nytt. Alle testene bestod, men det var to tester som ble bestått under tvil. Dette gjelder feilene hvor ikke alle brukere og publikasjoner kunne ses på deres oversiktssider. I testrunde to ble det derimot identifisert at de kunne vises, men man må gjennomføre ekstra steg for å se dem. Dette skyldes et element på siden, en “footer”, som blokkerer visningen av nederste bruker og publikasjon. På noen skjermstørrelser vil altså ikke den siste brukeren og publikasjonen være synlig uten å zoome ut på siden. Figur 15 og 16 simulerer denne utfordringer. Den nederste publikasjonen som skal vises på oversikten, er en artikkel med navn “TEST”. Figur 15 viser hvordan nettsiden ser ut når bruker er helt nederst på siden. Selv om man har navigert til bunnen, kan man ikke se den siste publikasjonen på tabellen. Figur 16 viser hvordan det ser ut dersom man zoomer ut fra siden. Da vises siste publikasjon, “TEST”.

2621	Child	
9179	Grand Child	
8091	Tekniske installasjoner	
6656	Bæreevne og stabilitet	
2594	Sikkerhet ved eksplosjon	
1496	Tiltak for å påvirke rømning	

Copyright © Ignist 2024.

Figur 15: Ikke alle publikasjoner er synlige på siden

ID	Title	Actions
4647	Materiale og produkter	
2621	Child	
9179	Grand Child	
8091	Tekniske installasjoner	
6656	Bæreevne og stabilitet	
2594	Sikkerhet ved eksplosjon	
1496	Tiltak for å påvirke rømning	
9098	TEST	

Copyright © Ignist 2024.

Figur 16: Alle publikasjoner vises dersom man zoomer ut på siden

Brukeren må altså utføre en ekstra handling for å se alle publikasjoner og brukere på nettsiden. Det er heller ikke intuitivt at løsningen er å zoome ut. Dette er ikke brukervennlig, men brukervennligheten kan forbedres ved å justere posisjonen til bunnelementet eller legge til mer avstand i designet.

5.3 Tilgjengelighetstest

Tilgjengelighetstest gjøres som en del av systemtesten og er en type ikke-funksjonell test hvor man tester om systemet er universelt utformet og tilgjengelig for alle, uavhengig av funksjonsnedsettelse eller andre begrensninger (NTB & ISTQB, 2019, s. 39). Ettersom webapplikasjonen etter hvert skal tilgjengeliggjøres for Ignist sine kunder, var fokus på UX design et viktig krav i kravspesifikasjonen. Norske nettsider skal være universelt utformet, dette gjelder både for offentlige og private virksomheter (Digitaliseringsdirektoratet, u.å.-a). Ignist og Brannhjelp skal følge WCAG-standarden (Digitaliseringsdirektoratet, u.å.-b).

For å forsikre at Ignist følger WCAG-standarden for fargebruk har det blitt utført en kontrastsjekk av fargene brukt i den siste iterasjonen av Figma modellen, se Figur 3. I krav 1.4.3 av WGAC-standarden heter det “Hovedregel er at tekst skal ha kontrast mot bakgrunnen på minst 4,5:1.” (Digitaliseringsdirektoratet, u.å.-c). For å utføre testen har det blitt brukt en nettbasert kontrastsjekker (Webaim, u.å.). Det ble utført tre tester. Tekstfargen ble sjekket mot bakgrunnen, som fikk resultatet 16.81:1. Tekstfargen ble også sjekket mot bakgrunnsfargen på venstre sidebar, og resultatet ble 15.68:1. Disse to testene dekker de to hovedområdene hvor det skal være tekst i webapplikasjonen. Den siste testen var detaljfargen (hexkode 00719C) mot bakgrunnsfargen, hvor resultatet ble 5.46:1. Denne siste testen er ikke tekst, og er derfor ikke nødvendig at passerer testen, men for å sikre at alle fargevalgene er innenfor rimelige krav ble denne også testet. Alle tre testene var innenfor WCAG-standarden.

5.4 Akseptansetest

Det er gjerne kunde som utfører akseptansetest. Akseptansetest utføres for å undersøke om systemet møter kundens krav og gjøres før systemet overføres til kunden og eventuelt settes i drift (Spillner et al., 2014, s. 61).

Akseptansetest ble utført under et overtagelsesmøte med veileder fra Ignist. Studentene hadde fått noen punkter til forbedring før det planlagte overtagelsesmøtet. Blant annet skulle det gjøres mindre endringer ved å fjerne skygge rundt en artikkel. I tillegg var det i navigasjonsbaren mulig for en vanlig bruker å se knapper som kun skulle være tilgjengelige for administrator. Ignist ønsket at linker/knapper som ikke er relevante for brukeren skulle skjules. Under overtagelsesmøtet demonstrerte studentene at disse endringene var utført og Ignist bekreftet at dette var godkjent.

I møtet ble det ferdige systemet vist frem, og oppdragsgiver ga uttrykk for at systemet var tilfredsstillende og godkjente det. Deretter ble systemet overført til oppdragsgivers GitHub. Ignist ble oppfordret av studentene til å komme med tilbakemeldinger, kommentarer og konstruktiv kritikk dersom de hadde noe. Per dags dato har vi ikke mottatt noen tilbakemelding av slik karakter fra Ignist, men de har derimot uttalt at de ønsker å ta i bruk systemet og videreutvikle det, noe som tyder på at løsningen er akseptert og kan ha nytte for dem.

5.5 Samsvar med testplanen

Det er tre større avvik fra testplanen. Disse er at integrasjonstest ikke ble utført, ikke alle planlagte testobjekter ble testet under enhetstestene og systemtestene ble gjennomført manuelt i stedet for automatisert.

5.5.1 Manglende testdekning

Planen var å enhetsteste alle modellene. "UserModel" og "UserUpdateModel" ble ikke testet, som følge av for lite tid. Før enhetstestene ble igangsatt, ble det bestemt at kontrollerne ble hovedfokuset for testingen, da disse inneholder det meste av logikken i systemet og det derfor er her det er størst sannsynlighet for feil. Testing av alle modeller skjer også indirekte, da modellene, inkludert UserModel og UserUpdateModel, benyttes i testene for kontrollerne.

5.5.2 Ikke utført integrasjonstest

Det var planlagt å utføre tester på alle de fire overordnede testnivåene, enhetstest, integrasjonstest, systemtest og akseptansetest. Integrasjonstest av systemets komponenter ble derimot ikke utført. Integrasjonstester utføres for å undersøke om systemers komponenter fungerer sammen som forventet, og bidrar dermed til systemets pålitelighet (NTB & ISTQB, 2019, s. 30). Slike tester er derfor svært nyttige å gjennomføre.

Enhetstestene tok lenger tid enn forventet og på grunn av manglende ressurser, ble det vanskelig å gjennomføre alle de planlagte testene. Under planlegging av prosjektet og testingen ble det satt en frist for å overlevere webapplikasjonen til Ignist i slutten av mai, uke 20. I forbindelse med arbeidet av poster til IT-expo, med frist 1. mai, fant studentene ut at webapplikasjonen burde være så og si ferdig innen da, for å kunne vise til prosjektresultater på posteren. Dette medførte at tid til testing ble kortere enn først planlagt. Det ble dermed bestemt at gjenværende tid gikk til systemtester, slik at systemet kunne testes i sin helhet før levering til Ignist.

5.5.3 Automatisering av systemtester

Systemtester kan automatiseres ved å benytte programvare for automatisering av tester, og det var i utgangspunktet planlagt å bruke et slikt verktøy. Innledningsvis ble det derfor testet et par testscenarior i både Selenium og Katalon, som er eksempler på slike testverktøy. Det er fordeler og ulemper ved både manuell og automatisert testing. Automatisert testing kan være tidsbesparende ved større systemer. Dersom testene må gjentas igjen og igjen, vil automatisering også være kostnadseffektivt. Automatiserte tester er skalerbart der som webapplikasjonen utvides og får ny funksjonalitet. Menneskelige feil vil alltid oppstå, og ved å automatisere tester kan man minske sannsynlighet for slike glipper (GeeksforGeeks, 2023).

Webapplikasjonen som skulle testes er derimot et lite komplekst, mindre system. Manuell testing kunne derfor være en god tilnærming. Manuell testing krever lite oppsett og kan derfor være tidsbesparende. I tillegg kan menneskelig innsikt under testingen gjøre det enklere å legge merke til små detaljer og uventet adferd under testingen. Manuell testing kan være nyttig for å vurdere brukeropplevelse, da mennesker er bedre egnet enn maskiner og script til å bedømme hvordan opplevelsen av et system og hvorvidt det er intuitivt (McMillan, 2023).

Med grunnlag i disse argumentene, falt valget på manuell testing i denne omgang. Dersom webapplikasjonen etter hvert skal skaleres og mer funksjonalitet legges til, kan det hende at testene bør automatiseres i fremtiden.

5.6 Innsikt etter endt testing

Testing viser tilstedeværelse av feil, men kan ikke bevise deres travær. Man kan heller ikke teste absolutt alt, og et system vil aldri kunne testes fullstendig (NTB & ISTQB, 2019, s. 15). I tillegg til testene som er utført, bør et system også gjennomgå andre tester, som for eksempel sikkerhets- og ytelsestest, før det settes i drift. Det var svært få feil som ble oppdaget av enhetstestene. Dette betyr ikke at det ikke er noen feil i koden. Enhetstestene

antyder derimot at koden fungerer som tiltenkt og at logikken er korrekt. En test case fra systemtesten, id 1.20, avdekket en feil som viser en feil i logikk i koden. Det var mulig å endre en e-post-adresse hos en allerede registrert bruker til å være en e-post som ikke er i gyldig e-post-format. Dette kunne vært oppdaget allerede i enhetstestene, men ble ikke fanget opp av de utførte testene. Dette illustrerer viktigheten av å utføre flere tester på ulike testnivåer.

Enhetstestene ble utført av et prosjektmedlem som ikke har vært med på å utvikle koden. Dette er noe ukonvensjonelt, da enhetstester gjerne gjennomføres av utviklerne selv (NTB & ISTQB, 2019, s. 29). Men på grunn av dette, ga enhetstestene også innsikt i at koden var forståelig for andre enn utviklerne selv. Dette er et pluss, da koden og systemet skal overleveres til Ignist og vedlikeholdes og videreutvikles av andre personer enn de som først utviklet det.

Under systemtestene oppstod det utfordringer med å teste kravet om at “UX-design og brukergrensesnittet skal være intuitivt og brukervennlig”. Hvordan definerer man og måler brukervennlighet? Det ville muligens vært enklere om kravspesifikasjonen inneholdt mer spesifikke kriterier for brukervennlighet, for eksempel “Webapplikasjonen skal ha responsiv design”. Noen av kravene refererer dog direkte til brukervennlighet, som kravet om at webapplikasjonen skal ha en søkefunksjon.

For å teste brukervennlighet, ble det utarbeidet ulike test cases. Eksempler på dette er at administrator skal kunne se hva slags type rolle en bruker har, samt at man ikke skal kunne endre en e-post-adresse til en ugyldig verdi. Det er også implementert sjekker etter tilbakemeldinger på brukernes handlinger. Brukervennlighet er også testet i tilgjengelighetstesten, hvor kontraster og fargesjekk er testet for å se at nettsiden er lett leserlig.

Testing av brukervennlighet ble kun utført av testere fra studentgruppen og via gjennomgang av webapplikasjonen av Ignist. For å ytterlig sikre brukervennlighet og at systemet er intuitivt, burde webapplikasjonen også ha vært testet av reelle brukere. Ved å involvere eksterne brukere i testingen, vil man kunne få verdifulle tilbakemeldinger og perspektiver om hvordan webapplikasjonen oppleves av de som faktisk skal bruke den. Utenforstående brukere kan belyse problemer som ikke er synlige for de som har vært dypt involvert i utviklingen og har god kjennskap til applikasjonen (Foss-Pedersen, 2017).

6 Produktdokumentasjon

I dette kapittelet presenteres webapplikasjonen, Brannhjelp, og dens formål. Kapittelet beskriver webapplikasjonens arkitektur, sentrale datastrukturer og hoved- og del-funksjoner i løsningen. Til slutt følger en gjennomgang av webapplikasjonens samsvar med kravspesifikasjonen.

6.1 Introduksjon av løsningen

Brannhjelp er en plattform spesielt designet for ingeniører som søker informasjon om spesifikke krav til ulike bygningstyper. Gjennom plattformen tilbys tilgang til fagartikler som publiseres av Ignist. Disse inneholder detaljerte kravspesifikasjoner som er relevante for brannsikkerhet. Hovedmålet med applikasjonen er å tilby en pålitelig informasjonskilde for ingeniører som trenger klargøring rundt tekniske krav eller ønsker å utvide sin kunnskap på feltet.

Applikasjonen tillater brukere å få tilgang til å lese og utforske en rekke artikler relatert til brannsikkerhet. For å få full tilgang må brukere opprette en konto. Administratorer av applikasjonen har utvidede rettigheter, inkludert muligheten til å legge til, redigere og slette artikler. De kan også opprette underartikler tilknyttet hovedartiklene for å gi ytterligere detaljer. Videre har administratorer kapasitet til å oppdatere brukerinformasjon og justere brukernes roller i systemet.

6.2 Løsningens arkitektur

Webapplikasjonen er inndelt i to distinkte applikasjonskomponenter, frontend og backend, og har en skybasert database som er hostet i Microsoft Azure. Frontend for applikasjonen er statisk, mens backend fungerer som et API som frontend er avhengig av. Dette er en såkalt “headless” backend. En headless-arkitektur er et utviklingskonsept hvor frontend skiller fra backend (Alokai, 2023).

Frontend vil si programvaren som er ansvarlig for implementeringen av det grafiske brukergrensesnittet. Hovedmålet er å tilrettelegge for brukerinteraksjon og visuell presentasjon. Dette innebærer utforming av kode som direkte påvirker de visuelle elementene presentert på brukerens skjerm, samt styring av brukerens interaksjoner med disse elementene (Granevang, 2020). For støtte til frontend brukes diverse biblioteker, som er oppført i filen “package.json”,

Backend-utvikling handler om det som skjer på serveren ved utvikling av webapplikasjoner. Den primære funksjonen er å motta, behandle, analysere og sende data. Backend-arbeidet inkluderer styring av viktig forretningslogikk, konfigurering av serveren og håndtering av brukernes tilgangskontroll gjennom autentisering og autorisering (Cloudworks, u.å.). For backend er støttebibliotekene angitt som “NuGet”-pakker, som kan vises i “Dependencies”-mappen.

En headless backend bringer med seg flere fordeler, blant annet økt smidighet og fleksibilitet i utviklingsprosessen. Ettersom frontend og backend er atskilt, har utviklerne full frihet til å benytte ulike teknologier og rammeverk på frontend, uten at dette påvirker backend. Dette reduserer både utviklingstiden og eventuelle kostnader ved implementering av nye funksjoner i fremtiden (Strapi, u.å.). Headless arkitektur støtter Single Responsibility Principle (S-prinsippet) i SOLID-prinsippene som Abba (2022) beskriver, ved å skille funksjonallitet og logikk fra design og presentasjon. Backend-systemet fokuserer kun på data- og logikkhåndtering, mens frontend tar seg av presenteringen. Dette muliggjør minimal kommunikasjon mellom lagene, forutsatt at et klart definert API tilgjengelig. En slik klar ansvarsfordeling, forenkler arbeidet med å vedlikeholde og videreutvikle delene separat, uten at en endring i for eksempel frontend påvirker backend. En headless arkitektur bidrar altså til en mer strukturert og håndterbar kodebase, som er kjernen i S-prinsippet.

6.3 Frontend

Programmets brukeropplevelse er direkte relatert til frontend-komponentene. Designerens oppgave er å planlegge hvordan brukerinteraksjonene skal foregå på en måte som er både intuitiv og engasjerende. Utviklerne har ansvaret for å implementere dette designet, og bringe det til liv gjennom frontend-programmering.

