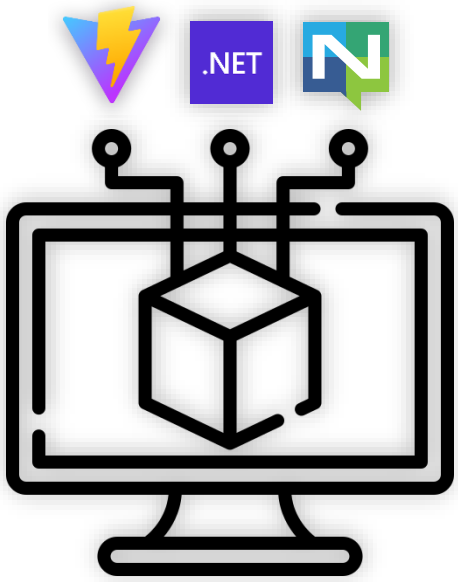
Universitetet i Sørøst-Norge

www.usn.no

TS3000 Bacheloroppgave

Utvikling av en administrasjonsportal for køsystemer basert på NATS



IA6-3-23

Daniel Eidsheim

Simen Klevengen

Tobias Günther

Fakultet for teknologi, naturvitenskap og maritime fag

Campus Porsgrunn

Universitetet i Sørøst-Norge

www.usn.no

Emne: TS3000 Bacheloroppgave

Tittel: Utvikling av en administrasjonsportal for køsystemer basert på NATS

Denne rapporten utgjør en del av vurderingsgrunnlaget i emnet.

Prosjektgruppe: IA6-3-23 **Tilgjengelighet:** Åpen

Gruppedeltakere: Daniel Eidsheim
Simen Klevengen
Tobias Günther

Veileder: Leila Ben Saad
Martin Nenseth

Prosjektpartner: Egde

Godkjent for arkivering: Ja

Sammendrag:

Mange offentlige og private bedrifter benytter digitalisering for å effektivisere etablerte rutiner og løsninger. Helsevesenet er en bransje hvor det har skjedd og skal skje mye digitalisering i de kommende årene.

Dette prosjektet var et samarbeid mellom Egde og tre bachelorstudenter fra Informatikk og automatisering ved USN Porsgrunn. Egde er en bedrift som utvikler digitaliseringsløsninger for kundene sine, herunder også helsevesenet. Bakgrunnen for prosjektet var Invictus (utviklingsteam i Egde) sitt behov for et verktøy som muliggjorde administrasjon av meldingskøer i deres digitale helseplattform «Egde Health Gateway».

Målet for dette prosjektet har vært å utvikle første utgave av en administrasjonsportal for NATS-teknologien. Det skulle opprettes en fullstack applikasjon som kunne nås gjennom en nettleser. Første utgaven av administrasjonsportalen støtter kun grunnleggende administrasjon av NATS-køer. I tillegg til utviklingen av applikasjon ble det også konfigurert et utviklings- og produksjonsmiljø som Invictus kunne overta og bruke til fremtidig videreutvikling av administrasjonsportalen.

Administrasjonsportalen som ble utviklet gjennom prosjektet er satt sammen av to parter. Den ene er frontenden som oppretter et brukergrensesnitt basert på Vite, TypeScript og Chakra-UI som sluttbrukeren benytter for å lese, opprette, slette og kopiere meldinger på en NATS-server. Backenden er den andre delen som har oppgaven med å håndtere NATS-forespørsler sendt av frontenden vha. NATS sitt eget C# bibliotek.

Universitetet i Sørøst-Norge

www.usn.no

Course: TS3000 Bacheloroppgave

Title: Development of an administration portal for queue systems based on NATS

This report forms part of the basis for assessing the student's performance on the course.

Project group: IA6-3-23 **Availability:** Open

Group participants: Daniel Eidsheim
Simen Klevengen
Tobias Günther

Supervisor: Leila Ben Saad
Martin Nenseth

Project partner: Egde

Approved for archiving: Yes

Summary:

Many public and private companies use digitalization to streamline established routines and solutions. Healthcare is an industry where there has been and will be a lot of digitalization in the coming years.

This project was a collaboration between Egde and three bachelor students from IT and automation at USN Porsgrunn. Egde is a company that develops digitalization solutions for its customers, which also include healthcare. The background for the project was that Invictus (team of developers from Egde) needed a tool that enabled administration of message queues in their digital health platform «Egde Health Gateway».

The goal of this project was to develop the first version of an administration portal for NATS technology. Therefore, a full-stack application that could be accessed through a web browser was created. The first version of the administration portal only supports basic administration of NATS queues. In addition to the development of the application, a suitable development and production environment was also configured. The goal was for Invictus to take over and use the environment for future development.

The administration portal developed through the project is composed of two parts. One is the frontend that creates a user interface based on Vite, TypeScript, and Chakra-UI that the end-user uses to read, create, delete, and copy messages on a NATS server. The backend is the other part, which is responsible for handling NATS requests sent by the frontend using NATS' own C# library.

Forord

Denne rapporten har blitt utformet med bakgrunn i programfaget TS3000. I dette faget skal studenter vise deres beherskelse av ingeniørprofesjonen gjennom utarbeidelse av en bacheloroppgave i grupper. Bacheloroppgaven skal teste studenters evne til å håndtere ingeniørfaglig arbeid med fokus på prosjektplanlegging, prosjektstyring, faglig kompetanse, samarbeid og selvstendighet. Problemstillingen eller oppgaven som bacheloren tar opp skal være av relevant ingeniørfaglig natur med tanke på studentens utdanning, samt at den skal være knyttet til næringslivet i privat eller offentlig sektor. Samarbeidspartneren for denne bacheloren var den private konsulentbedriften Egde.

Som gruppe vil vi takke alle i Egde som har vært involvert i bacheloroppgaven vår, men spesielt Martin Nenseth og Harald Loland. Som våre nærmeste veiledere loset og veiledet de oss gjennom hele bacheloren. Deres gode råd og tilbakemeldinger førte til faglig utvikling og stødig fremdrift i bachelorarbeidet.

Dessuten vil vi også takke vår veileder Leila Ben Saad fra USN. Hennes involvering og hjelpsomme holdninger sørget for at gruppen fikk utarbeidet en høykvalitativ bachelorrapport som inkluderte alle nødvendige elementer.

Daniel Eidsheim	Simen Klevengen	Tobias Günther
Signert 05.05.2023	Signert 05.05.2023	Signert 05.05.2023

Nomenklaturliste

Følgende liste gir en oversikt over forkortelser som er brukt i rapporten.

AIS	-	Association for Information Systems
API	-	Application Programming Interface
CD	-	Continuous Deployment
CI	-	Continuous Integration
CNCF	-	Cloud Native Computing Foundation
CRUD	-	Create, Read, Update and Delete
DoD	-	Definition of Done
DTO	-	Data Transfer Object
GUI	-	Graphical User Interface
IaaS	-	Infrastructure as a Service
IDE	-	Integrated Development Environment
IP	-	Internet Protocol
JSON	-	JavaScript Object Notation
MOM	-	Message-Oriented Middleware
MVC	-	Model View Controller
NATS	-	Neural Autonomic Transport System
PaaS	-	Platform as a Service
REST-API	-	RESTful Application Programming Interface
UAT	-	User Acceptance Testing
UI	-	User Interface
UU	-	Universell Utforming
WCAG	-	Web Content Accessibility Guidelines
YAML	-	Yet Another Markup Language

Innholdsfortegnelse

1 Innledning

Den første delen av denne bachelorrapporten skal danne grunnlaget for resten av dokumentet. Derfor skal den etablere forståelse for bakgrunnen og betydningen av problemstillingen som bacheloren tar opp og adresserer. Videre skal de neste kapitlene også introdusere prosjektmålet og rapportstrukturen.

1.1 Problemstilling

Dagens samfunn utvikler seg stadig raskere og dette krever effektivisering av etablerte rutiner og løsninger. Mange offentlige og private bedrifter og bransjer benytter seg av digitalisering for å oppnå den ønskede effektiviseringen. En bransje hvor det har skjedd og vil skje mye digitalisering i de kommende årene, er helsevesenet [1]. Måten digitaliseringen forekommer eller gjennomføres av ulike etater i helsevesenet, kan variere mye. Det som derimot ofte er et fellestrekk, er at eksterne tredjeparter gjennomfører digitaliseringen eller utformingen av digitale løsninger for helsevesenet.

Egde [2] er en slik ekstern tredjepart som utvikler digitaliseringsløsninger for blant annet bedrifter innenfor helsevesenet. Mer spesifikt er det utviklingsteamet Invictus som drifter og videreutvikler en digital helseplattform med navnet «Egde Health Gateway» [3] som brukes av forskjellige private og offentlige helsetjenesteleverandører. Denne helseplattformen er både stor og kompleks med tanke på omfang og teknologier som inngår i den, men et grunnleggende aspekt ved plattformen er kommunikasjon ved hjelp av meldingsutveksling. Begrepet «meldingsutveksling» betyr i dette tilfellet informasjonsutveksling mellom ulike endepunkter i helseplattformen.

Selve utvekslingen av meldinger organiseres og styres ved hjelp av NATS-teknologien [4]. Denne teknologien tilrettelegger for opprettelsen av et skalerbart køsystem, hvor meldingsutveksling kan foregå på en effektiv måte. Problemet med NATS er at det ikke finnes noe innebygde eller eksterne verktøy som muliggjør enkel administrasjon av meldingskøer. Ulempen med dette er at Invictus per dags dato sliter med tungvint vedlikehold og videreutvikling av helseplattformen, siden feilsøking av NATS-køer er tidkrevende og administrativ informasjon i forbindelse med NATS ikke er lett tilgjengelig.

1.1.1 Produkteier – Team Invictus (Egde)

Egde [5] er en norsk konsulentbedrift som i skrivende stund har omtrent 100 ansatte. Opprinnelig har Egde røttene sine på Sørlandet med kontor i Grimstad og Kristiansand, men i senere år har de også etablert kontorer i Porsgrunn og Tromsø. Visjonen til Egde er å forene mennesker og teknologi ved hjelp av digitale løsninger med formål om å skape varige verdier og gevinster for virksomheter eller samfunnet.