6.3.1 Single Page Application

Frontend-komponenten i webapplikasjonen er implementert som en Single Page Application (SPA). En SPA er en nettapplikasjon som dynamisk oppdaterer den gjeldende nettsiden med data fra serveren, noe som eliminerer behovet for tradisjonelt, fullstendig sideskift gjennom nettleseren. Dette er en funksjonalitet som React, primærbibliotek benyttet i prosjektet, tilbyr (Lawson, 2022).

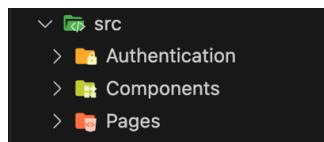
SPA gir en kortere responstid fordi det flytter en del av dataprosesseringen fra server til nettleseren. Ved bruk av SPA, lastes hele applikasjonen som regel kun én gang ved oppstart. Deretter kan brukeren navigere rundt på siden uten at hele siden må lastes ned på nytt, ettersom store deler av nettsiden allerede er lastet i nettleseren. Brukeren vil dermed oppleve kortere ventetid (Portney, 2018). Hastigheten ved sidelastning kan være kritisk for brukeropplevelsen og har vist seg å ha en direkte innvirkning på forretningsresultater. Sider som tar lengre enn 200 millisekunder å laste, kan påvirke brukeropplevelsen negativt (Brutlag, 2009).

På en annen side kan SPAs møte utfordringer relatert til søkemotoroptimalisering (SEO). En faktor som søkemotorer tar i betrakting under indeksering, er antall sider på et nettsted. Gitt at en SPA kun laster en enkelt side, kan dette potensielt medføre negative konsekvenser for nettsidens synlighet i søkemotorer (Portney, 2018).

Av hensyn til både fordelene og ulempene, samt oppdragsgivers preferanser, ble det besluttet å utvikle en SPA med React-rammeverket.

6.3.2 Strukturering av frontend-kode

Frontend-kildekoden er strukturert med tanke på “Separation of Concerns”-prinsippet (SoC) med hensikt å dele komplekse systemer inn i mindre, håndterbare deler (GeeksforGeeks, 2024). Kildekoden er inndelt i tre distinkte lag: “Authentication”, “Components” og “Pages”, se figur 17. Authentication-laget omfatter alle aspekter knyttet til autentisering. Components består av gjenbrukbare kodefragmenter som er spesifikt utviklet for applikasjonen. Pages-laget inneholder sammensetninger av flere komponenter som danner de forskjellige sidene som presenteres for brukeren.

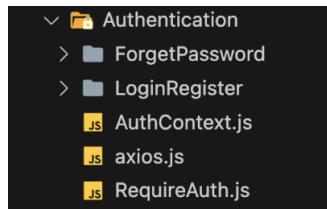


Figur 17: Arkitekturen til frontend-koden

6.3.2.1 Autentisering (Authentication)

“Autentisering”-mappen er en sentral del av frontend-strukturen og inneholder all funksjonalitet relatert til autentisering, se figur 18.

Inne i autentiseringslaget er det to undermapper:



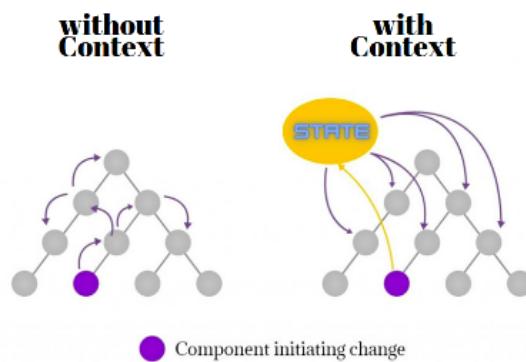
Figur 18: Authentication-mappen

- ForgetPassword som håndterer situasjoner hvor brukeren har glemt sitt passord
- LoginRegister som inneholder filer og logikk relatert til brukerens innloggings- og registreringsprosedyrer

På filnivå, består autentiseringslaget av tre JavaScript-filer:

- AuthContext.js som sentraliserer arkitekturen for brukerautentisering
- RequireAuth.js som gir en beskyttelsesmekanisme i form av tilgangskontroll
- Axios.js som håndterer HTTP-forespørslar for autentiseringsrelaterte forespørslar

I React-applikasjoner er det ofte nødvendig å dele data eller tilstand mellom flere komponenter. En tradisjonell metode for å gjøre dette er “props drilling”, som sender data gjennom lag av komponenter. Dette kan bli komplisert og uoversiktig. For å unngå dette benyttes “CreateContext” for å opprette en global tilstand som kan aksesseres av komponenter på forskjellige nivåer i komponenttreet, som vist i figur 19. Det er som å opprette en tilgjengelig “kilde” av data som enhver komponent kan nå direkte, uten å måtte motta det eksplisitt fra sine foreldre. Slik kan data nås direkte uten behovet for props. Dette lar andre komponenter i applikasjonen abonnere på autentiseringsstatusen og motta oppdateringer automatisk, uten behovet for å få data tilsendt via props fra en foreldrekomponent. Det forenkler flyten av data og gjør koden mer lesbar og lettere å vedlikeholde.



Figur 19: Endringsformidling i et komponenttre uten og med kontekst (Sharmaa, 2023)

“AuthProvider”-komponenten inneholder kritisk logikk for autentiseringsprosesser, inkludert funksjoner for pålogging, utlogging og lagring av brukerens tilstandsinformasjon. Ved vellykket pålogging, initieres en forespørsel via Axios for å motta en JWT-token som lagres sikkerhetsmessig i informasjonskapsler. JWT-tokenet dekodes deretter for å utvinne og tilstandslagre brukerens identitet og rettigheter.

For å opprettholde en stabil og sikker brukerinnlogging, overvåker “useEffect” konstant tokenets gyldighet. I tillegg gir “useAuth” tilgang til AuthContext, som gir et enklere grensesnitt for autentiseringsrelaterte oppgaver og tilgang til brukerdata. Dette styrker prinsippet om modularitet og gjenbruk i designet av applikasjonen.

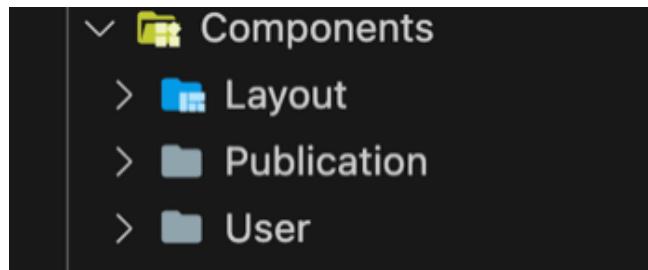
“RequireAuth”-komponenten fungerer som en beskyttelsesmekanisme for å håndheve tilgangskontroll til visse deler av applikasjonen. Den anvender en kombinasjon av JWT-token verifisering og rollebasert tilgangsstyring. RequireAuth validerer JWT-tokenet og sjekker brukerens rolle for å avgjøre om tilgang skal gis. Dersom tokenet er gyldig og brukeren har den nødvendige rollen, vil brukeren få tilgang til applikasjonens beskyttede deler.

I tilfeller der brukeren ikke er autorisert, enten på grunn av manglende token, en utgått token, eller manglende nødvendige roller, vil komponenten omdirigere brukeren til innloggingssiden. Omdirigeringen inkluderer også brukerens nåværende posisjon, (den de forsøkte å få tilgang til), slik at den kan brukes etter en vellykket innlogging for å sende brukeren tilbake til den opprinnelig ønskede siden.

Dette bidrar til å sikre at kun autoriserte brukere har tilgang til sensitivt innhold eller funksjonalitet, samtidig som det gir en brukervennlig tilbakeføringsflyt etter autentisering.

6.3.2.2 Komponenter (*Components*)

“Components”-mappen er en sentral del av frontend-strukturen og består av gjenbrukbare kodefragmenter i applikasjonen, se figur 20.



Figur 20: Compoenets-mappen

Innenfor “Layout”-underkatalogen er det en samling av komponenter som er essensielle for å bestemme applikasjonens visuelle arkitektur. Disse komponentene, ofte referert til som “skallkomponenter”, er avgjørende for å definere og vedlikeholde den strukturelle konsistensen på tvers av applikasjonens grensesnitt. De inkluderer, men er ikke begrenset til, komponenter som bygger opp sidens fundamentale layout, inkludert navigasjonsmenyer og footer-elementer.

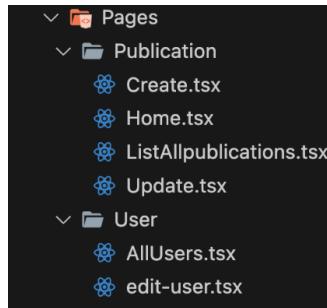
I “Publication”-underkatalogen, finner man en samling av komponenter og funksjoner som er spesifikt tilpasset for å håndtere produksjon, visning og distribusjon av publisert innhold. Dette kan omfatte redigeringsverktøy for innholdsskapning, skjemaer for datainnsamling, samt presentasjonskomponenter for å dynamisk vise frem innhold.

Videre er “User”-kategorien utstyrt med komponenter designet for å fremme brukerinteraksjon og personalisering. Dette omfatter mekanismer for brukerautentisering, profiladministrasjon og personalisering, som tillater en sikker og intuitiv navigering og modifikasjon av brukerprofiler.

Innenfor dette domenet finner man også funksjoner som muliggjør visning av alle brukere samt redigeringsfunksjonalitet for deres personlige informasjon. Det tilretteslegges for interaktiviteten gjennom intuitive grensesnitt som tillater autoriserte brukere å administrere sine brukerkontoer, herunder oppdatering av personopplysninger og administrering av tilgangsrettigheter, i et system som ivaretar sikkerhet og personvern.

6.3.2.3 Sider (*Pages*)

“Pages”-mappen inneholder de forskjellige visningene som er tilgjengelige i brukerapplikasjonen, se figur 21. Denne tilnærmingen reflekterer et komponentbasert rammeverk hvor hver side er konstruert av distinste komponenter. Denne metodikken forenkler ikke bare implementeringen av endringer, men forenkler også prosessen med å reorganisere komponenter for å oppnå optimal responsivitet.



Figur 21: Pages-mappen

Undermappene “Publication” og “User” viser en tydelig modular inndeling, hvor hver undermappe inneholder flere TypeScript React-komponenter (“.tsx”-filer). Disse komponentene håndterer opprettelse, oppdatering, visning og redigering av publikasjoner og brukerprofiler. Dette bidrar til å organisere funksjonaliteten på en måte som gjør koden enklere å vedlikeholde og utvide.

Videre reflekterer bruk av TypeScript et fokus på type-sikkerhet og robusthet i koden. Denne tilnærmingen tilrettelegger for en mer systematisk og intuitiv utviklingsprosess som er spesielt viktig i større og komplekse systemer.

6.4 Backend

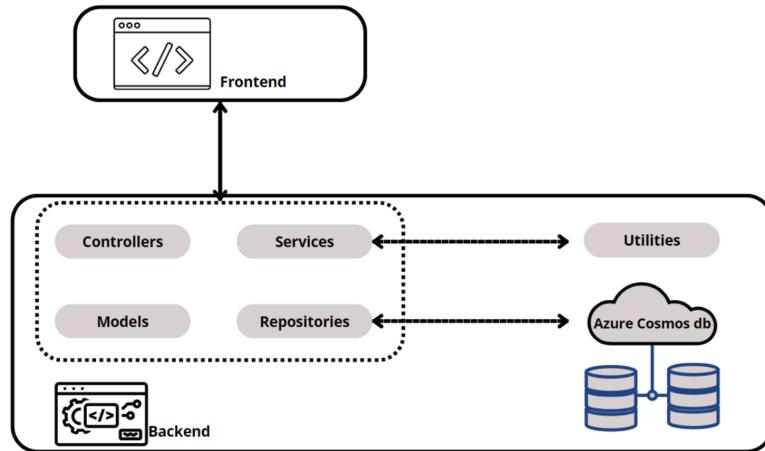
I praksis er backend det som arbeider i det skjulte for å prosessere og overføre data til frontend. Frontend delen av en applikasjon bruker denne dataen til å tilby brukeren en intuitiv og velfungerende brukeropplevelse, ofte formidlet gjennom API-er. I Brannhjelp-prosjektet er backend bygget opp av sentrale komponenter som er avgjørende for applikasjonens struktur. Disse komponentene sikrer at applikasjonen er stabil og gir rom for enkel implementering av fremtidige oppdateringer. Disse komponentene inkluderer kontrollere (controllers), oppbevaringssteder (repositories), modeller (models), tjenester (services) og loggføringer (logs).

Til sammen jobber disse komponentene sammen med databasen for å sikre at alt fungerer smidig og sikkert, slik at applikasjonen kan vokse og endre seg etter behov. Figur 22 illustrerer hvordan frontend kommuniserer med backend gjennom kontrollere, som deretter bruker tjenester og oppbevaringssteder for å prosessere og lagre data i databasen.

6.4.1 Kontrollere (*Controllers*)

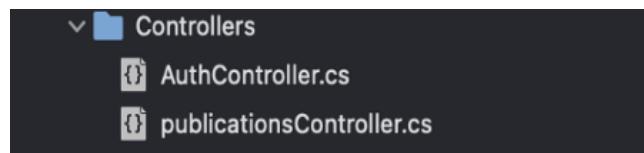
I moderne webutvikling spiller Web API-kontrolleren en nøkkelrolle i håndteringen av HTTP-forespørsler, styring av applikasjonsflyten og avgjørelsen om hva slags respons som skal sendes tilbake til klienten. En Web API-kontroller, som typisk er organisert i prosjekts Controllers-mappe, er primært ansvarlig for å definere API-endepunkter. Disse endepunktene tillater klientapplikasjoner å utføre operasjoner gjennom HTTP-forespørsler.

I kontrollere benyttes “Authorize”-attributter og rollebasert autentisering for å beskytte hvilke rettigheter en



Figur 22: Kommunikasjon mellom frontend- og backend-komponenten

bruker eller endepunkt skal ha. I Brannhjelp-prosjektet er det to sentrale kontrollere, som vist i figur 23. Den ene er “AuthController.cs”, som håndterer brukeradministrasjon som registrering, innlogging og passordstyring. Den andre sentrale kontrolleren er “PublicationsController.cs”, som administrerer operasjoner relatert til publikasjoner.



Figur 23: Kontroller-mappen som definerer API-endepunktene

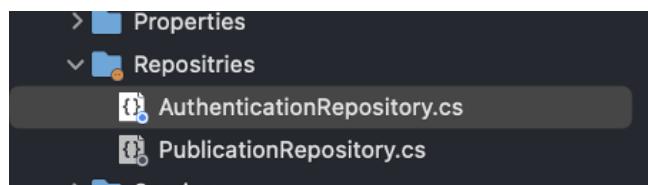
Kontrollerklassene inneholder metoder som definerer API-endepunkter, slik som `HttpPost` ("register"), som er ansvarlige for registrering av nye brukere. Disse metodene tilbyr en klar og logisk tilnærming til håndtering av klientforespørsler, inkludert aspekter som validering og feilhåndtering. Dette understøtter prinsippet om SoC, der kontrollerne fungerer som et mellomledd mellom frontend-logikk og service-logikk (GeeksforGeeks, 2024).

Når en klient sender en forespørsel fra frontend, for eksempel ved å trykke på en "register"-knapp, sendes et signal via en API-forespørsel til den relevante kontrolleren på serveren. Kontrolleren mottar forespørselen og sender den videre til riktig serviceklasse, som håndterer den underliggende logikken. Serviceklassen utfører nødvendige operasjoner, som kan inkludere interaksjoner med databaser gjennom et "repository"-lag, og returnerer resultatet tilbake til kontrolleren. Kontrolleren mottar svaret fra servicelaget og sender en passende respons tilbake til frontend, enten det er en bekreftelse på vellykket handling eller en feilmelding. Denne flyten sikrer en klar separasjon av ansvar og bidrar til å opprettholde systemets organisasjon og effektivitet.

6.4.2 Oppbevaringssteder (*Repositories*)

Repository-laget er svært viktig ettersom det håndterer logikken som trengs for å få tilgang til data. Som tidligere nevnt, er det essensielt å skille ulike ansvarsområder for å forbedre systemets evne til å håndtere vekst og vedlikehold. All logikk knyttet til tjenester, blir håndtert av dette laget. Repository-laget jobber tett med databasen for å lagre og hente data og for å utføre grunnleggende databaseoperasjoner som å opprette, lese, oppdatere og slette data. Denne funksjonen er viktig fordi den forenkler databasens interaksjoner og sikrer at systemet fungerer sømløst (Kumar & Simic, 2024).

Bruk av repository-mønsteret er et kraftfullt verktøy i C# og .NET-utvikling for å oppnå en klar separasjon av ansvarsområder, samt for å bygge applikasjoner som er skalerbare og kan vedlikeholdes (Kara, 2023). I Brannhjelp-prosjektet er hvert repository nøy utformet for å fremme en tydelig ansvarsdeling, noe som forenkler vedlikeholdet og forbedrer applikasjonens evne til å tilpasse seg endringer i datalagringsmekanismer. I Brannhjelp-prosjektet var det nødvendig med to type dataspøringer mot databasen, som vist i figur 24.



Figur 24: Repository-klassene som forenkler spørninger mellom controllere og databasen

6.4.2.1 AuthenticationRepository

Denne klassen utgjør en kritisk komponent i applikasjonens Data Access Layer (DAL), og spiller en sentral rolle i håndteringen av brukeridentifikasjon og tilgangskontroll. `AuthenticationRepository` styrer databaseoperasjoner for `AuthController` i Cosmos DB, se dens kodeoversikt i figur 25. Sikkerheten og integriteten til applikasjonen er avhengig av nøyaktig og sikker håndtering av autentiseringsdata. "AuthenticationServices.cs" spiller en nøkkelrolle i dette ved å støtte nødvendige operasjoner for å opprette, oppdatere og lese brukerdata. Gjennom nøy utformede spøringer, som for eksempel å hente brukerinformasjon basert på unike identifikatorer, sikrer klassen at brukerdata kan håndteres effektivt og sikkert. Disse operasjonene er avgjørende for å opprettholde brukernes integritet og sikkerhet i systemet.

```

public class CosmosDbService : ICosmosDbService
{
    private readonly CosmosClient _cosmosClient;
    private readonly Container _container;
    private readonly PasswordHelper _passwordHelper;
    private readonly JwtTokenService _jwtTokenService;
    private readonly IEmailService _emailService;

    public CosmosDbService(CosmosClient cosmosClient, IConfiguration configuration, PasswordHelper passwordHelper, JwtTokenService jwtTokenService)
    {
        // Henter en bruker basert på email, som er partition key
        public async Task<User> GetUserByEmailAsync(string email)...
        public async Task<User> GetUserByIdAsync(string userId)...
        public async Task<string> RegisterUserAsync(RegisterModel registerModel)...
        public async Task<string> LoginUserAsync(LoginModel loginModel)...
        public async Task<string> HandleForgotPasswordAsync(string email)...
        public async Task<string> UpdateUserPasswordAsync(string userEmail, string oldPassword, string newPassword, string confirmPassword)...
        public async Task<ServiceResponse> UpdateUserAsync(string userId, UserUpdateModel updateModel)...
        // metoden for å lage ny bruker
        public async Task AddUserAsync(User user)...
    }
}

```

Figur 25: AuthenticationRepository styrer databaseoperasjoner for AuthController i Cosmos DB

AuthenticationServices.cs fungerer som en kritisk forbindelse mellom brukergrensesnittet og data-laget i applikasjonen. Den muliggjør utførelsen av komplekse databasemanipulasjoner via et rent og tydelig definert API. Modulariseringen av datatilgang og logikk bidrar til å forbedre applikasjonens vedlikeholdbarhet og gjør det lettere å implementere tilpasninger som møter nye brukerbehov og teknologiske utviklinger.

6.4.2.2 PublicationRepository

I Brannhjelp-applikasjonen håndterer “PublicationsRepository” publikasjoner ved hjelp av Azure Cosmos DB. Denne klassen utfører CRUD-operasjoner som inkluderer henting, oppretting, oppdatering og sletting av publikasjoner basert på brukerens ID. Grensesnittet “IPublicationsRepository” definerer metodene for interaksjon med publikasjonsdata, noe som sikrer uavhengighet fra underliggende implementasjonsdetaljer. Dette fremmer løs kobling og forbedrer applikasjonens testbarhet. Figur 26 illustrerer metodene som er definert i publikasjon.

```

public class PublicationsRepository : IPublicationsRepository
{
    private readonly CosmosClient _cosmosClient;
    private readonly IConfiguration _configuration;
    private readonly Container _publicationContainer;

    public PublicationsRepository(CosmosClient cosmosClient, IConfiguration configuration)
    {
        _cosmosClient = cosmosClient;
        _configuration = configuration;
        _publicationContainer = new Container("Publications");
    }

    public async Task<IEnumerable<Publication>> GetAllPublicationsAsync()
    {
        var publications = await _publicationContainer.GetItemCollection("Publications").Query().ToEnumerableAsync();
        return publications;
    }

    public void InitializeHierarchy(IEnumerable<Publication> publications, HashSet<string> processedIds)
    {
        foreach (var publication in publications)
        {
            if (!processedIds.Contains(publication.Id))
            {
                publication.HierarchyLevel = 1;
                publication.ParentId = null;
                processedIds.Add(publication.Id);

                foreach (var child in publication.Children)
                {
                    child.HierarchyLevel = 2;
                    child.ParentId = publication.Id;
                    child.Processed = false;
                    processedIds.Add(child.Id);
                }
            }
        }
    }

    public async Task<Publication> GetPublicationByIdAsync(string id)
    {
        var publication = await _publicationContainer.GetItemCollection("Publications").Query().Where(p => p.Id == id).ToEnumerableAsync();
        return publication.FirstOrDefault();
    }

    public async Task<Publication> GetLastPublicationAsync()
    {
        var publications = await _publicationContainer.GetItemCollection("Publications").Query().ToEnumerableAsync();
        return publications.OrderByDescending(p => p.Id).First();
    }

    public async Task AddPublicationAsync(Publication publication)
    {
        await _publicationContainer.CreateItemAsync(publication);
    }

    public async Task UpdatePublicationAsync(Publication publication)
    {
        await _publicationContainer.ReplaceItemAsync(publication, publication.Id);
    }

    public async Task DeletePublicationAsync(string id, string UserId)
    {
        var publication = await GetPublicationByIdAsync(id);
        publication.UserId = UserId;
        publication.DeletedAt = DateTime.UtcNow;
        await UpdatePublicationAsync(publication);
    }

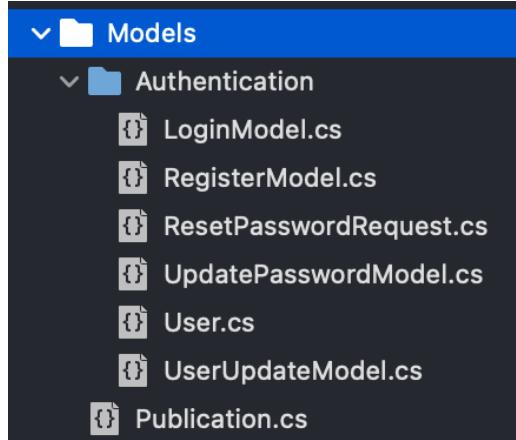
    public async Task<Publication> GetPublicationByIdAsync(string id, string UserId)
    {
        var publication = await _publicationContainer.GetItemCollection("Publications").Query().Where(p => p.Id == id).ToEnumerableAsync();
        return publication.FirstOrDefault();
    }
}

```

Figur 26: Kodeoversikt over PublicationsRepository, som styrer databaseoperasjoner for publikasjoner i Cosmos DB

6.4.3 Modeller (*Models*)

Innenfor mappen “Models” i applikasjonen, er det en rekke klasser som spiller en essensiell rolle i håndteringen av brukerinteraksjoner. Disse klassene er avgjørende for både autentisering og effektiv datahåndtering, og er nøyde organisert for å speile deres funksjonalitet og anvendelse i applikasjonen. Figur 27 viser de ulike klassene som finnes i modellmappen. Dette inkluderer modeller for autentisering og brukerhåndtering, som støtter sikker brukerregistrering, innlogging og oppdatering av brukerinformasjon.



Figur 27: Oversikt over 'Models'-mappen som viser datastrukturer brukt i backend

Blant kjernekomponentene er "User.cs", som lagrer brukerinformasjon inkludert identifikatorer, brukernavn, etternavn, e-postadresse og en hashet versjon av brukerens passord. Klassene inneholder også funksjonalitet for brukerroller og mekanismer for å tilbakestille passord, noe som understreker fokus på sikkerhet og brukervennlighet. Nedenfor, i figur 28, presenteres "User.cs"-klassens attributter som definerer en brukers datastruktur.

```
using System;
using Newtonsoft.Json;

namespace Ignist.Models
{
    public class User
    {
        [JsonProperty("id")]
        public string Id { get; set; } = Guid.NewGuid().ToString();

        [JsonProperty("UserName")]
        public string UserName { get; set; }

        [JsonProperty("LastName")]
        public string LastName { get; set; }

        [JsonProperty("email")]
        public string Email { get; set; }

        [JsonProperty("passwordHash")]
        public string PasswordHash { get; set; }

        [JsonProperty("Role")]
        public string Role { get; set; } = "Normal"; // Default role

        [JsonProperty("PasswordResetCode")]
        public string PasswordResetCode { get; set; }

        [JsonProperty("PasswordResetCodeExpires")]
        public DateTime PasswordResetCodeExpires { get; set; }
    }
}
```

Figur 28: 'User.cs'-klassen definerer brukerens datastruktur

For autentiseringsprosesser benyttes "LoginModel.cs", som krever at brukeren angir både e-postadresse og passord ved innlogging. Dette bidrar til å sikre at sikkerhetskrav og valideringsprosedyrer er oppfylt under innlogging. Sikkerheten forsterkes ytterligere gjennom måten passord og sensitive brukerdata behandles på. Avanserte krypteringsteknikker og sikre lagringsmetoder sørger for at brukerinformasjon er skjernet mot uautorisert tilgang.

"Publication.cs"-klassen er designet for å representere artiklene som publiseres på Brannhjelp. Denne klassen lagrer informasjon som tittel, innhold, og tidspunkt for opprettelse og oppdateringer. Figur 29 illustrerer hvordan klassen benytter JSON-annotasjoner for å sikre nøyaktig serialisering og deserialisering i samarbeid med databasen. Ved bruk av attributter som `JsonProperty`, sikres korrekt mapping av data, noe som er essensielt for å opprettholde dataintegritet og effektiv databehandling. Videre håndterer klassen relasjoner til andre publikasjoner gjennom partisjonsnøkkelen "UserId" og feltet "ParentId", som er kjernelementer i strukturen for

datahåndtering og datahierarki. Bruken av “Newtonsoft.Json” muliggjør en problemfri integrasjon med Cosmos DB, spesielt gjennom tilpasset serialisering av data ved hjelp av “JsonProperty”-attributter. Slike attributter, i kombinasjon med partisjonsnøkler og identifikatorer, er viktige for rask og effektiv databehandling i et distribuert databasesystem. Disse elementene bidrar til å hente og endre data raskt over flere database-setjenere.

```

using System;
using System.ComponentModel.DataAnnotations;
using Newtonsoft.Json;

namespace Ignist.Models
{
    public class Publication
    {
        [JsonProperty(PropertyName = "id")]
        public string Id { get; set; }

        [Required]
        [JsonProperty(PropertyName = "title")]
        public string Title { get; set; }

        [Required]
        [JsonProperty(PropertyName = "content")]
        public string Content { get; set; }

        [JsonProperty(PropertyName = "createdAt")]
        public DateTime CreatedAt { get; set; }

        [JsonProperty(PropertyName = "updatedAt")]
        public DateTime UpdatedAt { get; set; }

        [JsonProperty(PropertyName = "userId")]
        public string UserId { get; set; }

        [JsonProperty(PropertyName = "parentId")]
        public string ParentId { get; set; }

        public List<Publication> ChildPublications
        { get; set; } = new List<Publication>();
    }
}

```

Figur 29: Attributtene som finnes i ’Publication’-klassen

6.4.4 Tjenester (*Services*)

I prosjektet utgjør tjenestelaget “Service Layer” kjernen i backend-logikken og er avgjørende for effektiv håndtering av data og logikk. Dette laget består av flere serviceklasser hvor hver klasse er designet for å fungere gjennom klart definerte grensesnitt, kjent som “service interfaces”. Metodene som er definert i disse grensesnittene, må implementeres i de tilhørende repository-klassene. Grensesnitt representerer en kraftig funksjon i .NET Core-programmering som fremmer utvikling av kode som er mer vedlikeholdbar, testbar og gjenbrukbar. Ved å anvende grensesnitt, kan utviklere separere implementeringen av kode fra dens bruk, lette avhengighetsinnsjekasjon, og fremme kodegenbruk (Weerayut, 2023). Figur 30 viser ICosmosDbService-grensesnittet, som definerer alle nødvendige tjenester for brukerhåndtering.

```

5   using Ignist.Services;
6
7   namespace Ignist.Data.Services
8   {
9       public interface ICosmosDbService
10      {
11          Task<User> GetUserByEmailAsync(string email);
12          Task<User> GetUserByIdAsync(string userId);
13          Task AddUserAsync(User user);
14          Task UpdateUserAsync(User user);
15          Task<IEnumerable<User>> GetAllUsersAsync();
16          Task DeleteUserAsync(string email);
17          Task<string> RegisterUserAsync(RegisterModel registerModel);
18          Task<string> LoginUserAsync(LoginModel loginModel);
19          Task<string> HandleForgotPasswordAsync(string email);
20          Task<string> HandleResetPasswordAsync(ResetPasswordRequest request);
21          Task<string> UpdateUserPasswordAsync(string userEmail, string oldPassword, string newPassword, string confirmNewPassword);
22          Task<ServiceResponse> UpdateUserAsync(string currentEmail, UserUpdateModel updateModel);
23
24      }
25
26  }

```

Figur 30: Oversikt over 'Services'-mappen som viser IcosmosDbService-grensesnittet

Valget om å implementere servicegrensesnitt i prosjektet, er strategisk for å sikre konsistent og effektiv håndtering av applikasjonens funksjoner. Dette designvalget understøtter ikke bare kodegenbruk og forenkler vedlikeholdet, men det styrker også systemets evne til smidig å håndtere endringer i logikken uten behov for omfattende omstruktureringer. Ved å definere klare grensesnitt for alle serviceklassene, sikres en tydelig separasjon mellom de ulike lagene i applikasjonen, noe som resulterer i en robust arkitektur som er enklere å teste og feilsøke.

RESTful API benyttes for å tilrettelegge for kommunikasjon mellom tjenestelaget og andre komponenter i applikasjonen. Dette APIet håndterer HTTP-kall som er essensielle for å motta og svare på forespørsler fra klienter. Serviceklassene behandler disse forespørslene ved å validere og videreforside dem for databehandling. De tar seg av viktige oppgaver som brukerregistrering, innlogging, og profiloppdateringer. Dette forenkler administrasjonen og oppdateringen av applikasjonen over tid.

Tjenestelaget er derfor ikke bare hjertet i applikasjonens backend, men også en kritisk komponent som muliggjør en strukturert og effektiv implementering av nødvendige operasjoner og prosesser i henhold til kravspesifikasjonen.