Å være en konsulentbedrift innebærer at Egde leier ut sine ansatte til andre virksomheter hvor disse bistår i prosjekter av ulike slag. Bistanden kan være i form av kompetanse, rådgivning, analyse og utviklingsarbeid. Ved siden av konsulentvirksomhet har Egde også leveransemodellen kalt Smia. Dette er en tjeneste hvor Egde leier ut et helt team av sine ansatte til en kunde som har et system- eller applikasjonsbehov. Forklart med andre ord overlater kunden et helt utviklingsprosjekt til Egde som setter sammen et team av ansatte fra ulike fagfelt som arkitektur, design, utvikling og testing. Dette Smia-teamet vil deretter utforme en løsning som adresserer behovene til kunden.

Invictus er et av Smia teamene til Egde som består av under ti medlemmer. Teamet jobber med å drifte, vedlikeholde og videreutvikle en helseplattform som spenner flere ulike tjenester og applikasjoner. Totalt sett har teammedlemmene et bredt kunnskapsspekter som spenner seg over flere fagfelt som eksempelvis sikkerhet, fullstack utvikling, digitalisering, brukerdesign, testing og mer.

1.2 Prosjektmål

Målet for dette bachelorprosjektet var å utvikle en første versjon av en webbasert administrasjonsportal for NATS-teknologien. Dermed skulle det opprettes en fullstack applikasjon som kunne nås gjennom en nettleser. Med første versjon menes det at den utviklede administrasjonsportalen bare skulle støtte grunnleggende administrasjon av NATS-køer. I tillegg skulle det også settes opp og konfigureres et utviklingsmiljø som kunne overleveres til Invictus sammen med administrasjonsportalen. Hensikten med utviklingsmiljøet er å muliggjøre enkel og effektiv videreutvikling av portalen i framtiden. Vedlegg A inneholder noen flere detaljer i forbindelse med beskrivelse av prosjektmålet.

1.3 Rapportstruktur

Rapporten starter med en innledning som omhandler bakgrunnen for prosjektet etterfulgt av prosjektorganisering. Kapitlene tre til seks omhandler ulike faser som ble gjennomført iterativt utover prosjektet, mens kapittel syv og åtte er diskusjon og konklusjon. Punktlisten under forklarer hvert kapittel i detalj.

- Kapittel 1 er innledningen til rapporten som inneholder bakgrunnen og målet for hele prosjektet.
- Kapittel 2 tar opp organiseringen av prosjektet med fokus på utviklingsmetoder og verktøy som ble brukt under prosjektet.
- Kapittel 3 går igjennom planlegging og analyse delen av prosjektet. Her blir innhenting av informasjon i forbindelse med forskjellige teknologier og teknikker, samt analyse av tidligere utførte studier omtalt. Dessuten blir også kravspesifikasjonen til administrasjonsportalen forklart i detalj.
- Kapittel 4 tar opp designprosessen bak applikasjonen i form av diagrammer og tegninger som viser hvordan systemet skal fungere.
- Kapittel 5 forteller om hvordan løsningen ble implementert. Dette innebærer oppsettet av utvikling- og produksjonsmiljø, verktøyene som ble brukt under utviklingen, og teknologiene bak front- og backend.
- Kapittel 6 går igjennom metoder og teknikker som er brukt for å teste utviklet applikasjon og underliggende system.
- Kapittel 7 tar opp diskusjonen rundt opphavsrett, forbedringer og videreutvikling av NATS administrasjonsportalen. Det blir også gjennomgått hvordan kildekoden ble kalitetssikret i løpet av prosjektet.
- Kapittel 8 er den endelige konklusjonen for hele prosjektet.

2 Prosjektorganisering

Effektiv prosjektorganisering er nøkkelen til et vellykket prosjekt. Det finnes mange ulike prinsipper, modeller og verktøy for organisering av prosjektgrupper og ressurser for å oppnå prosjektmål på en effektiv måte. De etterfølgende kapitlene vil ta opp og forklare hvordan dette bachelorprosjektet ble organisert og hvilke grunnleggende verktøy som ble brukt.

2.1 Agil utvikling

Å organisere utviklingsprosjekter på en agil måte innebærer gjennomføring av flere små og etterfølgende utviklingssykluser/-iterasjoner. Dette betyr at man kartlegger, designer, utvikler, tester og implementerer små deler av hele løsningen om gangen i tett samarbeid med kunden eller oppdragsgiveren frem til løsningen er ferdig laget. Til sammenligning vil mer tradisjonelle prosjektmodeller, som for eksempel Fossefallmodellen [6], gjennomgå alle de overnevnte stegene bare én gang under utviklingen av løsningen eller systemet [7].

I senere tid har det vist seg at agil utvikling gir mye bedre kvalitet på sluttproduktet, samtidig som at utviklingsprosessen foregår mer effektivt, i forhold til tradisjonelle prosjektmodeller. Derfor ble dette prosjektet organisert på en agil måte ved å adoptere deler av Scrum-rammeverket [8].

Oppbygningen av rapporten kan indikere at en tradisjonell prosjektmodell ble brukt i prosjektet. Det vil si at planlegging, design, implementasjon og testing ble gjennomført stegvis etter hverandre og det var ingen tilbakegang mellom stegene [9]. Dette var ikke tilfelle og oppbygning av hovedkapitlene er tenkt til å være en sammenfatning av fasene i de ulike utviklingssyklusene (sprintene) som har blitt gjennomgått i prosjektet.

2.1.1 Scrum-rammeverket

Scrum-rammeverket [10] er en samling av agile roller og metoder som gjør det mulig å organisere og gjennomføre prosjekter av kompleks natur. Dette gjør Scrum ved å tilrettelegge for en fleksibel og transparent leveranseprosess ved bruk av samarbeid, iterering og tilpasning blant prosjektteamet. Essensen i Scrum er å levere et verdifullt og brukbart produkt til kunden

basert på en evaluer-og-tilpass tilnærming. Noen av de mest sentrale komponentene i Scrum er produkteier, Scrum master, daglig Scrum, Sprint planlegging og Sprint Review.

Størrelsen på et Scrum team ligger vanligvis mellom fem til ni medlemmer. Et såpass lite team er å foretrekke, da det tillater mangfold i perspektiver og ferdigheter. Samtidig vil samarbeid og kommunikasjon foregå mer effektiv i mindre teams takket være enkel koordinering av oppgaver og ansvar [11]. Som presentert i vedlegg B består prosjektgruppen av tre personer. Dermed var Scrum-teamet mindre enn normen som medførte at noen av Scrum komponentene ble utelatt i dette prosjektet. Mer spesifikt ble det ikke utnevnt noe Scrum master eller gjennomført daglig Scrum. Bakgrunnen for denne beslutningen var faktumet at gruppen jobbet på samme lokasjon, både fysisk og digitalt, gjennom hele prosjektet. Dermed visste alle om hverandres arbeidsoppgaver og fremdrift til enhver tid.

Rollen som produkteier var noe som Invictus overtok i dette prosjektet. Dermed hadde de følgende ansvarsområder utover prosjektet [12]:

- Prioritere oppgaver i kravspesifikasjonene og fremdriftsplanen, slik at disse samsvarer med overordnede prosjektmål.
- Ta beslutninger om å inkludere nye eller fjerne eksisterende oppgaver fra prosjektet.
- Sikre at prosjektgruppen har en tydelig forståelse av alle oppgavene i kravspesifikasjonen og annen relevant informasjon.
- Håndtere eventuelle interessekonflikter med tanke på prosjektomfang, tidsrammer og kostnader, slik at prosjektet leverer best mulig resultat.
- Vurdere og godkjenne arbeidet og produktleveranser som prosjektgruppen gjennomfører og kommer med.

Scrum-rammeverket oppnår iterativ prosjektorganisering gjennom såkalte sprinter. En sprint representerer en kort og gjentakende syklus på normalt 1 – 4 uker. I løpet av denne perioden jobber prosjektgruppen med å fullføre oppgavene som ligger i fremdriftsplanen, bedre kjent som produkt-backlog. Målet for hver sprint er å øke kvaliteten og brukbarheten på sluttproduktet [13].

En sprint er bygget opp av fire faser. Den første fasen er sprintplanlegging hvor produkteieren og prosjektgruppen definerer omfanget til sprinten, det vil si hvilke oppgaver som skal gjennomføres i løpet av sprinten. Neste fase er selve gjennomføring av sprinten, hvor

prosjektgruppen jobber med å gjennomføre oppgavene fra planleggingen. Slutten av en sprint markeres med en sprint review og -retrospektiv. I review fasen presenterer prosjektgruppen gjennomført arbeid fra sprinten for produkteieren og kunden med formål om å få tilbakemeldinger på leveransen. Tilbakemeldingene tar prosjektgruppen med inn i sprintretrospektiv som er fjerde fasen. Her reflekterer prosjektgruppen over arbeidet som er utført med formål om å identifisere forbedringer som må implementeres i produktet fremover [10].

I dette prosjektet var lengden på sprintene satt til omtrent tre uker, og totalt ble det gjennomført fem sprinter. Dette var passelig lengde og antall sprinter med tanke på prosjektets totale varighet. Oppbygning av sprintene fulgte standard struktur med tanke på fasene, men det ble foretatt noen modifikasjoner. Disse gikk ut på at produkteieren deltok i sprint retrospektiv istedenfor planleggingen. Denne endringen ble gjort siden gjennomføringen av sprint review og retrospektiv ble kombinert til et møte kalt «Sprint Demo».