6.4.5 Loggføring (Logs)

Logging er en avgjørende del av enhver applikasjon, og spiller en sentral rolle i montering, feilsøking og sikkerhet. Å føre nøyaktige og detaljerte logger, gir systemet innsikt i applikasjonens oppførsel over tid. Det kan være avgjørende for å diagnostisere og løse problemer. I Brannhjelp-applikasjonen registerets hendelser som systemfeil, unntak, sikkerhetsrelaterte hendelser og brukeraktiviteter. Denne informasjonen gir utviklere og systemadministratører verktøy for å raskt identifisere og rette opp i feil, forbedre applikasjonens ytelse, og forstå brukeratferd.

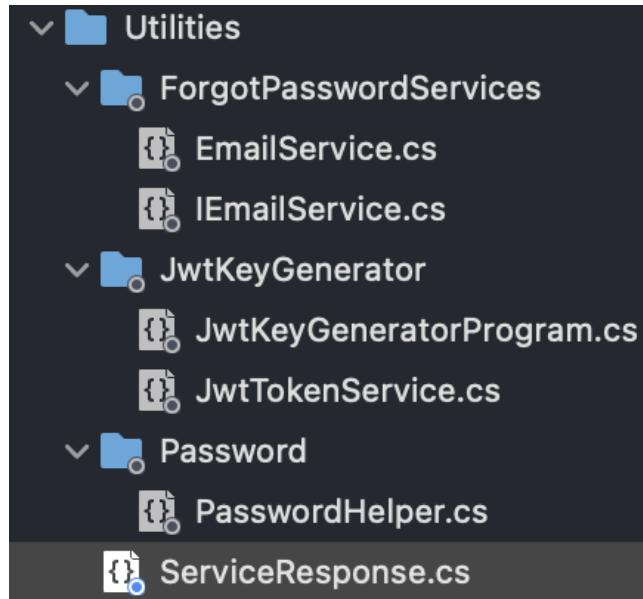
For loggføringsbehovene til applikasjonen har prosjektteamet implementert Serilog, en loggingplattform kjent for sin kompetanse innen strukturert logging. Denne teknologien tillater loggføring av data som strukturerte objekter i stedet for konvensjonelle tekstbaserte innlegg. Serilog muliggjør også en presis spesifisering av loggnivåer, fra detaljert "Debug"-logging, som er kritisk under utvikling og feilsøking, til "Information"-nivå for overvåkning av generell systemaktivitet, og videre til "Error" og "Fatal"-nivåer for umiddelbar varsling om kritiske systemhendelser. (Serilog, u.å.).

Serilog-biblioteket sikrer at loggføringen ikke bare er informativ og funksjonell for utviklingsteamet, men også at den er fininnstilt for å maksimere systemets ytelse og støtte vedvarende datalagring. Dette er fundamentalt for å garantere kontinuerlig overvåkning og vedlikehold av systemet på en effektiv måte. I en bredere kontekst, styrker bruken av Serilog det underliggende rammeverket for applikasjonsovervåkning, som er avgjørende for

pålitelig drift og kvalitetssikring.

6.4.6 Verktøy (*Utilities*)

I prosjektstrukturen, under mappen “Utilities”, er det flere hjelpeklasser som bidrar til å gjøre prosjektet forståelig og enkelt å vedlikeholde. Mappestrukturen med hjelpetjenester er vist i figur 31. En slik sentralisering av hjelpefunksjoner gir flere fordeler. Blant annet vil kall til disse funksjonene forenkles, som reduserer tid og kompleksitet ved å bruke dem. Denne strukturen bidrar også til en ryddigere kode, som gjør prosjektet enklere å vedlikeholde og videreutvikle.



Figur 31: 'Utilities'-mappen som inneholder hjelpetjenester og funksjonaliteter for systemet

“ForgotPasswordServices” tilbyr funksjonalitet for brukere som har glemt passordet sitt. Denne tjenesten bruker SendGrid for å sikre at e-poster for tilbakestilling av passord blir levert effektivt og pålitelig. SendGrid er valgt for sin evne til å håndtere store volumer av e-post og fordi det gir utviklere verktøy for enkel integrasjon i applikasjoner (SendGrid, u.å.). SendGrid var også kjent for oppdragsgiver fra før av. Når en bruker ber om å tilbakestille passord, sender systemet automatisk en e-post med en lenke eller kode som brukeren kan bruke for å opprette et nytt passord.

“JwtKeyGenerator”-mappen inneholder verktøy for å generere sikre nøkler som brukes til å signere og validere JSON Web Tokens (JWTs). Disse nøklene er avgjørende for å sikre at tokens er autentiske og uendret. Under innlogging genererer JWT en sikkerhetsalgoritme som holder brukerdata trygg, og for hver innlogging opprettes en ny unik hash for brukeren.

“Password”-mappen sikrer at alle passord som opprettes av brukere er unike og oppfyller bestemte krav som lengde, bruk av store bokstaver, og spesialtegn.

Til slutt er ServiceResponse-klassen som er designet for å standardisere svarene som sendes fra serveren til klientene. Denne klassen indikerer om en operasjon var vellykket og inkluderer et tilhørende meldingsfelt for å formidle ytterligere informasjon eller feilmeldinger. Ved å forenkle håndteringen av tilbakemeldinger på klientsiden, bidras det til en mer effektiv og sømløs interaksjon for brukerne, som igjen gir en mer brukervennlig opplevelse.

6.5 Sentrale datastrukturer

Systemet har implementert flere nøkkeldatastrukturer som er avgjørende for både funksjonaliteten og effektiviteten til systemet. Disse datastrukturene er spesifikt designet for å støtte komplekse operasjoner og sikre rask datahåndtering, noe som er essensielt for å møte tekniske krav og brukernes forventninger. Blant de sentrale datastrukturene finner vi objekter som "User" og "Publication", som begge spiller en viktig rolle i hvordan data organiseres og administreres innenfor applikasjonen.

User-klassen er strukturert for å håndtere detaljer relatert til brukeridentifikasjon og autentisering. Denne klassen inkluderer felt som "Id", "UserName", og "PasswordHash", som er avgjørende for sikker brukerhåndtering. Bruken av GUID for Id sikrer at hver bruker får en unik identifikator, noe som eliminerer risikoen for identifikasjonskonflikter i databasen.

Publication-klassen gjør det mulig å lagre og organisere informasjon relatert til innlegg eller artikler som brukerne publiserer. Denne klassen inkluderer en rekke felt som "Title", "Content" og "CreatedAt", samt en liste over "ChildPublications" som tillater hierarkisk organisering av innhold. Dette understøtter effektiv datahenting og manipulasjon, spesielt i scenarier hvor innhold må vises eller redigeres dynamisk.

Gjennom disse datastrukturene implementeres robuste løsninger for datahåndtering som er skalerbare og tilpassningsdyktige til endrede krav. Ved å velge gunstige datastrukturer og tilnæringer til dataorganisering, sikres det at applikasjonen ikke bare oppfyller nåværende behov, men også er forberedt på fremtidige utvidelser og forbedringer. Dette bidrar til at applikasjonen kan håndtere økt kompleksitet og datamengde, uten å påvirke ytelse eller brukervennlighet.

6.6 Hoveddeler i løsningen

Kjernen i systemet er innholdshåndtering, som tillater opprettelse, lagring og deling av digitalt innhold som artikler og lenker. Dette gir administratorer muligheten til å skape og formtere innhold. Innholdsredigeringsfunksjonen støtter oppdatering, sletting og redigering av eksisterende innhold.

En annen viktig del av systemet er håndtering av brukeradministrasjon og brukerautentisering, som er avgjørende for sikkerhet og personvern i applikasjonen. Formålet med denne delen er å sikre enkel og sikker brukerregistrering og innlogging. Brukerregistrering tillater nye brukere å opprette en profil ved å angi nødvendig informasjon. Innloggingssystemet verifiserer brukeren for å sikre tilgangen til brukerkontoen. Profilhåndtering gir brukeren muligheten til å oppdatere sin informasjon, samt endre eller tilbakestille passord ved behov. I denne delen av applikasjonen er sikkerhet prioritert, og systemet bruker moderne autentiseringsteknologier for å beskytte brukerdata mot uautorisert tilgang.

En annen sentral del av løsningen er datahåndtering. For at plattformen skal kunne håndtere og tilpasses store datamengder, benyttes Azure Cosmos DB. Denne tjenesten tilbyr en globalt distribuert database som garanterer rask tilgang til data over hele kloden. Azure Cosmos DB behandler databaseforespørsler raskt, noe som er svært viktig for funksjoner som innholdshåndtering og brukeradministrasjon. Bruken av denne databasen muliggjør også skalerbarhet, slik at den kan behandle store volumer med data uten at det påvirker systemets ytelse.

Disse komponentene utgjør fundamentet i systemet, og samarbeider for å nå målet om å tilby en omfattende og informasjonsrik brukeropplevelse. Ved å kombinere sikkerhetsprotokoller med funksjoner for innholdshåndtering og avansert datalagring, sikres det at innholdets kvalitet og brukernes integritet opprettholdes.

6.7 Samsvar mellom produkt og kravspesifikasjon

Det er foretatt en vurdering av hvorvidt systemet samsvarer med kravspesifikasjonen. Det er ikke utført en inngående diskusjon av samsvar med alle kravene, men oppnåelse vises i form av tabeller, se figur 32 og 33. Noen krav er relativt komplekse, og disse ble vurdert av studentene til å ha behov for å diskuteres nærmere. Øvrige krav er relativt lett frem og krever derfor ikke ytterligere drøfting. I tabellene vises oppnåelse i form av "Ja", "Nei" eller "Usikkert" i kolonnen som heter "Oppnådd krav", og illustreres med fargene grønn (ja), rød (nei) og gul (usikkert).

Funksjonelle krav	
Beskrivelse	Oppnådd krav
Nettsiden skal ha en søkefunksjon, som brukerne kan benytte til å finne innhold.	Ja
Det skal være mulig for bruker å logge inn med to ulike roller, disse er administrator og kunde.	Ja
Administrator-rolle skal ha tilgang til å redigere, opprette og slette innhold og kategorier.	Ja
Administrator-rolle skal kunne administrere brukertilganger.	Ja
Kunde-rolle skal ha lese-tilgang. Kunden skal ikke kunne redigere, opprette og slette innhold og kategorier. Kunde-rolle skal ikke kunne administrere brukertilganger.	Ja
Dersom kunden ikke har et gyldig abonnement, skal kunden fortsatt ha lesertilgang, men miste denne etter en gitt tid.	Ja
Personer som ikke er kunder (ikke logget på) skal kunne lese innhold, men innhold skal blokkeres etter kort tid, f.eks. etter 10 sekunder. Hensikten er å gi eksempel på innhold. Se kontohjelp.no.	Ja
Det er ønskelig at systemet på sikt har en integrasjon mot betalingsløsning som brukere kan benytte for å betale for tilgang. Integrasjonen er ikke en del av leveransen, men webapplikasjonen skal være utviklet med tanke på integrasjon i fremtiden.	Ja
Det må være mulig å hente data fra Brannhjelp til STRGI via et API.	Ja
Det er ønskelig å kunne tilrettelegge for potensielle underartikler til hver hovedartikkel, dersom det er mulig innenfor prosjektets rammer.	Ja

Figur 32: Oversikt over oppnåelse av funksjonelle krav

De funksjonelle kravene er oppfylt. Brukere av applikasjonen har tilgang til søkefunksjon, og begge rollene, administrator og kunde, er implementert med sine tilhørende rettigheter. Det er også implementert funksjonalitet for at personer som ikke er logget inn, får begrenset tilgang til applikasjonen. Kravet som omhandler å hente data fra Brannhjelp til STRGI er møtt så godt det lar seg gjøre fra studentenes ende, ved å benytte en kompatibel databaseløsning slik at integrasjon kan gjøres enklest mulig. Videre arbeid med å hente data fra Brannhjelp til STRGI må gjøres i STRGI, som er programvare studentene ikke har tilgang til.

I de funksjonelle kravene er det også spesifisert at webapplikasjonen på sikt skal ha en betalingsløsning. Denne integrasjonen er ikke en del av studentenes leveranse, men webapplikasjonen skulle utvikles med tanke på en integrasjon i fremtiden. Studentene har implementert en innloggingsside, som gir et grunnlag for å administrere brukerkontoer og tilganger. Ettersom funksjonalitet for innlogging er på plass, kan en fremtidig betalingsløsning knyttes til brukerkontoer. Det er også utviklet en side hvor administrator kan drive brukeradministrering. Her kan det legges til funksjonalitet for abonnementsstyring, hvor administrator for eksempel kan endre, se og slette eksisterende brukerabonnementer. I tillegg har studentene utviklet en "Min profil"-side, som kan tilpasses for abonnementsstyring. Her kan en bruker se og oppdatere informasjon om seg selv. Videreutvikling kan være å utvide siden til å vise informasjon om brukerens abonnement med mulighet for å se betalingsinformasjon, kansellere og endre abonnementet. Disse tre sidene, innloggings-, administrator- og "Min profil"-side, kan være svært nyttig ved en fremtidig integrasjon mot betalingsløsning. De kan bidra til å redusere behov for utvikling i fremtiden og kan dermed gjøre implementasjon av betalingsløsning mer effektivt.

Ikke-funksjonelle krav	
Beskrivelse	Oppnådd krav
UX-design og brukergrensesnittet skal være intuitivt og brukervennlig, slik at webapplikasjonen gir en positiv brukeropplevelse.	Usikkert
Databasen som brukes skal være Microsoft Azure Cosmos.	Ja
Nettsiden skal støtte publisering av tekst. Dersom det er mulig innenfor prosjektets rammer, er det ønskelig at det er mulig å publisere bilder og tabeller.	Ja

Figur 33: Oversikt over oppnåelse av ikke-funksjonelle krav

De ikke-funksjonelle kravene er i stor grad oppfylt. Azure Cosmos DB har blitt benyttet som databaseløsning og webapplikasjonen støtter publisering av tekst og tabeller. Den støtter ikke publisering av bilder, men dette var ikke et må-krav.

Kravet om at grensesnitt og UX-design skal være intuitivt og brukervennlig har vist seg å være vanskelig å bedømme hvorvidt er oppnådd eller ikke. Studentene har vært usikre på hvordan dette kravet kan defineres og måles. I systemtestene ble det avdekket to feil som påvirker brukervennligheten negativt. Administrator kan ikke se alle brukere eller publikasjoner uten å zoome ut på nettsiden. Dette skyldes at et element på siden, en “footer”, blokkerer visningen av nederste bruker og publikasjon. Dette kan løses i fremtiden ved å endre på posisjonen til dette bunnelementet eller legge til mer avstand i designet.

Det er dog implementert funksjonalitet som ikke er direkte spesifisert i kravspesifikasjonen, men som bidrar til økt brukervennlighet. For eksempel er det implementert mulighet for å tilbakestille passord ved behov og bruker kan oppdatere sin egen profilinformasjon. Systemet gir tilbakemelding på utførte eller ikke utførte handlinger, med forklarende feilmeldinger. For eksempel får bruker beskjed dersom en e-post er ugyldig, eller bekreftende melding om at publikasjon er opprettet. Ved sletting av brukere og publikasjoner ber systemet om bekreftelse for å hindre feilsletting.

UX-designet er enkelt med lite av innhold og knapper gjemt bak menyer. Ettersom arbeidet med designet har vært i tett samarbeid med veileder fra Ignist, bør det være innenfor deres definisjon av intuitivt. Kontrastsjekken utført viser at fargevalgene er innenfor WCAG-kravene som bedrifter i Norge er påkrevd å følge. Ideelt burde det ha blitt utført en brukertest med 10 eller flere deltagere av UX-designet eller den ferdige løsningen for å forsikre at brukergrensesnittet faktisk er intuitivt og brukervennlig. Denne typen testing er dog svært tids- og ressurskrevende, og ikke realistisk å få til på tiden avsatt til prosjektet.

Det er godt samsvar mellom webapplikasjonen og kravspesifikasjonen. Det er ingen krav som er direkte feilet, kun ett krav som det er noe usikkerhet rundt om er tilstrekkelig oppnådd. Som beskrevet i drøftingen over, har systemet et par forbedringsmuligheter, men har flere funksjonaliteter og egenskaper som tilsier at det er brukervennlig og intuitivt. Gjennom akseptanse-test og overtakelsesmøte ble det tydelig at oppdragsgiver var fornøyd med produktet, og det legges dermed til grunn at produktet samsvarer godt med kravspesifikasjonen.

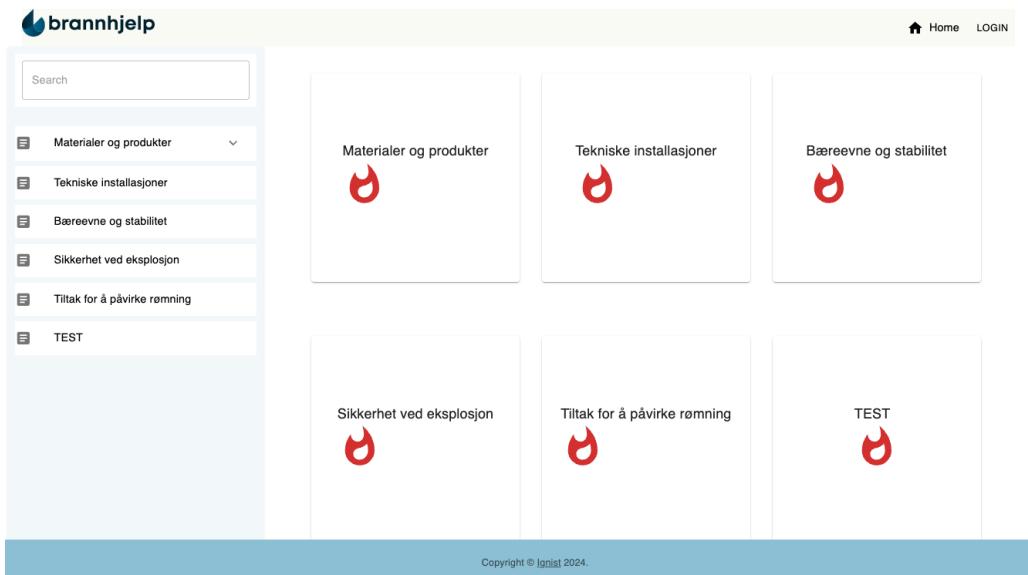
7 Brukerveiledning

I dette kapitelet gis en beskrivelse av brukerveiledning for bruk av Brannhjelp. Dette inkluderer brukerveiledning for vanlige brukere og for administrator. Brannhjelp-applikasjonen er designet for bruk av ingeniører med varierende grad av erfaring i bruk av slike teknologiske verktøy. Det forutsettes at brukerne av webapplikasjonen har noe kjennskap til slike verktøy fra før av.

7.1 Landingsside

Ved første tilgang, blir brukeren navigert direkte til hjemmesiden, se figur 23. Grensesnittets landingsside er delt inn i to distinkte seksjoner. Seksjonen til venstre tilbyr en oversikt over nylig opprettede artikler og er toppet med et søkefelt. Søkefeltet gjør det mulig å søke opp publikasjoner. Det er verdt å merke seg at søkerfunksjonaliteten er begrenset til å kun omfatte hovedkategorien, noe som gjør at brukerne kun kan søke på ett nivå under den primære kategorien. Seksjonen til høyre presenterer alle tilgjengelige artikler med klikkbare illustrasjoner.

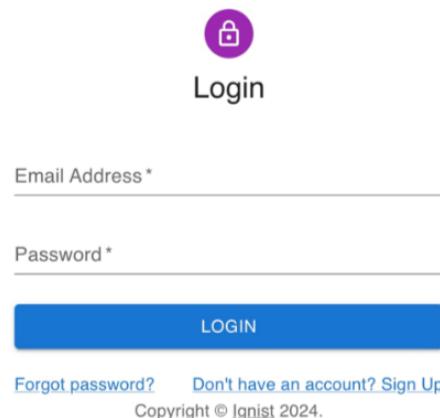
Artiklene kan leses ved å klikke direkte på dem, både fra listen på venstreside og ved klick på illustrasjonene på høyre side. For artikler med underkategorier, vises en nedtrekksmeny som gir tilgang til underartikkelen. Dersom brukeren ikke er innlogget, blir vedkommende automatisk navigert til innloggingssiden. Dette er i samsvar med bedriftens kravspesifikasjon, som krever at brukere må være autentiserte for å få tilgang til innholdet.



Figur 34: Hjemmeside

7.2 Innloggingsside

Innloggingssiden er konstruert med to felt, som vist i figur 35. For autentisering kreves det at brukeren oppgir sin registrerte e-postadresse og tilhørende passord. Brukere som ennå ikke har opprettet en konto, eller de som har glemt sitt passord, oppfordres til å benytte seg av relevante lenker plassert under innloggingsknappen for videre assistanse.

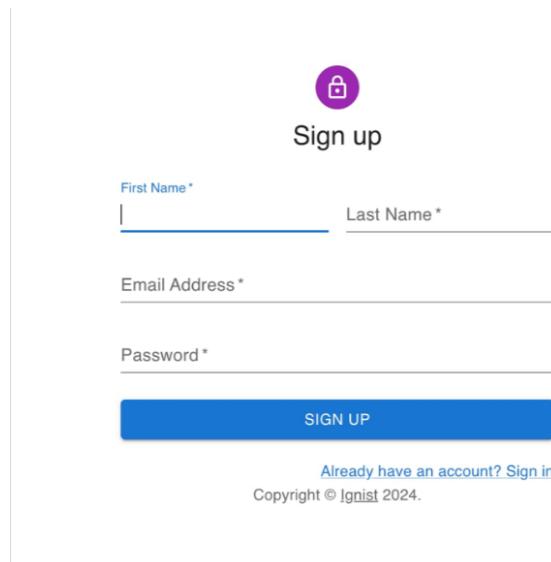


The image shows a login form with a purple lock icon at the top. Below it is the word "Login". There are two input fields: "Email Address *" and "Password *". A blue "LOGIN" button is centered below the fields. At the bottom, there are links for "Forgot password?", "Don't have an account? Sign Up", and copyright information "Copyright © Ignist 2024".

Figur 35: Innloggingsside

7.3 Registreringsside

Registreringsskjemaet inneholder fire obligatoriske inndatafelt: fornavn, etternavn, e-post og passord, som vist i figur 36. E-postadressen og passordet som oppgis av brukeren, vil undergå valideringsprosesser både på serversiden (backend) og klient-siden (frontend) for å sikre at de oppfyller fastsatte krav.

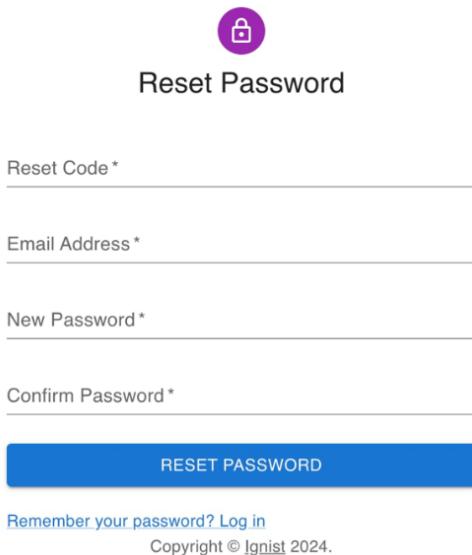


The image shows a sign-up form with a purple lock icon at the top. Below it is the word "Sign up". There are four input fields: "First Name *", "Last Name *", "Email Address *", and "Password *". A blue "SIGN UP" button is centered below the fields. At the bottom, there are links for "Already have an account? Sign in" and copyright information "Copyright © Ignist 2024".

Figur 36: Registreringsside

7.4 Glemt passord-side

Dersom en bruker har glemt sitt passord, kan vedkommende benytte seg av lenken for glemt passord som er tilgjengelig på innloggingssiden. Etter å ha klikket på denne lenken, vil brukeren bli omdirigert til en ny side der vedkommende oppfordres til å oppgi sin registrerte e-postadresse. Hvis den oppgitte e-postadressen er gyldig, vil en verifiseringskode bli sendt til denne adressen. Deretter vil brukeren bli ledet til en annen side for å opprette et nytt passord, se figur 37.



The image shows a 'Reset Password' form. At the top is a purple circular icon with a white padlock symbol. Below it is the text 'Reset Password'. There are four input fields: 'Reset Code *', 'Email Address *', 'New Password *', and 'Confirm Password *'. A large blue button labeled 'RESET PASSWORD' is centered below the input fields. At the bottom left is a link 'Remember your password? Log in' and at the bottom right is the text 'Copyright © Ignist 2024.'

Figur 37: Side for å tilbakestille passord

7.5 Administrators muligheter

Administratorrolen har mer utvidet funksjonalitet sammenlignet med standardbrukere. Ved pålogging dirigeres administratoren til hovedsiden, som er utrustet med avanserte navigasjonsmuligheter. Videre beskrives funksjonene som er tilgjengelige for administrator, og det illustreres hvordan disse kan anvendes.

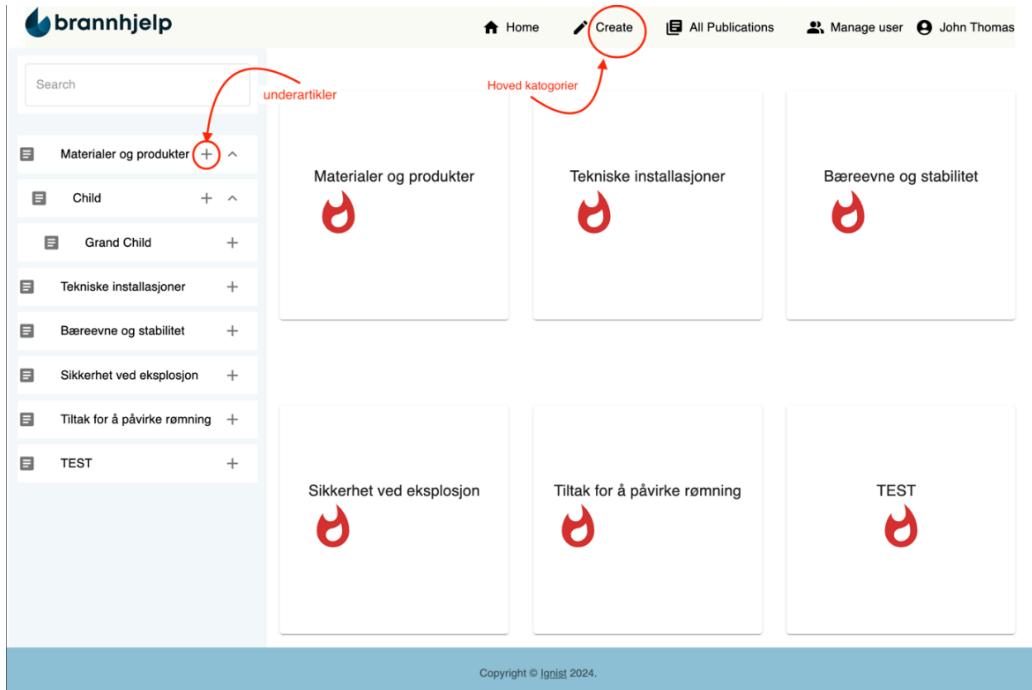
7.5.1 Opprette artikler og underartikler

For å opprette en hovedkategori kan administratoren klikke på "Create" som finnes i navigasjonsbaren. For hovedkategorier og underartikler er det to obligatoriske felt som må fylles ut. Disse er tittelen på hovedkategorien og innholdet. Administratoren kan benytte Froala, som tilbyr en rich text-editor. Figur 38 viser siden hvor administratoren kan opprette en ny kategori.

The screenshot shows a web-based content creation interface. At the top, there's a navigation bar with the 'brannhjelp' logo, 'Home', 'Create', 'All Publications', 'Manage user', and a user profile for 'John Thomas'. Below the navigation is a form for creating a new publication. It includes a 'Title *' input field, a toolbar with various text and media editing icons, a main content area with a placeholder 'Edit Your Content Here!', and a 'Powered by Froala' watermark. At the bottom is a large blue 'POST' button.

Figur 38: Side for å opprette ny artikkel

For å kunne opprette underartikler bør administratoren navigeres til hjemmesiden. I seksjonen som viser listen over alle artikler finnes det et plussstegn over hver kategori og underartikler. Dersom administratoren ønsker å legge til en underartikkkel, kan man klikke på plussstegnet ved den aktuelle artikkelen, og administratoren blir deretter dirigert til «Create»-siden. Figur 39 viser hjemmesiden til administratoren.



Figur 39: Opprettelse av underkategori

7.5.2 Redigere og slette

Administratoren har mulighet til å redigere og slette artikler. Ved bruk av navigasjonsbaren kan administratoren navigeres til siden der alle artikler er listet. Artiklene er ordnet slik at hovedkategorien kommer først og underartiklene følger etter. Ved hver artikkkel er det to ikoner, ett for sletting og ett for redigering. Administratoren kan velge mellom disse og utføre ønsket handling. Figur 40 viser siden med alle artiklene og muligheter for redigering og sletting.

ID	Title	Actions
4647	Materialer og produkter	
2621	Child	
9179	Grand Child	
8091	Tekniske installasjoner	
6656	Bæreevne og stabilitet	
2594	Sikkerhet ved eksplosjon	

Copyright © Ignist 2024.

Figur 40: Side for administrering av artikler

7.5.2.1 Brukeradministrasjon

Administratoren har autorisasjon til å slette og redigere brukerinformasjon, samt endre en brukers rolle. Denne prosessen ligner metoden som anvendes for redigering og sletting av artikler.

8 Konklusjon

Dette kapitelet tar for seg studentens læringsutbytte fra prosjektarbeidet, og refleksjoner om hva som fungerte godt og hva som kunne vært gjort annerledes. Videre følger en drøfting av bruks- og nytteverdien av webapplikasjonen ved å vurdere om prosjektmålene er oppnådd og analysere rollen denne spiller for Ignist. Deretter vil kapitelet ta for seg veien videre for løsningen, inkludert muligheter for bruk, utviklingsmuligheter og om webapplikasjonen kan settes i produksjon. Til slutt oppsummeres hovedfunnene og det trekkes en konklusjon basert på studentgruppens samlede erfaringer og vurderingen av webapplikasjonen.

8.1 Læringsutbytte

Som gruppe har vi lært mye gjennom prosjektet. Vi hadde allerede en del erfaring med både frontend og backend, men hvert prosjekt presenterer nye utfordringer og læringsmuligheter. Gjennom dette prosjektet har vi fått en dypere forståelse av hvordan single page applications (SPA) fungerer. Vi har også lært om integrasjon med skybaserte databaser. Spesielt har tilkoblingen til Azure Cosmos DB vært et sentralt punkt i læringsprosessen. Dette har ikke bare forsterket vår tekniske kompetanse, men også vårt samarbeid og problemløsningsferdigheter som et team.

Enhetstesting bød på en bratt læringskurve. Noen av studentene hadde noe erfaring med testing fra tidligere kurs om testing, men dette var begrenset til testing av Java-kode i et enkelt system. I bachelorprosjektet måtte studentene lære hvordan man setter opp tester i Visual Studio, og skrive kode for testing av .Net-applikasjoner med C# som programmeringsspråk. Prosjektet har gitt studentene en bredere forståelse for testing og hvordan tidligere erfaring kan benyttes til å løse nye, ukjente utfordringer.