2.2 Prosjektverktøy

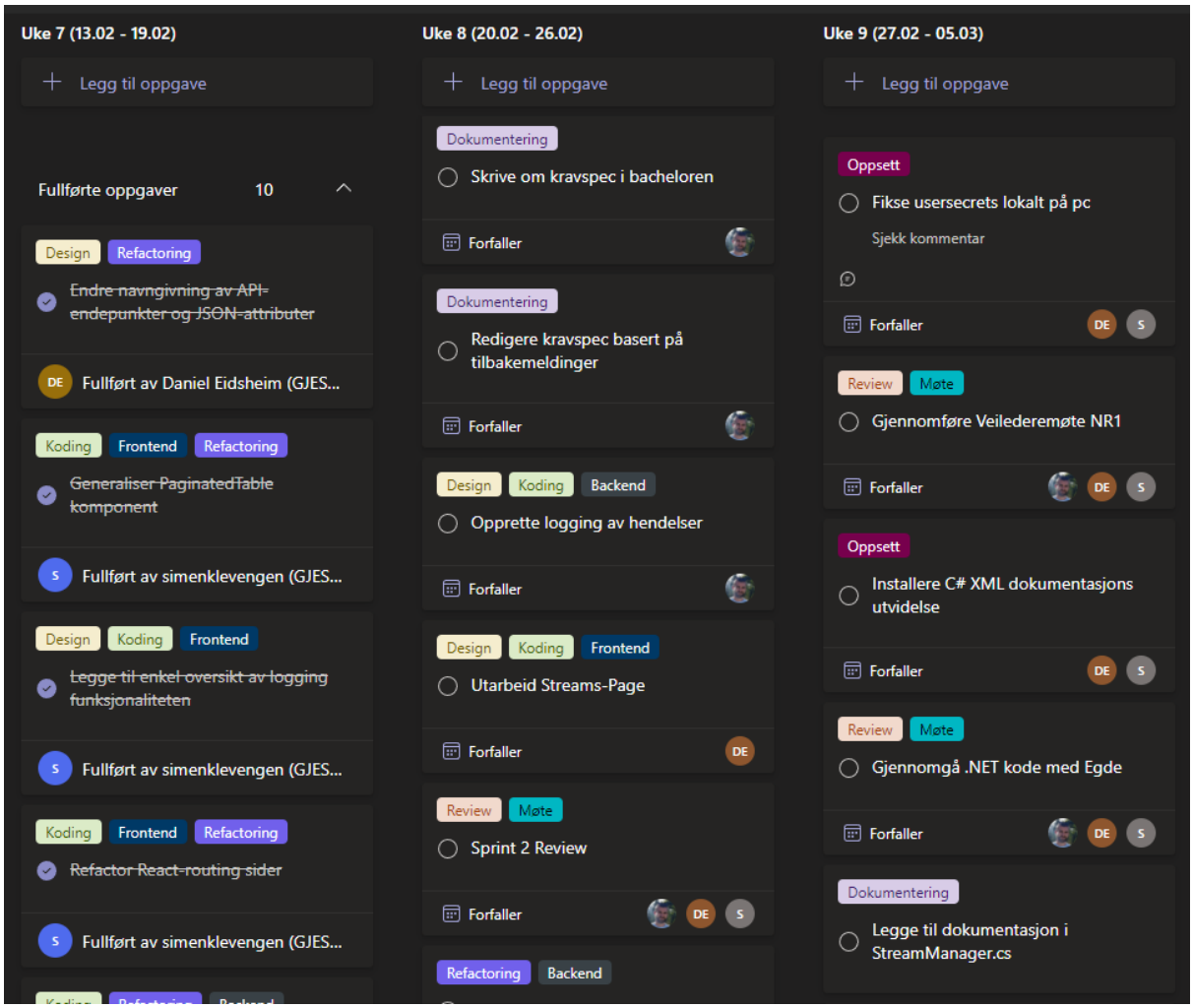
Gjennom prosjektet ble det brukt en rekke ulike verktøy og hjelpemidler for å organisere og håndtere alt arbeid relatert til administrering og gjennomføring av utviklingsprosessen. De etterfølgende kapitlene vil gi en oversikt over disse.

2.2.1 Teams

Microsoft Teams [14] er en samarbeidsplattform hvor brukere kan dele filer, avholde møter og planlegge prosjekter. Brukere kan laste opp og dele filer med teammedlemmer hvor de også har muligheten til å redigere filer sammen i sanntid. Teams har også støtte for opprettelse og administrasjon av fremdriftsplaner gjennom bruk av utvidelsen «Tasks». Denne utvidelsen leveres av Microsoft Planner [15] og lar brukere opprette og tildele oppgaver. Oppgaver kan utfylles med informasjon i forbindelse med forfallsdato, prioritet,

merkelapper, sjekklister og mer. I tillegg kan filer legges ved i oppgaver, og ulike teammedlemmer kan kommentere på oppgaven.

Under utviklingen av NATS administrasjonsportalen ble Teams brukt som et overordnet prosjektstyringsverktøy. Dette innebar at all dokumentasjon ble lagret i Teams, samt at møter med både veiledere og Invictus teammedlemmer ble avholdt her. Den tidligere omtalte Tasks-utvidelsen ble også brukt for å organisere en fremdriftsplan. Fremdriftsplanen var satt opp til å vise oppgaver på en ukentlig basis og den inkluderte alle prosjektrelaterte oppgaver, unntatt koding. Nærmere forklart viste fremdriftsplanen bare de overordnede kodingsoppgavene som ble utført. Dette hang sammen med at et eget verktøy, som omtales i kapittel 2.2.2, ble brukt for å organisere kodingsoppgavene. *Figur 2-1* viser et utklipp fra fremdriftsplanen under utviklingen, og vedlegg C viser historikken over oppgavene som ble gjennomført i løpet av hele prosjektet.



Figur 2-1: Fremdriftsplan i Teams

2.2.2 Kanban

Å organisere utviklingsprosjekter på en effektiv måte er helt essensielt, men det er like viktig å ha kontroll over alle de individuelle arbeidsoppgavene relatert til koding. På dette området finnes det mange ulike modeller og konsepter som kan hjelpe med å holde oversikt over kodingsoppgaver med hensyn til forfallsdatoer, ansvarlig person, sammenheng mellom ulike oppgaver også videre. Kanban [16] er en av de mest brukte modellene for å organisere prosjekter med fokus på kildekodeutvikling.

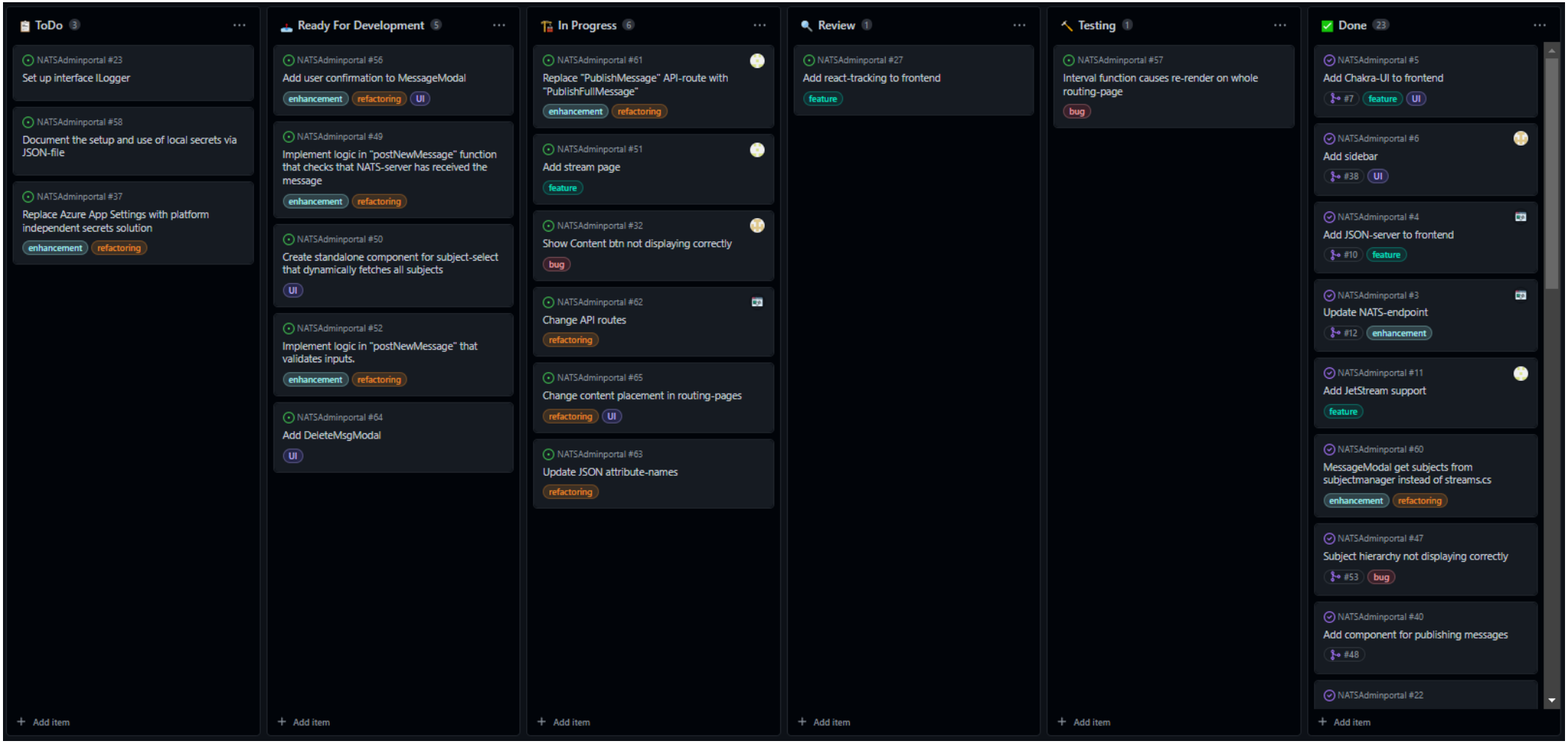
Fundamentet i Kanban-modellen er at det finnes et brett som skal gi full oversikt over alle momentane, fremtidig og tidligere oppgaver i prosjektet. Selve brettet er delt inn i kolonner, hvor hver av dem representerer en spesifikk aktivitet som skal utføres på oppgaver som er plassert i kolonnen. De enkelte oppgavene som ligger på brettet er organisert som visuelle kort, og de kan flyttes på tvers av kolonnene. Et viktig aspekt ved Kanban er at oppgaver skal være minst mulig, det vil si at hvert kort på brettet bare inneholder en enkel oppgave. Dette sørger for at Kanban-brettet er fylt med mange oppgaver, men disse er små og kan gjennomføres kjapt.

Antall kolonner og aktiviteter knyttet til kolonnene kan variere fra prosjekt til prosjekt. Kanban-modellen setter ikke noen restriksjoner til antall kolonner, men et viktig aspekt her er at bare nødvendige kolonner skal tas med. Dette betyr at prosjektteamet selv kan definere kolonnene som skal være med på brettet sitt. I dette prosjektet ble følgende kolonner brukt i Kanban-brettet:

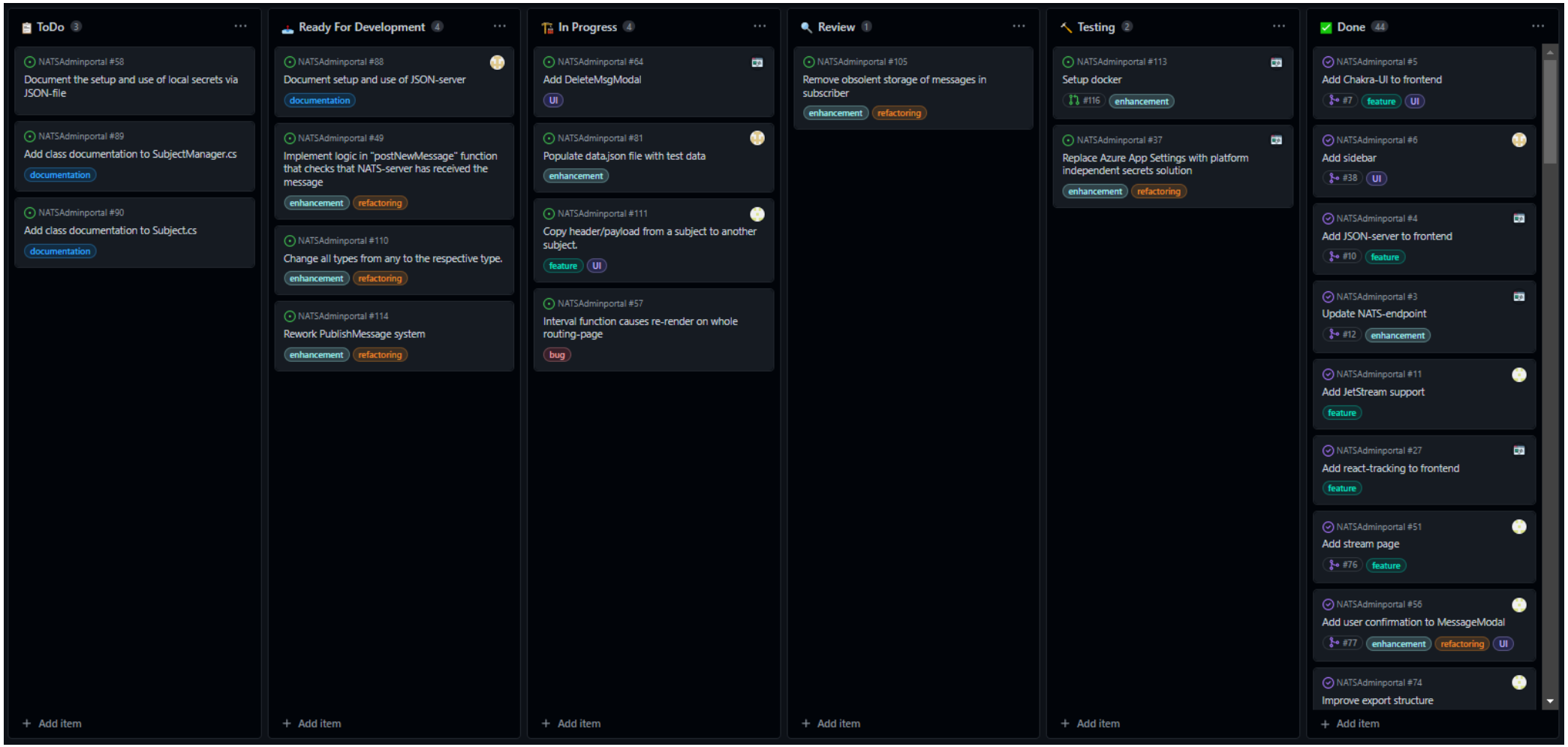
- «To Do»: startkolonnen som holdt på forslag og ideer om oppgaver som skulle tas med i utviklingsprosessen. Alt som lå i denne kolonnen, måtte først utfylles med informasjon og akseptansekriterier før det ble til en ekte oppgave som kunne flyttes videre på brettet.
- «Ready For Development»: denne kolonnen holdt på nye oppgaver som var utfylt med akseptansekriterier og informasjon, og som ventet på å bli påbegynt.
- «In Progress»: oppgaver som var påbegynt lå i denne kolonnen, og man kunne se hvem som jobbet på hvilke oppgaver.

- «Review»: oppgaver som hadde blitt fullført i henhold til sine akseptansekriterier lå i denne kolonnen. Her ventet oppgavene på å bli kvalitetssjekket av en annen utvikler enn den som jobbet med oppgaven.
- «Testing»: denne kolonnen holdt på oppgaver som hadde blitt kvalitetssjekket, men hvor kildekode måtte testes ytterligere før den kunne ansees som godkjent.
- «Done»: sluttkolonnen som holdt på alle oppgaver som ble ansett som gjennomført og avsluttet. Denne kolonnen fungerte som et arkiv som ga oversikt over alt utført arbeid.

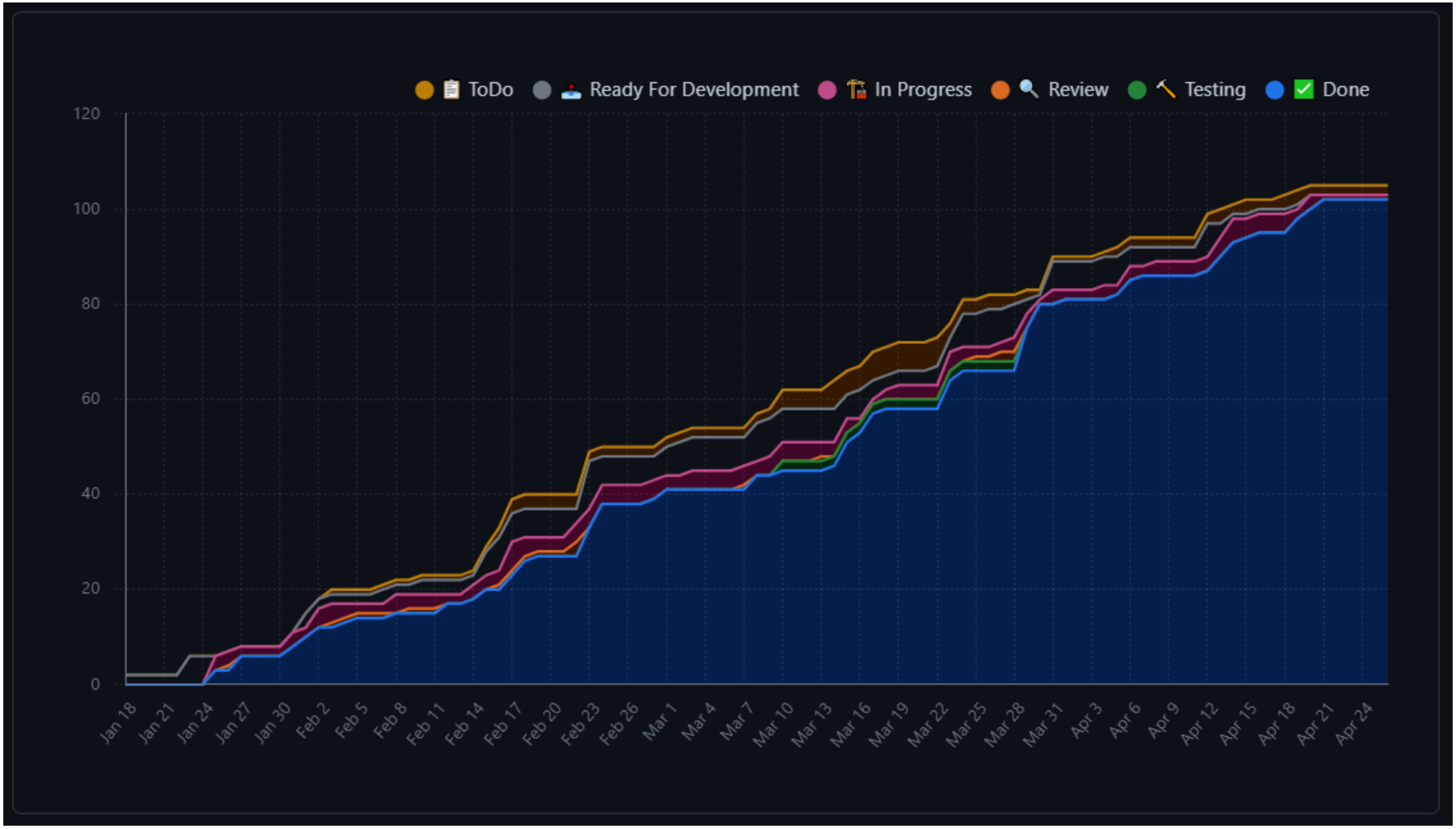
Tilbyderen av Kanban-brettet som ble brukt gjennom utviklingsprosessen var GitHub [17]. Nærmere forklart ble GitHub Projects [18] brukt for å opprette et eget Kanban-brett. Årsaken til at akkurat GitHub Projects ble brukt, var muligheten til å knytte brettet opp mot en Git-repository. Dette sørget for at alle oppgaver i brettet fikk en tydelig kobling mot branches og kildekodeendringer i prosjektets repository [19]. Begrep som Git, branches og repository blir forklart i mer detalj i kapittel 5.1.1. *Figur 2-2* og *Figur 2-3* viser noen utklipp fra Kanban-brettet underveis i utviklingsprosessen, og *Figur 2-4* viser en progresjonsgraf for hele prosjektet.