Da resultatene fra enhetstestene skulle dokumenteres i rapporten, innså studentene at enhetstestene var utilstrekkelig dokumentert underveis. Testprosessen ble loggført i prosjektdagbøker og commit-beskrivelser på GitHub, men det manglet skjermbilder og detaljert dokumentasjon av feilede tester, noe som ville forbedret spørbarheten og presentasjonen av testresultater.

Viktigheten av god kommunikasjon mellom alle ledd har også blitt veldig klart for oss i løpet av prosjektet. Det har vært jevnlig kommunikasjon mellom studentene underveis i prosjektet, men ettersom vi valgte å bruke hver enkelt students talenter til den beste evne, ble det til en grad litt som å operere med forskjellige avdelinger i en bedrift. Dersom vi hadde vært enda flinkere på kommunikasjon mellom de forskjellige delene av prosjektet hadde vi hatt muligheten til å levere et enda bedre produkt til oppdragsgiver.

8.2 Brannhjelps bruks- og nytteverdi

Brannhjelp kan benyttes til kunnskapsdeling og opplæringsstøtte. Brannhjelp fungerer som et ressursbibliotek for Ignist, der selskapet kan dele veiledninger, opplæringsmateriale, og anbefalte metoder. Dette tilrettelegger for egen læring blant kundene ved å gi dem tilgang til nødvendige informasjonsressurser.

Løsningen kan også bidra til forbedring av kundestøtte. Brannhjelp vil spille en rolle i å minske belastningen på Ignist sin kundestøttetjeneste ved å tilby et lager med ressurser. Dette gjør at kunder selv kan finne løsninger og svar, noe som vil heve kundeopplevelsen og gir Ignist muligheten til å allokkere tid og ressurser mot mer komplekse forespørsler.

Brannhjelp kan også i fremtiden generere inntekt for Ignist og benyttes som markedsføringsplattform. Implementeringen av en betalingsmodell for tilgang til Brannhjelp vil gi en direkte inntektskilde for Ignist. Dette kan

bidra til å finansiere ytterligere utvikling av applikasjonen eller andre tjenester. Brannhjelp vil fungere som en markedsføringsplattform for Ignist ved å gi mulighet til å demonstrere bedriftens ekspertise.

8.3 Veien videre

Webapplikasjonen har potensiale til å bli en viktig del av Ignist sin infrastruktur. Ignist planlegger å ta i bruk Brannhjelp for kommunikasjon og informasjonsdeling via publikasjonsmodulen som allerede er implementert i systemet. Med det integrerte autentiseringssystemet, tilbyr webapplikasjonen en sikker og effektiv håndtering av brukerdata.

I tillegg til de eksisterende funksjonene, ønsker Ignist å videreutvikle løsningen ved å inkludere funksjonalitet for å publisere bilder. Dette vil tillate administratorer å legge til visuelle elementer, noe som kan forbedre forståelsen for brukerne. Det er også planlagt implementering av en betalingsløsning, som vil gi brukerne tilgang til premium-funksjoner eller innhold.

Plattformen har flere videreutviklingsmuligheter, blant annet:

- Videoinnlegg: Forbedre publikasjonsmodulen med funksjonalitet for videoinnlegg, for å forbedre forståelse
- Integrasjon med sosiale medier: Muliggjør deling av innhold på sosiale medier for å øke applikasjonens synlighet.
- Flerspråklig støtte: Implementere støtte for flere språk for å tiltrekke et bredere publikum.
- Mobilapplikasjon: Utvikle en mobilapplikasjon for å forbedre tilgjengeligheten og brukeropplevelsen på mobile, noe som er avgjørende i dagens digitale samfunn.
- Chatbot: Implementere en chatbot som kan guide brukere til riktig informasjon, forbedrer brukerstøtte og navigasjonen på plattformen.

Det finnes altså flere muligheter for å videreutvikle Brannhjelp. Med den nåværende løsningen, og et fortsatt fokus på videre teknologisk innovasjon, har Ignist og Brannhjelp gode muligheter til å imøtekommne behovene til ingeniører over hele landet.

8.4 Oppsummering og konklusjon

Vi har fullført et oppdrag om å utvikle en webapplikasjon for Ignist. Ved en blanding av smidig og plandrevne metode har det blitt utarbeidet en skisse for UX-designet, utviklet backend ved bruk av .NET og C#, og frontend ved bruk av React, og deretter blitt utført testing. Slik det er konkludert i delkapittel 6.7 har de aller fleste kravene fra kravspesifikasjonen blitt oppfylt, hvor det kun er usikkerhet om et av disse av den grunn at det er vanskelig å måle kravet. Produktet vi har levert til oppdragsgiver er fungerende og har gode muligheter for videreutvikling.

Oppdraget vi fikk fra Ignist hadde et stort omfang, men mange aspekter de ønsket at vi skulle arbeide med. Ignist har gitt oss mye ansvar på en arena hvor de selv mangler kompetanse, men har fulgt oss godt opp under arbeidet i den grad de hadde kompetanse til. Det store omfanget har gjort det til et meget spennende og lærerikt prosjekt, og til tider utfordrende. Vi har lært mye som vi kan ta med oss videre i studier og arbeidslivet.

Vi vil konkludere med at vi har lært masse, og er stolte over ikke bare slutproduktet som ble overlevert til Ignist, men også alt arbeidet rundt som måtte til for å komme hit. Med sterke kommunikasjon kunne slutproduktet blitt enda bedre, men dette har også vist seg å være et aspekt vi har lært mye av. Ignist er meget fornøyd

med innsatsen vi har lagt i prosjektet og webapplikasjonen vi har utviklet for dem, noe attesten de ga oss understreker (se vedlegg E) – dette var målet vårt som gruppe.

Litteraturliste

- Abba, I. V. (2022, 26. april). *SOLID Definition – the SOLID Principles of Object-Oriented Design Explained*. FreeCodeCamp. Hentet 15.mars 2024 fra <https://www.freecodecamp.org/news/solid-principles-single-responsibility-principle-explained/>
- Alokai. (2023, August 23). *Headless architecture: What is it, and why is it the future?* Alokai Blog. Hentet 5. mai 2024 fra <https://alokai.com/blog/headless-architecture>
- Benyon, D. (2010). *Designing Interactive Systems: A comprehensive guide to HCI and interaction design* (2. Utg.). Pearson Education Limited.
- Brutlag, J. (2009, 23. juni). *Speed matters*. Google Research Blog. Hentet 25.mars 2024 fra <https://blog.research.google/2009/06/matters.html>
- Cloudworks. (u.å.). *Technology: Identity and Access Management*. Cloudworks. Hentet 15.April 2024 <https://www.cloudworks.co.uk>
- Digitaliseringsdirektoratet (u.å.-a). *Regelverk*. Utilsynet. <https://www.utilsynet.no/regelverk/regelverk/266>
- Digitaliseringsdirektoratet. (u.å.-b). *WCAG-standarden*. Utilsynet. <https://www.utilsynet.no/wcag-standarden/wcag-standarden/86>
- Digitaliseringsdirektoratet (u.å.-c). *Kontrast minimum nivå AA*. Utilsynet. Hentet 8.mai 2024 fra <https://www.utilsynet.no/wcag-standarden/143-kontrast-minimum-niva-aa/95>
- Figma. (u.å.). *Creative tools meet the internet*. Figma. Lastet ned 21. april 2024, fra <https://www.figma.com/about/>
- FN. (2006). *Konvensjon om rettighetene til personer med nedsatt funksjonsevne*. United Nations. Hentet 4. mai 2024 fra <https://fn.no/assets/images/FN-kunnskap/Avtaler/FN-konvensjoner-filer/Konvensjon-om-rettighetene-til-personer-med-nedsatt-funksjonsevne.pdf>
- Foss-Pedersen, R. J. (2017, 24. august). *Brukertest – slik gjør du det*. UX-bloggen. Hentet 4. mai 2024 fra <https://www.usit.uio.no/om/organisasjon/ffu/ux/blogg/2017/brukertest.html>
- Froala. (u.å.). *Overview*. Froala. Hentet 20. Mars 2024 fra <https://froala.com/wysiwyg-editor/demo/>
- GeeksforGeeks (2023, 7. desember). *Manual Testing vs Automated Testing*. GeeksforGeeks. Hentet 4. mai 2024 fra <https://www.geeksforgeeks.org/software-engineering-differences-between-manual-and-automation-testing/>
- GeeksforGeek (2024, 13. februar). *Separation of Concerns (SoC)*. GeeksforGeeks. Hentet 20. Mai 2023 fra <https://www.geeksforgeeks.org/separation-of-concerns-soc/>
- Google (u.å.). *Open Sans*. Hentet 06. mai 2024 fra <https://fonts.google.com/specimen/Open+Sans>
- Granevang, M. (2020, 31. juli). fFrontend. Store norske leksikon. Hentet 20. mars 2024 fra <https://snl.no/frontend>
- Ignist AS. (u.å.). *Om oss*. Ignist. Hentet 10 mars 2024 fra https://www.ignist.no/omoss/gad_source=1gclid=Cj0KCQiAoeGuBhCBARIAGfKY7z7KOHs5FKXFeJQ2wdcAWoSDF3GsN8tOW5Hy1o5IfQkD0MPYp8YxKYa
- Kara, K. (2023, June 10). *Implementing the repository pattern in C# and .NET*. Medium. Hentet 12. April 2024 fra <https://medium.com/@kerimkkara/implementing-the-repository-pattern-in-c-and-net-5fdd91950485>
- Kinsta. (2023a, 17. november). *What Is GitHub? A Beginner’s Introduction to GitHub*. Hentet 10. Mars 2024 fra <https://kinsta.com/knowledgebase/what-is-github/>

Kinsta. (2023b, 17. november). *What Is GitHub? A Beginner's Introduction to GitHub*. Hentet 10. Mars 2024 fra <https://kinsta.com/knowledgebase/what-is-github/>

Kumar, D., & Simic, M. (2024). *Differences between repository and service?* Baeldung. Hentet 3. mai 2024, fra <https://www.baeldung.com/cs/repository-vs-service>

Lawson, K. (2022, 16. September). *What Are Single Page Applications and Why Do People Like Them So Much?* Bloomreach. Hentet 20. Mai 2024 fra <https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application>

Material UI. (u.å.). *Material UI – Overview*. MUI Core. Hentet 21. mars 2024 fra <https://mui.com/material-ui/getting-started/>

McMillan, T. (2023, 9. september). *Manual Testing vs Automated Testing: Key Differences*. Testrail. Hentet 4. mai 2024 fra <https://www.testrail.com/blog/manual-vs-automated-testing/>

Medietilsynet. (2022, 16. november). *Slik fungerer Discord*. Lastet ned 24. mars 2024, fra <https://www.medietilsynet.no/digital-medier/dataspill/discord/>

Microsoft. (u.å.-a). *What is OneDrive for work or school?* Microsoft. Hentet 2. April 2024, fra <https://support.microsoft.com/en-gb/office/what-is-onedrive-for-work-or-school-187f90af-056f-47c0-9656-cc0ddca7fdc2>

Microsoft. (u.å.-b). *Introduksjon til sikkerhet i ASP.NET Core*. Microsoft. Hentet 4. April 2024, fra <https://learn.microsoft.com/us/aspnet/core/security/?view=aspnetcore-8.0>

Microsoft. (u.å.-c). *Hva er Visual Studio?* Microsoft. Hentet 20. mars 2024, fra <https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022>

Microsoft. (u.å.-d) *What is word processor?* Microsoft. Hentet 20. Mars 2024 fra <https://www.microsoft.com/en-us/microsoft-365/word/word-processor>

Norwegian Testing Board & International Software Testing Qualifications Board. (2019). *Sertifisert Tester: Pensum for grunnleggende nivå (Foundation level): Norsk versjon basert på CTFL 2018 V1.0*. <http://www.istqb-norge.no/wp/wp-content/uploads/2019/04/CTFL-2018-Pensum-norsk-v1.0.pdf>

Portney, D. (2018, 9. april). *An SEO's survival guide to Single Page Applications (SPAs)*. Search Engine Watch. Hentet 1. April 2024 fra <https://www.searchenginewatch.com/2018/04/09/an-seos-survival-guide-to-single-page-applications-spas/>

Sandnes, F. E. (2022). *Universell utforming av IKT-systemer: Brukergrensesnitt for alle* (3. utgave). Universitetsforlaget.

SendGrid (u.å.). *Make every email count with SendGrid*. Twilio SendGrid. Hentet 16. Mai 2024 fra <https://sendgrid.com/en-us/why-sendgrid>

Serilog. (u.å.). *Why Serilog?* Serilog. Hentet 16. Mai 2024 fra <https://serilog.net/>

Sharmaa, M. (2023, December 19). *Figur 13. Understanding useContext in React: Unveiling its power*. Stackademic. <https://blog.stackademic.com/understanding-usecontext-in-react-unveiling-its-power-51233e599ee5>

Slack. (u.å) *What is Slack?* Slack. Hentet 21. mars 2023 fra <https://slack.com/help/articles/115004071768-What-is-Slack->

Sommerville, I. (2016). *Software Engineering, Global Edition* (10th edition.). Pearson.

Spillner, A., Linz, T. Schaefer, H. (2014). *Software Testing Foundations - A Study Guide for the Certified Tester Exam* (4th edition.). Rocky Nook.

Strapi. (u.å.). *What is a Headless CMS?* Strapi. Hentet 12. mars 2024 fra <https://strapi.io/what-is-headless-cms>

Stack Overflow. (2023). Stack Overflow Developer Survey 2023. Stack Overflow. Hentet May 15, 2024, from <https://survey.stackoverflow.co/2023/technology>

Swagger. (u.å.). *What is Swagger?* Swagger. Hentet 21. Mars fra <https://swagger.io/docs/specification/about/>

Robinson, S. O'Neill, L. (2024). *Microsoft Teams*. TechTarget. Hentet 21. Mars 2024 fra <https://www.techtarget.com/searchunifiedcommunications/microsoft-teams>

TypeScript. (u.å.). *Why create typescript*. TypeScript. Hentet 21. Mars 2024 fra <https://www.typescriptlang.org/why-create-typescript/>

Webaim. (u.å.). *Contrast Checker*. <https://webaim.org/resources/contrastchecker/>

Weerayut, T. (2023, November 4). *What is interface in .NET Core and why is it important?* Medium. Hentet 1. mai 2024, fra <https://medium.com/itthirit-technology/what-is-interface-in-net-core-and-why-is-it-important-40d57ecc8025>

A Iterasjoner av UX-design



Figur 41: Første iterasjon av UX-design laget i Figma, i oransj fargepalett

The screenshot shows a wireframe of a web page with a yellow header and footer. The header features the Brannhjelp logo, a search bar with placeholder text 'Søk' and a magnifying glass icon, and a 'LOGIN' button with a user icon. The main content area has a white background. On the left, there's a sidebar with a vertical list of 'List item' entries, each with a right-pointing arrow icon. To the right of the sidebar, there's a large text area titled 'Temanavn' containing placeholder text. At the bottom right of the page, there's a small note 'Publisert 12.04.2024'.