Figur 2-2: Prosjektets Kanban-brett (del 1)



Figur 2-3: Prosjektets Kanban-brett (del 2)



Figur 2-4: Progresjonsgraf for prosjektet

2.2.3 Definition of Done

Et viktig aspekt innenfor programvare- og systemutvikling er kvalitetssikring. Kvalitetssikring omhandler i denne konteksten verifikasjon av kildekode med hensyn på struktur, oppsett, standarder, logikk, dokumentasjon også videre. Mange utviklingsteam og bedrifter, som for eksempel Egde, benytter seg av konseptet rundt DoD (Definition of Done) [20]. Dette konseptet går ut på å definere en rekke med krav og gjøremål, også kjent som akseptansekriterier, som må være oppfylt for at nylig opprettet kildekode kan ansees som «ferdig». Når man jobber etter et sett med forhåndsdefinerte akseptansekriterier kan man sikre at både kundens, bransjens og interne krav følges og tilfredsstilles. Dette krever selvfølgelig at akseptansekriteriene er presise, gjennomtenkte og ikke minst relevante.

DoD er det man kaller en de facto-standard innenfor utviklingsbransjen. Dette henger sammen med at manglende kvalitetssikring på kildekode i større utviklingsprosjekter alltid vil føre til mangler, enten det er at løsningen ikke fungerer, eller at kundens behov ikke blir tilstrekkelig tilfredsstilt. Derfor er det god praksis å definere en overordnet DoD for alle typer utviklingsprosjekter. Hvilke krav og gjøremål som skal inngå i DoD, er opp til enhver bedrift eller prosjektgruppe å bestemme selv. Det er også verdt å nevne at man kan ha ulike nivåer av DoD, hvor for eksempel hvert utviklingsteam har en egen DoD basert på et sett med hoved-akseptansekriterier [21].

I Smia har Egde en overordnet DoD som alle utviklingsteamene tar utgangspunkt i. Det vil si at alle teams bruker denne som et grunnlag når egne DoD-er utformes. *Figur 2-5* viser alle punktene som inngår i Smia sin DoD. Disse punktene var også grunnlaget for opprettelsen av DoD-en som ble brukt under utviklingen av NATS administrasjonsportalen. Følgende punkter inngår i DoD-en som ble brukt gjennom prosjektet:

- Code Review, der alle pull-requests må godkjennes av en reviewer. Det kan gjøres unntak fra denne regelen dersom det er begrunnet.
- Alle oppgaver som ligger klar til utvikling, må inkludere en liste med akseptansekriterier.
- En pull-request kan først opprettes når alle akseptansekriterier er oppfylt.
- Alt arbeid må dokumenteres og være tilgjengelig. I tilfeller hvor dokumentasjon eksisterer fra før, må denne oppdateres hvis nødvendig.

- Relevante enhetstester må være opprettet, og de må være godkjente.
- Pull-requests kan kun opprettes når alle enhetstester har passert. Det kan gjøres unntak fra denne regelen dersom det er begrunnet.

Smia Definition of Done:

- code review, regelen er pull request med reviewer, men mulig å gjøre unntak dersom det er begrunnet
- akseptansekriterier i brukerhistorie er verifisert ok
- relevante akseptansekriterier for Smia eller prosjektets ikke-funksjonelle krav er verifisert ok
- intern / ekstern dokumentasjon er laget, oppdatert og tilgjengeliggjort
- alle deloppgaver er satt til ferdig og har førte timer før brukerhistorien settes til ferdig
- relevante enhetstester er skrevet og code review skal gå gjennom dette
- alle enhetstester passerer grønt
- code coverage dokumentert i byggepipeline skal kvalitetssikres
- relevant monitorering og logging er på plass
- container image er bygget og kjører i qa-miljø (for løsninger som benytter container-teknologi)

Figur 2-5: Smia sin DoD [121]

3 Planlegging og analyse

Før utviklingsarbeidet på NATS-applikasjonen kunne starte var det nødvendig å analysere og definere løsningen eller systemet som skulle lages. Dette forarbeidet er viktig fordi det legger grunnlaget for hele prosjektet. Manglende analyse- og kartleggingsarbeid vil sørge for at prosjekter ikke oppnår ønskede resultater eller kvalitet [22]. De neste delkapitlene vil forklare analyse- og planleggingsprosessen som har blitt gjennomført i dette prosjektet.

3.1 Forkunnskap

Et viktig steg i analysearbeidet er innhenting av informasjon. Dette underkapittelet viser til og forklarer informasjonen som ble hentet inn i starten av planleggingsfasen. Teoriene og konseptene som blir forklart her ble mye brukt under utviklingen av NATS administrasjonsportalen.

3.1.1 Cloud Native Computing Foundation

Cloud Native Computing Foundation (CNCF) [23] er en organisasjon med hovedmål om å fremme utvikling og bruken av cloud-native teknologier. CNCF er med på å definere den beste framgangsmåten for utvikling av skybaserte teknologier samt støtte et åpent samarbeid mellom utviklere og operatører for å forbedre skybasert programvare. I tillegg er CNCF veldig opptatt av å være et leverandørnøytralt knyttepunkt for sky-baserte løsninger.

Et av prosjektene som CNCF støtter er NATS. Årsaken til dette er at NATS er et stort Open-Source prosjekt som enhver har tilgang til og kan bruke [24].

3.1.2 Message-Oriented Middleware

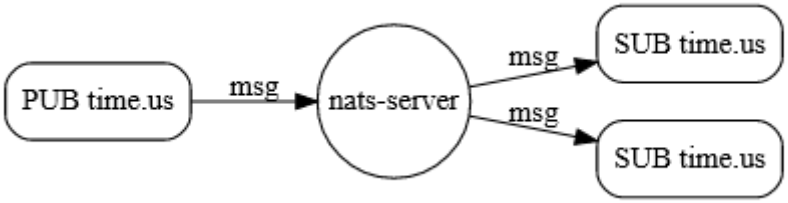
Message-Oriented Middleware (MOM) [25] [26] er programvare som gjør prosessen med å sende og motta meldinger mellom distribuerte systemer enklere. MOM lar applikasjoner kommunisere asynkront med hverandre uten at disse er klar over hverandres eksistens.

Dessuten sikrer MOM at meldinger ikke endres under transport og at de når fram til sin destinasjon ved hjelp av teknikker som bekreftelse og gjensending.

3.1.3 NATS

Neural Autonomic Transport System (NATS) [4] er et MOM designet for å være effektivt, raskt og brukervennlig. Egde bruker NATS ovenfor konkurrentene som RabbitMQ [27] og Apache Kafka [28] i helseplattformen sin fordi NATS er ressurseffektivt og enkelt å skalere. Tidligere har Egde brukt Apache Kafka, men på grunn av den høye ressursbruken til Kafka-infrastrukturen gikk de over til NATS.

NATS bruker en publisher-subscriber modell for å sende og motta meldinger. *Figur 3-1* viser en tegning som forklarer denne modellen. Her kan utgivere sende meldinger til et spesifikt subjekt, eller subject på engelsk, og abonnenter kan motta meldinger fra gjeldene subjekt. Subjekter kan organiseres i hierarkier ved å bruke «.» mellom to ord eller tokens. *Figur 3-2* viser et eksempel på et subjekthierarki som kunne ha eksistert i en applikasjon med verdensklokke funksjonalitet.



Figur 3-1: Publisher-subscriber modellen [119]

```
time.us
time.us.east
time.us.east.atlanta
time.eu.east
time.eu.warsaw
```

Figur 3-2: Eksempel på subjekthierarki [119]

Ved siden av publisher-subscriber modellen støtter NATS også request-reply modellen. Denne modellen er bygget på publisher-subscriber modellen og går ut på at en klient sender en forespørsel og får svar fra NATS-serveren direkte. Mer spesifikt, blir en melding publisert til et subjekt som har et definert reply-subject. Denne meldingen vil mottas av en mottaker som overvåker det originale subjektet og deretter sender et svar til reply-subject. Reply-subjects er enestående og blir automatisk rettet mot den originale avsenderen, uavhengig av hvor partene befinner seg [29].

En av nøkkelegenskapene til NATS er dets evne til å håndtere store antall tilkoblede klienter samtidig, og høy gjennomstrømning av meldinger. Det er skrevet i Go programmeringsspråket og er en åpen-kilde tjeneste [30].

3.1.3.1 Streams og Consumers

Innenfor NATS finnes det to konsepter som muliggjør lagring og håndtering av meldinger. Streams [31] er en av disse, og den lagrer meldinger ved å konsumere meldinger publisert på et eller flere forhåndsdefinerte subjekter. Eksempelvis vil en stream konfigurert til å konsumere subjektet «exampleSubject», lagre alle meldinger som publiseres under dette subjektet på NATS-serveren. Ved siden av subjekter kan en stream også konfigureres på andre måter. Noen andre stream-parametere er maksimal meldingsstørrelse, hvordan meldingene skal lagres, levetiden på lagrede meldinger og mer.

Consumers [32] er det andre konseptet etter hvilket meldinger kan håndteres. Enkelt forklart kan en consumer anses som et grensesnitt mellom klienter og meldinger som ligger i en stream. Consumeren konsumerer et sett av meldingene i streamen for å levere de til klienten. Gjennom consumer-konfigurasjon kan det angis spesifikke subjekter som meldinger skal hentes fra. I tillegg kan consumeren konfigureres til å være av typen «push» eller «pull», om den skal få alle meldinger eller bare de seneste og hvordan den skal acknowledge meldinger.

En «push» consumer vil bare motta meldinger levert til et spesifikt subjekt, mens en «pull» consumer har mulighet til å få grupper av meldinger etter forespørsel. Dette betyr at hvis en bruker ønsker å motta bestemte meldinger fra en stream, bør en «push» consumer brukes. Hvis brukeren trenger å behandle meldinger og skalere horisontalt, ville brukeren bli en "pull" consumer. En fordel ved å bruke consumere er leveringsgaranti av meldinger. Ved vanlig

sending av meldinger gjennom «Core NATS» leveres de med en «at most once»-garanti. I motsetning tilbyr en consumer «at least once»- garanti.