Søk

LOGIN

Temanavn

List item >

List item ▾

Undertema level 1 >

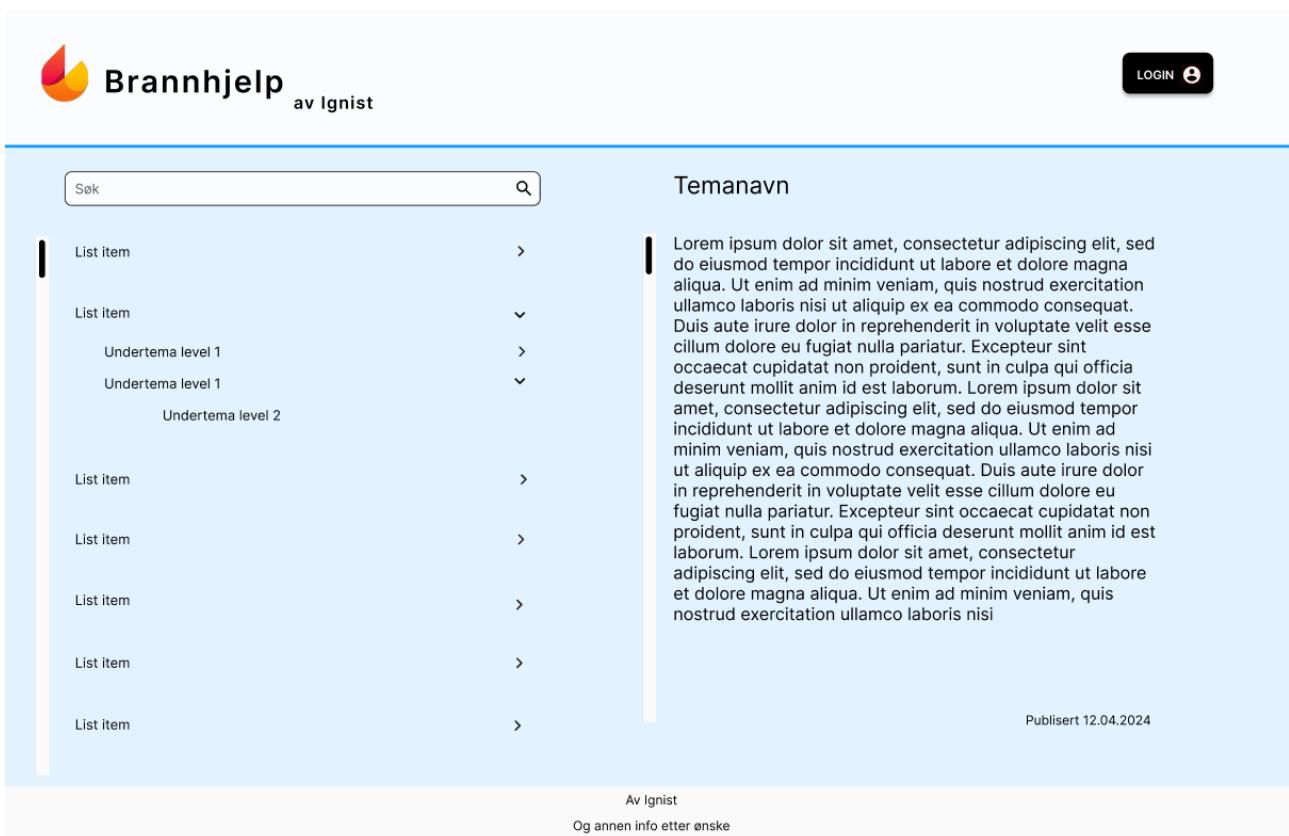
Undertema level 1 ▾

Undertema level 2

List item >

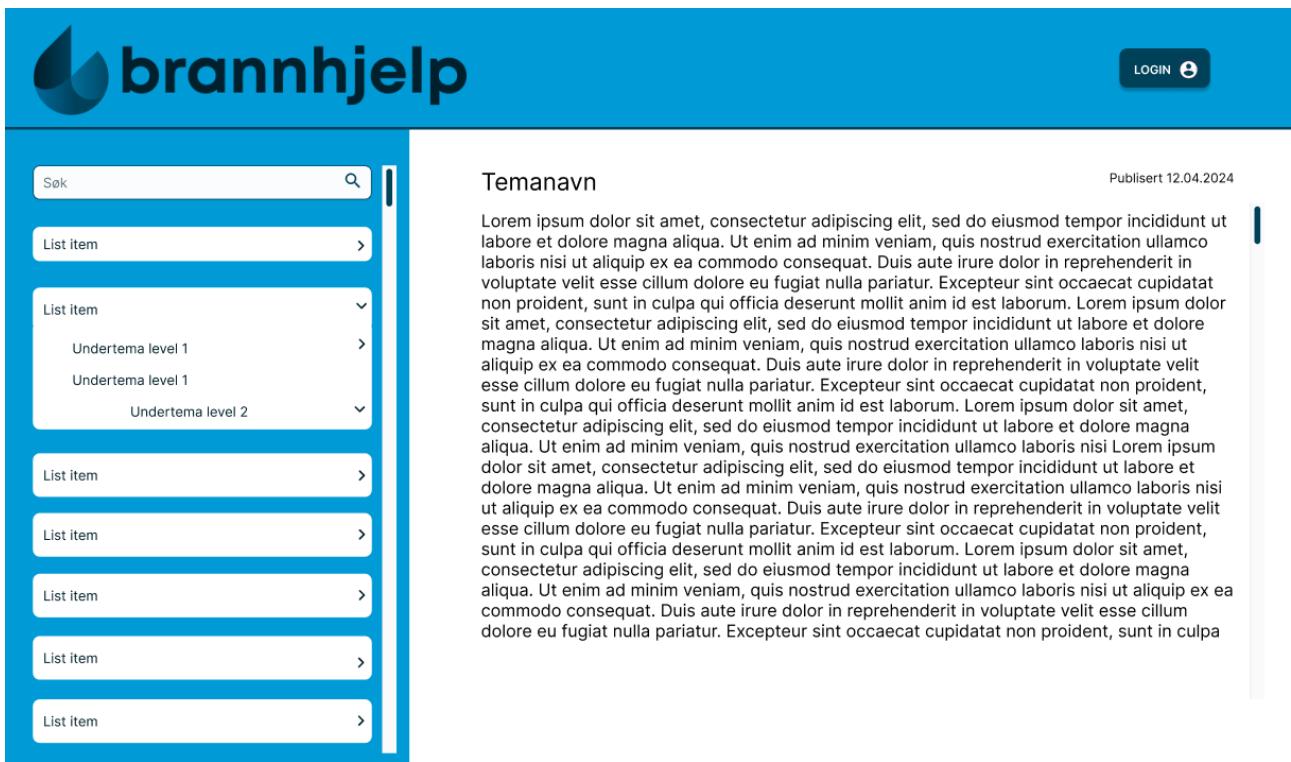
Publisert 12.04.2024

Figur 42: Første iterasjon av UX-designet laget i Figma, med andre detaljer i oransj



Figur 43: Første iterasjon av UX-designet laget i Figma, med andre detaljer i blått

Figur 44: Andre iterasjon av UX-designet laget i Figma.



Figur 45: Tredje iterasjon av UX-designet laget i Figma

Søk

Temanavn

Publisert 12.04.2024

Main content

Av Ignist

Personværserklæring og annen info

Figur 46: Fjerde iterasjon av UX-designet laget i Figma

Main content

Materialer og produkters Barn Overflatekrav & brann 

Overflatekrav

Kledningskrav

Barnebarn

Tekniske installasjoner Bæreevne og stabilitet Sikkerhet ved eksplosjon Generelle krav til rømin Tiltak for å påvirke rømin Forelder Forelder Forelder Forelder Overflatekrav og Brannbeskyttende kledning 

Publisert 12.04.2024

Overflatekrav for veggger og tak. Ved prøving observeres tid til overtenting, varmeavgivelse, røykproduksjon, brannutbredelse og brennende dråper eller deler. Minste tid til overtenting er 10 minutter for klasse D, 12 minutter for klasse C og 20 minutter for klass A og B. Overflaten kan være brennbar [D], men skal ikke produsere brennende dråper [d0] og avgj begrenset mengde røyk [s2].

Med kledning menes en byggevarer som benyttes innvendig eller utvendig på en vegg eller på undersiden av en etasjeskiller. Kledningsklassen angir kledningens evne til å beskytte sin egen bakside og bakenforliggende materiale mot antennelse. Klassen K210 betyr beskyttelse mot antennelse i 10 minutter [klassene K1-A, K1 og K2]

Overflatekrav 

Vi bruker Euroklassene for å fastsette kravene til overflater som benyttes på veggger og tak. Med overflate menes her det ytterste laget av en bygningsdel (det du kan ta på), for eksempel overflatesjikt som dannes av maling, tapet og tilsvarende. Overflate må ses i sammenheng med underlaget som overflatene er på, som sponplate, gipsplate, isolasjonsmateriale og lignende. Klassifiseringen gjelder derfor det endelige produktet, det vil si overflatene på det aktuelle underlaget.

Overflater skal dokumenteres og testes etter xxxx

Kledningskrav 

Av Ignist

Personværserklæring og annen info

Figur 47: Femte iterasjon av UX-designet laget i Figma

B Testplan

1 Innledning

Systemet som skal testes er en webapplikasjon, kalt Brannhjelp. Forsiden/landingssiden skal være en side som viser publikasjoner som kan leses og søkes på.

Brannhjelp skal inneholde en side for innlogging og opprettelse av bruker. Det er to ulike brukere: kunde og administrator. Dersom en person som ikke er logget inn, besøker Brannhjelp, skal personen få se innhold i noen sekunder før hen sendes videre til siden for innlogging. Det må altså gjennomføres en sjekk på om person er innlogget.

Administrator skal kunne oppdatere, lese, endre og slette (CRUD) innhold/publikasjoner på Brannhjelp. Administrator skal kunne opprette publikasjoner som er barn og/eller barnebarn av hovedartiklene.

Administrator skal også administrere brukere og må ha mulighet for å se, endre og slette brukere.

Kunde skal ha tilgang til å endre informasjon på sin egen bruker.

For håndtering av glemt passord/resetting av passord, benyttes et API som heter SendGrid.

2 Teststrategi

I forbindelse med utviklingen av Brannhjelp, vil vi utføre enhetstest, integrasjonstest, systemtest, akseptansetest og tilgjengelighetstest.

Enhetstest benyttes for å oppdage feil i koden og utviklingen før webapplikasjonen lanseres. Jo tidligere man oppdager feil, jo enklere er det stort sett å løse de. Enhetsstest er nyttig for å forbedre kvaliteten på systemet.

Integrasjonstest benyttes for å sikre at komponenter i webapplikasjonen fungerer sammen som forventet. Det er flere utviklere som skriver koden, og da er det spesielt viktig å sjekke at koden fra ulike personer fungerer sammen slik som den skal.

Systemtest er viktig for å sikre at webapplikasjonen tilfredsstiller oppdragsgivers krav og ønsker, og for å finne ut hvordan løsningen faktisk brukes av andre utenforstående. Ignist har uttrykket ønske om fokus på UX design, så systemtest vil være spesielt viktig for å tilfredsstille dette kravet.

Akseptansetest benyttes til å undersøke om produktet svarer til oppdragsgivers krav og mål. Før leveranse av webapplikasjonen, må oppdragsgiver godkjenne systemet. Ignist skal i denne fasen kontrollere Brannhjelp og sjekke at den svarer til kravspesifikasjonen.

Tilgjengelighetstest gjøres for å undersøke om Brannhjelp er universelt utformet og tilgjengelig for alle brukere.

2.1 Tidsplan

Prosjektrapporten har innleveringsfrist 24. mai 2024. Levering av webapplikasjon til Ignist er planlagt å skje en uke i forveien av innlevering av rapporten, 16. mai.

Testing gjøres kontinuerlig under utvikling av webapplikasjonen. Ved feilretting og oppdateringer, skal endringer og ny funksjonalitet testes. Siste test av webapplikasjonen gjøres samtidig som ferdigstillelse av den. All testing skal altså være fullført innen 16.mai.

Enhetstest gjøres samtidig som koden skrives. Integrasjonstest utføres når funksjoner og komponenter er ferdigstilt. Eventuelle feil rettes etter integrasjonstest og testes så på nytt.

Systemtest og tilgjengelighetstest bør påbegynnes ca. 2 uker før systemet skal overlevers, 2.mai, for å rekke å teste og rette eventuelle feil.

Flere tester gjøres ved behov så lenge vi har ressurser til det.

2.2.1 Mål for enhetstester Øke kvalitet på koden ved å sikre at hver kode-enhet har korrekt logikk.

Utført positiv og negativ enhetstest på alle individuelle enheter i alle controllere og models, for å sikre at koden fungerer korrekt i ulike scenarier.

Oppdage feil i koden tidlig i utviklingsprosessen.

Rettet og retestet alle feil som oppdages, retestet til testene er passert.

2.2.2 Mål for integrasjonstest Alle funksjonaliteter og komponenter som jobber sammen, skal være integrasjonstestet og fungere

Identifisere og rette feil i grensesnittet mellom forskjellige deler av systemet.

Teste at dataflyt og ulike komponenter fungerer som forventet.

2.2.2 Mål for systemtest Brukertestene skal ha spesifiserte krav som skal være oppfylt.

Kravene skal være dokumenterte i form av brukerhistorier.

Validere at systemet oppfyller alle funksjonelle krav i kravspesifikasjonen.

Verifikasiere at brukergrensesnitt og brukeropplevelsen er brukervennlige.

Identifisere og rette eventuelle feil før systemet overleveres til Ignist

2.2.3 Mål for akseptansestest

Bekrefte at Brannhjelp oppfyller kravene til Ignist og sluttbrukere

Validere at systemet er i tråd med kravspesifikasjonen

Sjekke at systemet fungerer som forventet når systemet er i drift.

Identifisere eventuelle avvik mellom systemets funksjonalitet og Ignist sine behov

Levere et system til Ignist som de kan godkjenne og akseptere **2.2.4 Mål for tilgjengelighetstest**

Bekrefte at Brannhjelp er universelt utformet

Sikre at Brannhjelp overholder lovkrav og retningslinjer

Bekrefte at Brannhjelp har god lesbarhet for personer med nedsatt syn og/eller fargeblindhet

Validere at eventuelle bilder har alternativ tekst som beskriver bildet innhold og funksjon

3 Gjennomføring av testene

3.1 Testverktøy

Enhetstest:

Benytter innebygd funksjonalitet i Visual Studio for å enhetsteste. Oppretter eget testprosjekt med referanse til Brannhjelp-prosjektet.

Må installere bibliotek, Moq.Entity.FrameworkCore, slik at database og kontrollere kan mockes.

Benytter Visual Studio Community. Det er kun Enterprise-versjonen som har innebygd funksjonalitet for å sjekke code coverage. Vurdere om vi kan finne andre verktøy for å sjekke code coverage.

Testmetodene skrives som metoder med arrange, act og assert.

Integrasjonstest: testskjema i Excel. Testene gjennomføres i SoapUI.

Systemtest: benytte Katalon eller Selenium for automatiserte tester. Benytter Word til å planlegge, administrere og dokumentere brukertestene. Skriver brukerhistorier basert på kravspesifikasjonen og omgjør disse til test cases. Alle relevante krav i kravspesifikasjonen dekkes av brukerhistoriene.

Akseptansestest: ingen spesielle verktøy-behov. Gjøres muntlig i overtagelsesmøte hvor studentene viser frem det de har gjort.

Tilgjengelighetstest: Nettside for sjekk av tilgjengelighetstest. Ikke bestemt hvilken enda.

3.2 Testobjekter

Controllere:

PublicationsController

AuthController

Modeller:

Publication

RegisterModel

LoginModel

User

UserUpdateModel

ResetPasswordRequest

UpdatePasswordModel

De to controllerne prioriteres i enhetstest, da disse inneholder den mest sentrale funksjonaliteten i systemet. Alle funksjoner i controllere enhetstestes.

Krav og annotasjoner i modellene skal også enhetstestes, f.eks. attributtet ‘Username’ som er annotert med ‘Required’.

Alle funksjoner i controllerene skal integrasjonstestes.

I systemtest er systemet i sin helhet testobjekt, både front end og back end. For systemtesten lages brukerhistorier og test cases for å teste funksjonalitet i systemet. Brukerhistorier og test cases baseres på krav fra kravspesifikasjonen og bør dekke alle kravene derfra. Som en del av systemtest, gjøres også en tilgjengelighets-test hvor nettsiden testes for brukervennlighet (universell utforming).