3.1.3.2 JetStream

For at en NATS-server kan benytte streams må JetStream [33] være aktivert på den. JetStream er et system som bygger på «Core NATS» og som åpner for stream funksjonalitet. Denne funksjonaliteten innebærer eksempelvis bekreftelse av meldinger som sendes mellom en stream på serveren og klienten. Videre lagrer JetStream meldinger på en stream slik at consumere ikke trenger å være aktivt tilkoblet serveren. Hvis en tidligere inaktiv consumer kobler seg opp mot serveren igjen vil den kunne hente meldinger direkte fra streamen. Meldinger som lagres i en stream gjennom JetStream får tildelt en identifikator i form av et sekvensnummer. Denne identifikatoren er unik for hver melding i en stream. Dette sørger for at meldinger kan slettes eller kopieres ved å identifisere den ved hjelp av navnet på streamen og sekvensnummeret.

3.1.4 Azure Cloud

Microsoft sin Azure Cloud [34] er en skytjeneste som tilrettelegger for bygging og distribusjon av applikasjoner, hosting av virtuelle maskiner, drift av databaser og mer. En av tjenestene som tilbys gjennom Azure Cloud er Azure App Services [35]. Denne tjenesten muliggjør hosting av web-applikasjoner i skyen. Applikasjoner som kan publiseres gjennom App Services kan være utviklet med .NET, Java, Node.js og mange andre programmeringsspråk.

Azure Cloud brukes av NATS-applikasjonen i forbindelse med utviklings- og produksjonsmiljøet. Her kan både utviklerne og produkteier teste den mest oppdaterte og operative versjonen av applikasjon i et levende miljø. Kapittel 5.2 vil ta for seg dette temaet i mer detalj.

3.1.5 REST-API

En sentral del av administrasjonsportalen er kommunikasjonen mellom backend og frontend. Kommunikasjonen mellom disse to partene organiseres ved hjelp av et RESTful Application

Programming Interface (REST-API) [36]. Et API er en samling av protokoller og definisjoner som styrer og kontrollerer forespørsler og responser mellom en informasjonsleverandør og -bruker. I NATS-applikasjonen fokuserer forespørsler og responser på utveksling av stream og meldingsdata.

At et API er RESTful betyr at det følger en rekke med arkitektoniske begrensninger som skal sørge for at API-et er ukomplisert og fleksibelt i bruk og vedlikehold. Essensen i de arkitektoniske begrensningene er at API-et skal fungere på server-klient-basis, samtidig som at all kommunikasjon mellom server og klient skal være tilstandsløs. Dette betyr at serveren ikke skal lagre noe informasjon om klienten, slik at alle forespørsler fra klienten behandles likt. På lik linje med at serveren ikke skal ha informasjon om klientene, skal klientene ikke vite noe om hvordan serveren opererer. I praksis betyr dette at klientene ikke vet noe mer enn at serveren betjener forespørslene deres [36].

3.2 Litteratursøk

Et viktig steg som ble gjennomført i starten av prosjektet var et litteratursøk. Mer spesifikt ble det undersøkt hva forskningen rundt fullstack-utvikling, MOM og NATS hadde å si. Hensikten med litteratursøket var å identifisere eksisterende kunnskap, relevant informasjon, kunnskapshull og eventuelle metoder [37]. Det vil si at litteratursøket hjalp med å danne en solid kunnskapsbase for dette prosjektet og den ga en dypere forståelse av prosjektets fagområde. Videre hjalp den med å forebygge unødvendig dobbeltarbeid, siden eksisterende forskning avdekket problemer og utfordringer som allerede har blitt undersøkt og løst.

De neste kapitlene gjennomgår noen av de mest relevante studiene som ble funnet under litteratursøket. Informasjonen og innsikten fra studiene har vært med på å definere framgangsmåter i dette prosjektet og fordeler ved å bruke NATS.

3.2.1 Fullstack-utvikling

For å definere terminologien «Full-Stack» er det sett på én studie utviklet av Association for Information Systems (AIS) [38]. Selve terminologien Full-Stack er relativt ny innenfor

utviklingsverdenen og derfor har forfatterne av studien prøvd å definere terminologien. Studien tar for seg hvordan Full-Stack kan defineres på en generell basis, ettersom den egentlige definisjonen er under debatt. De største uenighetene i forbindelse med definisjonen baserer seg på den forventede kompetansen til såkalte Full-Stack utviklere.

Per i dag er en stack definert som en stabel av kompetanse som inkluderer server systemer, som for eksempel database- eller web-servere, og programmeringsspråk. Videre deles ulike stack-er opp i forskjellige teknologier. Et eksempel på en stack er .NET som bruker C#-klassebiblioteker for programmering, SQL-server for database, IIS for web-hosting og Microsoft Server som operativt system [38].

Målet med studien er å definere terminlogien Full-Stack slik at det blir lettere for utviklere å definere egen kompetanse. Til gjengjeld hjelper dette bedrifter og bransjen med å finne de rette personene til ulike jobber. Med tanken på prosjektet hjalp informasjon fra denne studien med å fordele ansvarsområder på de ulike medlemmene i prosjektgruppen. Dermed fikk hvert medlem tildelt arbeidsoppgaver som passet best til vedkommendes evner og kompetanse.

3.2.2 Sammenligning av forskjellige MOM-er

Det finnes mange studier som sammenligner ulike MOM-er [39] [40] [41] [42] med tanke på brukbarhet og ytelse. Siden NATS er en relativt ny MOM betyr dette at mange av studiene ikke inkluderer den i sammenligningene, men det ble funnet to studier [41] [42] som gjør det.

Å avgjøre hvilken MOM som best egner seg i forskjellige situasjoner kan være vanskelig. Dette grunnet et hav av muligheter hvor hvert MOM skiller seg ut når det kommer til ytelse, ressursbruk, funksjonalitet og mer. I de to studiene som ble funnet tar forfatterne for seg akkurat denne problemstillingen. Målet med studiene er å definere bruksområdene som de ulike MOM-ene passer best til. Dette gjøres ved å sammenligne MOM-ene med hensyn på hvilke egenskaper de har og deres ytelse under forskjellige belastninger.

Selv om studiene ikke ga direkte relevant kunnskap og innsikt for utviklingen av administrasjonsportalen på grunn av kravet om å bruke NATS, var informasjonen likevel nyttig. Informasjonen fra studiene bidro til en bedre forståelse av Invictus' valg om å bruke

NATS i Egde Health Gateway. Dette igjen førte til en bedre utforming av systemarkitekturen til administrasjonsportalen.

3.3 Kravspesifikasjon

Kravspesifikasjonen er et av de viktigste dokumentene innenfor planleggingsfasen i utviklingsprosjekter. Årsaken til dette er at kravspesifikasjonen fungerer som utgangspunkt for utviklingsteamet til å lage en løsning som tilfredsstiller alle krav og ønsker til kunden eller sluttbrukeren. Innholdet i kravspesifikasjonen definerer alle funksjonelle, forretningsmessige og tekniske krav som løsningen burde inkludere eller oppfylle [43]. Å ha en veldefinert og detaljert kravspesifikasjon er også til stor hjelp i forbindelse med planlegging. Siden en oversikt over alle krav hjelper med å identifisere nødvendige arbeidsoppgaver og deres omfang. De identifiserte arbeidsoppgavene kan deretter organiseres i en fremdriftsplan og danne grunnlaget for budsjetter av ulike slag [44].

Et annet viktig aspekt ved kravspesifikasjonen er reduksjon av potensielle misforståelser mellom utviklingsteamet og kunden [45]. Normalt er det utviklingsteamet som utarbeider et første utkast av kravspesifikasjonen som kunden deretter vurderer og godkjenner. Det er under kundens vurdering av utkastet hvor eventuelle misforståelser eller mangler avdekkes. Her er det også viktig å påpeke at en kravspesifikasjon er ett levende dokument. Med andre ord vil spesifikasjonen endres fortløpende utover prosjektet, og derfor vil det være kontinuerlig samarbeid mellom utviklingsteamet og kunden.

Hvordan en kravspesifikasjon utformes, og hva den skal inkludere kan variere mye fra prosjekt til prosjekt. Det finnes mange ulike modeller som kan sikre at en grundig kravspesifikasjon utformes, og i dette prosjektet ble FURPS-modellen brukt. Videre ble det også utarbeidet et sett med Use-case diagrammer som inngikk i kravspesifikasjonen.

3.3.1 FURPS-analyse

FURPS-modellen [46] er et mye brukt rammeverk innenfor programvareutvikling som hjelper med å samle og kategorisere krav. Hensikten bak kategoriseringen er å sikre at løsningen

oppfyller alle nødvendige kvalitetskrav som kunden eller sluttbrukeren etterspør. Kategoriene som benyttes i FURPS er:

- Functionality: hvilken funksjonalitet løsningen må inneholde. Her inngår både funksjonelle og ikke-funksjonelle krav.
- Usability: hvor enkel og intuitiv løsningen er å bruke.
- Reliability: løsningens evne til å fungere korrekt og konsekvent. I tillegg til at løsningen forebygger og unngår feil eller krasj.
- Performance: hvor effektivt løsningen klarer å utføre overordnet funksjonalitet eller enkelte hendelser.
- Supportability: hvor enkelt løsningen kan vedlikeholdes, oppgraderes og feilsøkes.

Alle funksjonelle og ikke-funksjonelle krav som ble inkludert i FURPS-analysen ble dessuten prioritert ved hjelp av MoSCoW metoden [46]. Dette er en teknikk ofte brukt i prosjektstyring for å avgjøre viktigheten av krav eller funksjoner med hensyn på prosjektets mål. Selve ordet MoSCoW basere seg på de fire prioriteringsnivåene som teknikken går ut på: «Must have», «Should have», «Could have» og «Won't have».

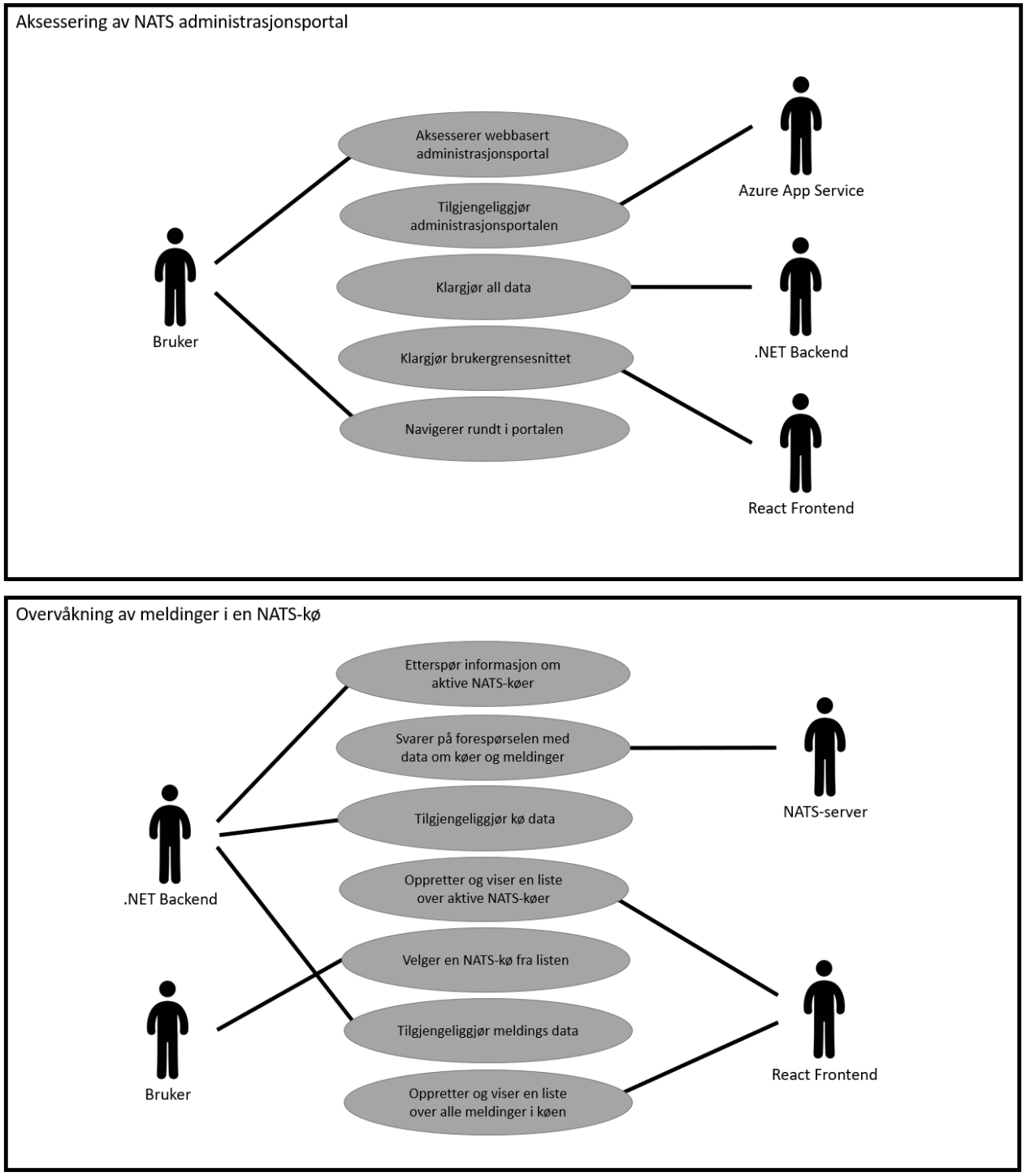
Funksjonalitetskrav som var definert med «Must have» eller «Should have» skulle inkluderes i administrasjonsportalen. «Could have» kravene kunne potensielt inkluderes i utviklingen, hvis tiden mot slutten av prosjektet tillot det. Krav som var definert med «Won't have» skulle ikke inkluderes i dette prosjektet, men skulle tas med til videreutvikling av NATS administrasjonsportalen. Se vedlegg D for en oversikt over hele FURPS-analysen inklusive prioritering av funksjonalitetskravene.

3.3.2 Use-case diagrammer

Use-case diagrammer [47] er en visuell representasjon av et systems funksjonelle krav. Slike diagrammer viser relasjonene og sammenhengene mellom ulike aktører, systemet og prosessene som foregår mellom dem. En aktør kan både være en fysisk sluttbruker eller bare en digital tjeneste som for eksempel NATS-server.

Årsaken til at Use-case diagrammer brukes i utviklingsprosjekter, og ofte inkluderes i kravspesifikasjoner, er fordi de hjelper med å danne et klart og tydelig bilde av hva løsningen

må kunne gjøre. Dermed er dette artefakter som gir en fellesforståelse av løsningen til både produkteier, utviklere, prosjektstyrere også videre. Dessuten blir Use-case diagrammer også brukt i forbindelse med testing og validering av løsningen [48]. Som tidligere omtalt ble det også inkludert et sett med Use-case diagrammer i prosjektets kravspesifikasjon. *Figur 3-3* viser to av Use-case diagrammene, for resten se vedlegg E.



Figur 3-3: Use-case for aksessering av administrasjonsportal og overvåkning av meldinger

4 Design

Et av de viktigste målene med designfasen er å utforme en løsning som adresserer de etterspurte kravene fra produkteieren og eventuelle sluttbrukere. Videre skal det sørges for at løsningen designes slik at den er både brukbar, pålitelig, effektiv og ikke minst at den kan vedlikeholdes.

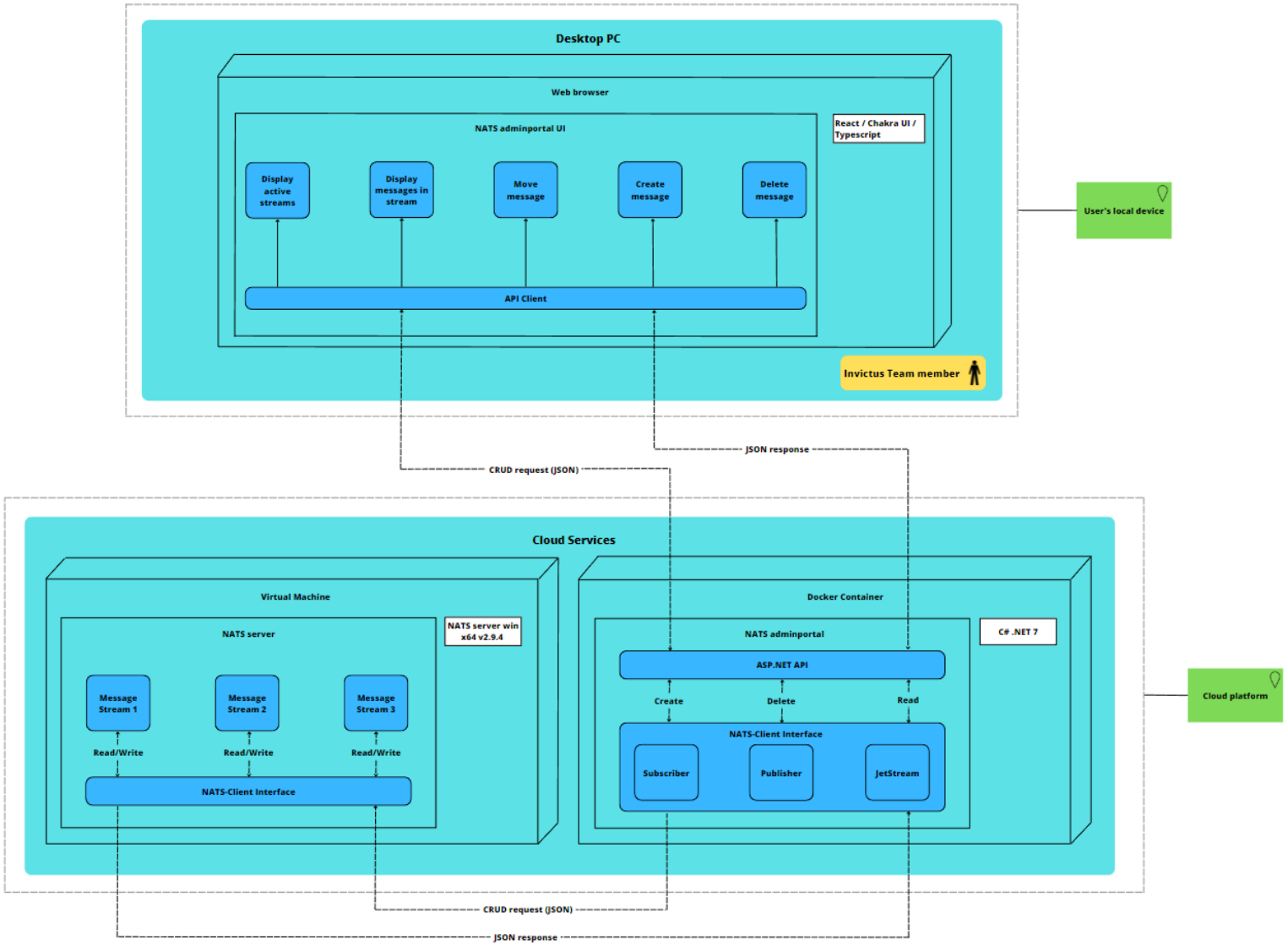
For å kunne komme fram til et brukbart systemdesign er kommunikasjon helt essensielt. Nærmere forklart må det være tett samarbeid mellom prosjektgruppen og produkteieren, i tillegg må begge partene sørge for å ha en felles forståelse [49]. Bare når disse tingene er på plass vil det kunne utvikles et system som tilfredsstiller alle funksjonelle og ikke-funksjonelle krav. Derfor er det veldig vanlig at det utarbeides en rekke med ulike konseptuelle tegninger, diagrammer og grafikker gjennom designfasen [50]. Dette er fordi visualisering er et sterkt verktøy i arbeidet for å skape felles forståelse blant ulike personer.

De påfølgende kapitlene vil ta for seg og forklare de mest sentrale elementene i designet bak NATS administrasjonsportalen. Alle tegninger, diagrammer og lignende ble brukt gjennom hele prosjektet for å vise Invictus hvordan systemet ville være satt opp og se ut.