I akseptansetesten er det også systemet i sin helhet som testes. Studentene viser Ignist funksjonalitet i systemet. Brannhjelp overrekkes deretter til Ignist for utprøving av systemet. Ignist oppfordres av studentene til å gi tilbakemelding og konstruktiv kritikk.

C Brukerhistorier til systemtest

Brukohistorie 1:

Som en person som ikke er logget inn, skal jeg kunne se eksempel på innhold, men innholdet skal blokkeres etter kort tid.

Brukohistorie 2:

Som kunde skal jeg kunne logge inn på Brannhjelp og se innhold.

Brukohistorie 3:

Som administrator skal jeg kunne logge inn med administrator-rolle på Brannhjelp og se, opprette, redigere og slette innhold. Jeg skal kunne opprette artikler som underartikler til hovedartikler. Alle artikler som opprettes skal vises på forsiden og under en oversiktsside over alle publikasjoner/artikler.

Brukohistorie 4:

Som administrator skal jeg kunne se alle andre brukere og jeg skal kunne administrere brukertilganger ved å gi andre brukere administratorrettigheter. Jeg skal også kunne endre og slette andre brukere. Jeg vil ha mulighet til å enkelt se hvilken rolle en annen bruker har.

Brukohistorie 5:

Som kunde skal jeg kunne lese artikler, men jeg skal ikke ha mulighet til å endre eller slette noen artikler eller andre brukere. Jeg vil ha mulighet til å finne frem til artiklene fra en meny som viser alle artikler eller ved å klikke på illustrasjon for artikkelen på forsiden. Jeg vil også ha mulighet til å søke meg frem til artikler.

Brukohistorie 6:

Som person uten allerede opprettet konto, vil jeg ha mulighet til å registrere meg som kunde på Brannhjelp og deretter kunne logge inn med denne brukeren.

Brukohistorie 7:

Dersom jeg som kunde eller administrator har glemt passordet mitt, skal jeg kunne få resatt passordet mitt slik at jeg får logget inn på brukeren min.

Brukohistorie 8: Som kunde og/eller administrator skal jeg kunne endre min egen brukerinformasjon. Jeg vil ha mulighet til å endre mitt brukernavn, navn og epostadresse.

Brukohistorie 9:

Som kunde og/eller administrator, skal jeg kunne se tidspunkt for når en publikasjon sist ble endret.

D Test cases og resultater fra systemtester

Testrunde 1					
ID	Beskrivelse av testscenario	Forutsetninger	Steg	Resultat	Godkjent /Feilet
1.1	Man skal kunne registrere seg som kunde på Brannhjelp	Personen har ikke en registrert bruker og har tilgang til registreringssiden	Gå til Brannhjelpp nettsiden Trykk på "Don't have an account? Sign up" Fyll inn navn, epost og passord Trykk på "Sign up"	Brukere ble opprettet	Godkjent
1.2	Kunde skal kunne logge inn	Kunden har allerede en registrert bruker og forsøker å logge inn med korrekt epost og passord	Gå til Brannhjelpp nettsiden Trykk på "Login" Fyll inn epost og passord Trykk på "Log in"	Ble logget inn og videresendt til forsiden	Godkjent
1.3	Person som ikke er logget inn, blir sendt videre til innloggingssiden etter noen sekunder	Brukere ikke logget inn	Gå til Brannhjelpp nettsiden Vent 10 sekunder Bli videresendt til innloggingssiden	Ble sendt til innloggingssiden etter ca. 5 sekunder.	Godkjent
1.4	Administrator skal kunne logge inn med riktig epost og passord	Administrator har allerede registrert en bruker Epost: TEST1@example.com Passord: Test123!	Gå til innloggingssiden Trykk på feltet "Email adress" Skriv inn: TEST1@example.com Trykk på feltet "Password" Skriv inn: Test123! Trykk på "Log in"	Ble videresendt til forsiden	Godkjent

1.5	Bruker skal kunne tilbakestille passordet sitt	Man har allerede en registrert bruker og en gyldig epost-adresse. Siste versjon av kode som er pushet til repository må ha oppdatert token	Gå til innloggingssiden Trykk på "Forgot password?" Trykk på feltet "Email address" Skriv inn gyldig epost Trykk på "Send reset code" Fyll inn kode som ble sendt til epost, epost, nytt passord og bekrefte nytt passord Trykk på "Reset password"	Passord ble oppdatert til nytt passord	Godkjent
1.6	Administrator skal kunne endre rolle på andre brukerkontoer	Man har allerede logget inn med en administrator-bruker	Gå til Brannhjelppartnertidenside Trykk på "Manage user" Trykk på blyant-ikonet ved siden av brukeren som skal redigeres På "Role" velg admin i rullgardimenyen Trykk på "Update user"	Rollen endres	Godkjent
1.7	Administrator skal kunne slette andre brukerkontoer.	Man har allerede logget inn med en administrator-bruker	Gå til Brannhjelppartnertidenside Trykk på "All users" Trykk på søppelbøtte-ikonet ved siden	Etter at man trykker på "Delete", får man opp en bekreftende melding om at bruker ble slettet. Brukeren ble slettet og vises ikke lenger	Godkjent

			av brukeren som skal slettes Trykk på "Delete" Se at man får bekreftende melding om at bruker er slettet	under "Manage users"	
1.8	Administrator skal kunne endre en publikasjon	Man har allerede logget inn med en administrator-bruker	Gå til Brannhjelppartnertidens nettsiden Trykk på "All publications" Trykk på blyanten ved siden av publikasjonen som skal endres Endre på tittel Endre på teksten Trykk på "Update article"	Ble sendt tilbake til oversiktsbildet over publikasjoner. Tittelen og teksten ble endret. Endringen vises både på oversikten over alle publikasjoner og på forsiden	Godkjent
1.9	Administrator skal kunne slette en publikasjon.	Man har allerede logget inn med en administrator-bruker	Gå til Brannhjelppartnertidens nettsiden Trykk på "All publications" Trykk på søppelbøtte-ikonet ved siden av publikasjonen som skal slettes Trykk på "Delete"	Fikk bekreftende melding om at publikasjonen ble slettet. Publikasjonen er borte fra oversikt over alle publikasjoner og fra forsiden	Godkjent
1.10	Nyoppretta publikasjoner vises på forsiden og på oversikten over alle publikasjoner	Man har allerede logget inn med en administrator-bruker	Gå til Brannhjelppartnertidens nettsiden Trykk på "Create" Trykk på feltet "Title" Skriv inn tittel	Publikasjon ble opprettet Melding om at publikasjon ble opprettet vises på skjermen Publikasjonen ble publisert på forsiden	Feilet

			Trykk på tekstredigeringsvindu Skriv inn innhold på publikasjonen Trykk på "Post"	Publikasjonen ble ikke publisert på oversiktssiden over alle publikasjoner	
1.11	Man kan se tidspunkt for siste endring av en publikasjon	En publikasjon har blitt opprettet og man er logget inn med gyldig bruker.	Gå til Brannhjelppartnernettiden Trykk på en publikasjon	"Last updated" vises og indikerer korrekt dato og klokkeslett for endringstidspunkt.	Godkjent
1.12	Kunde skal kunne lese en publikasjon ved å trykke på illustrasjon på forsiden	Man har allerede logget inn som en vanlig bruker	Gå til Brannhjelppartnernettiden Trykk på rute med illustrasjon til en publikasjon	Publikasjonen åpnes og kan leses. Bruker kan scrolle for å lese hele publikasjonen	Godkjent
1.13	Kunde skal kunne lese en publikasjon ved å trykke på den i menyen til venstre	Man har allerede logget inn som en vanlig bruker	Gå til Brannhjelppartnernettiden Trykk på en publikasjon i menyen	Publikasjonen åpnes og kan leses. Bruker kan scrolle nedover for å lese hele publikasjonen.	Godkjent
1.14	Kunde skal kunne lese en publikasjon som er barn og barnebarn til en av publikasjonene i hovedmenyen på venstre side	Man har allerede logget inn som en vanlig bruker	Gå til Brannhjelppartnernettiden Trykk på pil ved siden av publikasjon for å vise barn Trykk på barnpublikasjonen Sjekk at barnpublikasjonen åpnes og kan leses Trykk på pil på barnpublikasjonen for å vise barnebarn Trykk på barnebarnpublikasjonen	Hele hierarkiet kan vises, og både barn og barnebarn er synlig i menyen til venstre ved å trykke på pil. Begge kan åpnes og leses.	Godkjent

			Sjekk at barnebarn-publikasjonen kan åpnes og leses		
1.15	Man skal kunne søke opp en publikasjon	Man har allerede logget inn	Gå til Brannhjelppublikasjonen Trykk på feltet "Search" Skriv "tek" i søkerfeltet	Kun publikasjoner som inneholder "tek" i tittel-feltet vises i listen.	Godkjent
1.16	Kunde skal ikke ha CRUD-rettigheter på publikasjoner	Man har allerede logget inn som en vanlig bruker	Gå til Brannhjelppublikasjonen Se at det ikke er knapper i navigasjonsmenyen for CRUD av publikasjoner Forsøk å skrive inn localhost:3000/APIUsers i url-feltet Se at man ikke får tilgang til denne siden som kundekolle Gjenta de to forrige stegene med url-endelser /Create og /All	Knapper for å opprette publikasjon vises ikke når man er logget inn som kunde. Dersom man prøver å skrive inn url som tar deg til side for å f.eks. opprette en publikasjon, blir man sendt tilbake til forsiden i stedet	Godkjent
1.17	Administrator skal kunne se hvilken rolle en bruker har	Man er logget inn med en administrator-bruker	Gå til Brannhjelppublikasjonen Trykk på "Manage users"	Rollen en bruker har, vises ikke. Vises ikke inne på informasjon om brukeren eller i listevisning over alle brukere	Feilet
1.18	Administrator skal kunne opprette en publikasjon som er barn til en annen publikasjon.	Man er logget inn med en administrator-bruker	Gå til Brannhjelppublikasjonen Trykk på "+" ved siden av publikasjonen som skal være forelder	Barne-publikasjon ble opprettet Systemet gir beskjed til bruker om at publikasjon ble opprettet.	Godkjent

			<p>Trykk på "Title"</p> <p>Skriv inn tittel</p> <p>Trykk på tekstredigerings-vinduet</p> <p>Skriv inn en tekst</p> <p>Trykk på "Post"</p>	<p>Opprettet en publikasjon med navn "Testbarn" som vises under korrekt forelder-publikasjon i venstremenyen på forsiden.</p> <p>Den vises også som barn på oversiktssiden over publikasjoner</p>	
1.19	Man skal kunne oppdatere sin egen brukerinformasjon.	Logget inn med registrert bruker	<p>Gå til localhost:3000</p> <p>Trykk på bruker-ikonet</p> <p>Trykk på "Profile"</p> <p>Rediger username fra "Slette" til "Slette2"</p> <p>Sjekk at informasjonen ble endret under "Profile".</p> <p>Logg inn med administrator-rolle</p> <p>Trykk på "Manage user"</p>	<p>Endring ble utført.</p> <p>Endringer vises hos en selv, under "Profile", og hos admin under "Manage users"</p>	Godkjent
1.20	Man skal ikke kunne oppdatere epost-adresse til en ugyldig epost-verdi	Logget inn med registrert bruker	<p>Gå til Brannhjel-nettsiden</p> <p>Trykk på bruker-ikonet</p> <p>Trykk på "Profile"</p> <p>Endre email til "test.test.no"</p>	<p>Epostadressen ble endret.</p> <p>Endringen ble godkjent av systemet og brukeren er nå registrert med en epost-adresse som ikke er en gyldig verdi.</p>	Feilet

			Trykk på "Update user"		
1.21	Alle brukere skal vises for administrator under "Manage users"	Logget inn med administrator-bruker. Antall brukere er 7 stk.	Gå til Brannhjel-nettsiden Trykk på "Manage users" Tell antall brukere	Antall brukere var 6 stk. Det mangler en bruker.	Feilet

Andre kommentarer og tilbakemeldinger fra testrunde 1:

En pil inne på "All publications" som ikke gjør noen ting. Fremstår som en knapp og når man trykker på den, virker det som om noe skjer. Indikerer at man kan sortere på id – men dette gjør den ikke. Se bilde under av pilen.

ID	Title
4647	Materialer og produkter
2621	Child
9179	Grand Child
8091	Tekniske installasjoner
6656	Bæreevne og stabilitet

Testrunde 2					
ID	Beskrivelse av testscenario	Forutsetninger	Steg	Resultat	Godkjent /feilet
2.10	Nyoppretta publikasjoner vises på forsiden og på oversikten over alle publikasjoner	Logget inn med administrator-rolle Antall publikasjoner er 8	Gå til Brannhjelppinnelsen Sjekk at antall publikasjoner på forsiden er 8 stk. Trykk på "All publications" Sjekk at antall publikasjoner her er 8 stk.	Antall publikasjoner på forsiden og på oversiktsbildet over publikasjoner er 8 stk. Den nederste publikasjonen er derimot skjult under footeren. Kan vises ved å zoome ut på siden. Publikasjonen er der og kan vises, men ikke god brukervennlighet.	Usikker
2.17	Administrator skal kunne se hvilken rolle en bruker har	Logget inn med administrator-rolle Brukernavn John er administrator og brukernavn Ali er normal bruker.	Gå til Brannhjelppinnelsen Trykk på "Mange users" Sjekk at bruker John har rolle "Admin" Sjekk at bruker Ali har rolle "Normal"	Kolonnen "Role" viser rollen til brukerne. John har rolle "Admin" og Ali har rolle "Normal"	Godkjent
2.20	Man skal ikke kunne oppdatere epost-adresse til en ugyldig epost-verdi	Logget inn med registrert bruker	Gå til Brannhjelppinnelsen Trykk på bruker-ikonet Trykk på "Profile" Rediger email fra "Test@test.no" til "Test.test.no" Trykk på "Update user"	Endringen lagres ikke. E-post-feltet markeres rødt og det vises en informerende tekst om at eposten ikke er i gyldig format. Ved klick på "Update user", vises feilmelding om at bruker ikke kan oppdateres	Godkjent

E Attest fra oppdragsgiver



OsloMet - storbyuniversitetet
Pilestredet 46
0167

DATO: 03.05.2024
ADRESSE
Ignist AS
C/O Collektivet
Ulvenveien 82E
0581 Oslo

ATTEST – BACHELOROPPGAVE «BRANNHJELP»

Denne attest bekrefter at *Hamid Hamrah, Hanne Larsen Fjeldaas, Karine Sørhus Johannessen, Zulfeqar Shirzadeh og Johan Sereba* (heretter «studentene») har arbeidet med en bacheloroppgave hos *Ignist AS* i perioden 10.01.2024 til 31.05.2024.

Studentene har levert en nettside med prosjektnavn «brannhjelp.no» med hensikt om å tilby veiledning for løsninger knyttet til brannteknisk utbedring av byggverk. Løsningen er bygget med tanke på integrasjon mot andre systemer (API) og med rettigheter som admin og vanlig bruker. Oppbygging og produksjon av innhold kan gjøres som admin ved en «no-code»-løsning. Studentene har blant annet jobbet med Microsoft Azure, Cosmos-DB, .NET 8 med REST API og React.

Arbeidet som er gjennomført har stor verdi for Ignist som en del av et større IT-prosjekt hvor byggebransjen skal digitaliseres og kvaliteten på leveransene økes. Løsningen gir Ignist muligheten tilby bedre tjenester til våre kunder ved at prosjekterte løsninger enklere kan kommuniseres med forslag til faktiske løsninger hos sluttbruker.

Ignist takker studentene for god innsats og ønsker de lykke til videre.

Med vennlig hilsen
Joaikim Folkesson
Co-founder og utviklingssjef
Ignist AS

1 av 1