4.1 Systemarkitektur

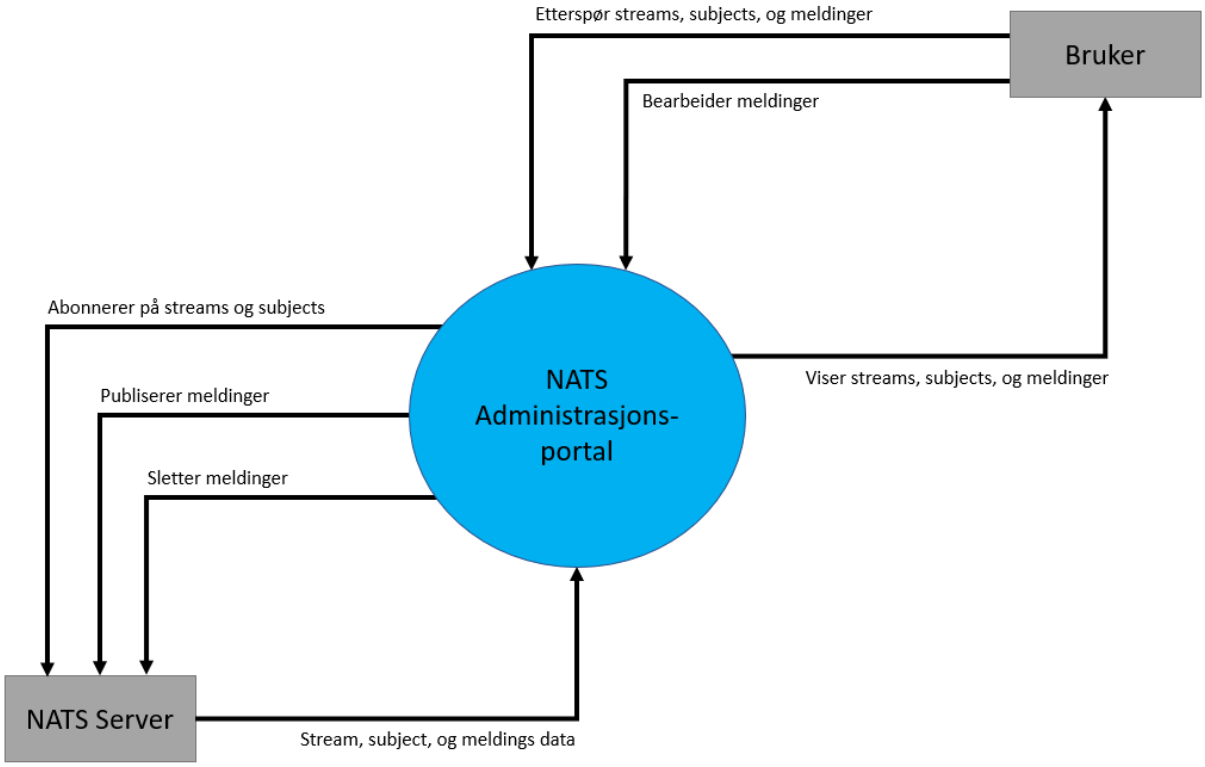
For å kunne demonstrere designet av NATS administrasjonsportalen på en enkel måte ble det utarbeidet et par arkitekturtegninger. Mer presist ble det opprettet en systemtegning og et kontekstdiagram. Disse to artefaktene ble opprettet tidlig i prosjektet og ble brukt til å presentere administrasjonsportalen til produkteierne. Tegningen ble også oppdatert underveis i prosjektet når den generelle systemarkitekturen endret seg basert på tilbakemeldinger fra Invictus.

Systemtegningen, vist i *Figur 4-1*, er en visuell presentasjon av de ulike byggeklossene som til sammen utgjør NATS administrasjonsportalen. Nærmere forklart viser tegningen alle de ulike komponentene, enhetene og teknologiene som inngår i løsningen, samt hvordan disse henger sammen.



Figur 4-1: Systemtegning

I motsetning til systemtegningen, gir kontekstdiagrammet en abstrakt overordnet oversikt over systemet og tilhørende miljø. Hensikten med denne oversikten er å identifisere eksterne enheter og aktører, og hvordan disse interagerer og kommuniserer med systemet, samt inputs og outputs mellom dem. Videre skal innblikket fra kontekstdiagrammet hjelpe med å identifisere det funksjonelle omfanget av systemet [51]. *Figur 4-2* viser kontekstdiagrammet til NATS administrasjonsportalen.



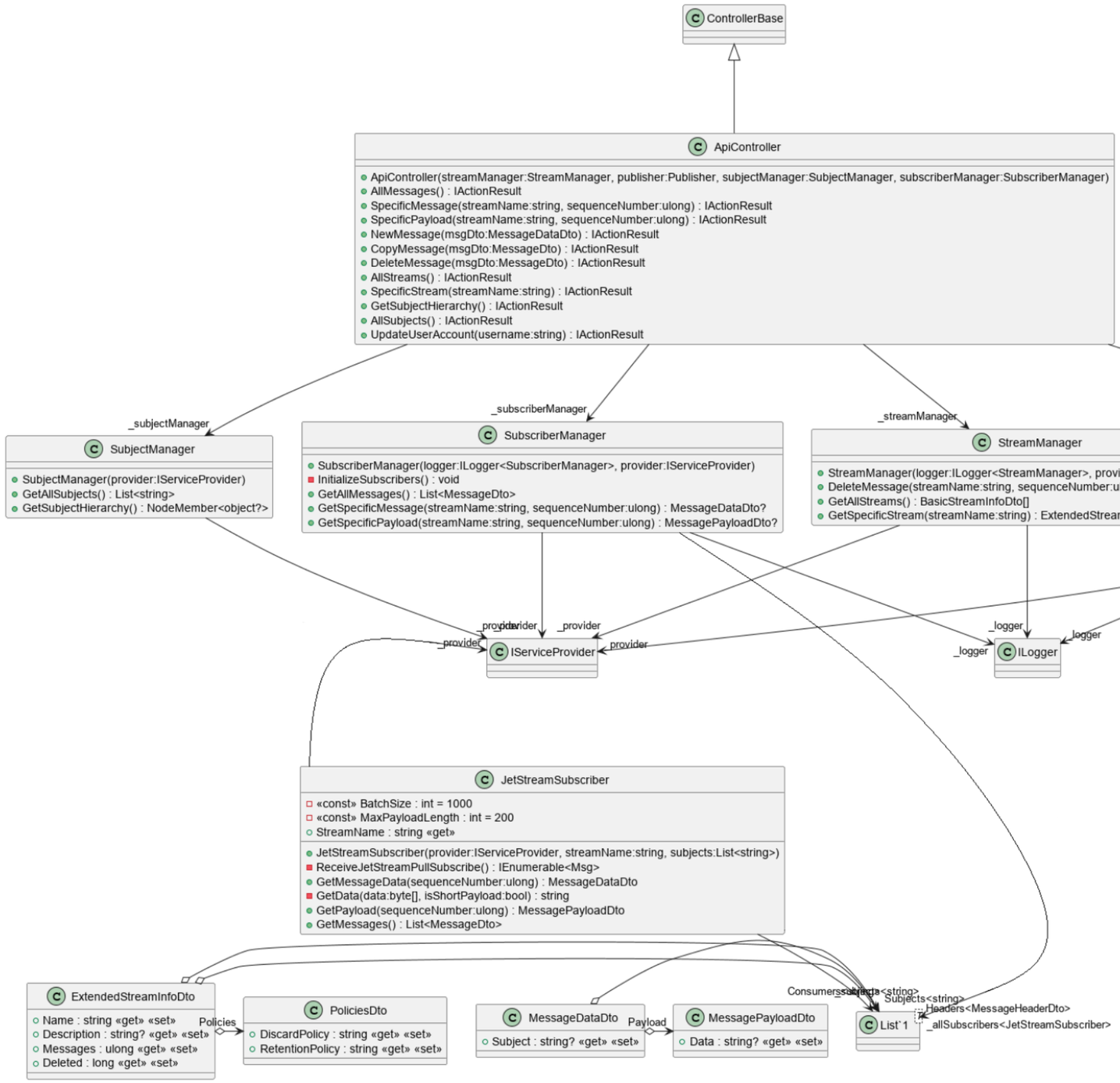
Figur 4-2: Kontekstdiagram

4.2 Klassediagram

Et klassediagram er en visuell representasjon av klasser med tilhørende attributter, metoder og deres parametere, samt relasjonen mellom ulike objekter i et system. Klassediagrammer er en sentral del av designfasen i de fleste utviklingsprosjektene som fokuserer på objektorientert programmering [52]. Dette skyldes at slike diagrammer hjelper med å identifisere eventuelle designfeil og forbedre den overordnede systemarkitekturen.

Under utviklingen av NATS administrasjonsportalen ble klassediagrammet, vist i *Figur 4-3* og *Figur 4-4*, brukt til å etablere en felles forståelse av oppsettet til backenden. Med andre ord viser diagrammet hvilke klasser som utgjør API-et til administrasjonsportalen, og hvilke relasjoner det har til C# NATS-Client [53] grensesnittet.

Figur 4-3: Klassediagram (del 1)



Figur 4-4: Klassediagram (del 2)

4.3 Brukergrensesnitt

Bortsett fra oppsettet av den generelle systemarkitekturen var også utformingen av brukergrensesnittdesignet en viktig del av designfasen. Per definisjon er brukergrensesnitt kontaktflaten som sluttbrukere har med et digitalt system. Normalt kan et UI (User Interface) enten klassifiseres som grafisk eller tekstlig [54]. Siden administrasjonsportalen skulle være webbasert, ble det opprettet et GUI (Graphical User Interface).

Arbeidet om å utforme et GUI startet med idemyldring og inspirasjonsinnsamling. Her var NATS-WebUI [55] og Apache Kafka GUI [56] to sentrale applikasjoner hvor mye inspirasjon ble hentet fra. NATS-WebUI er en webapplikasjon som muliggjør overvåkning av NATS-servere, og Apache Kafka GUI er en frittstående applikasjon for administrering av Apache Kafka-servere.

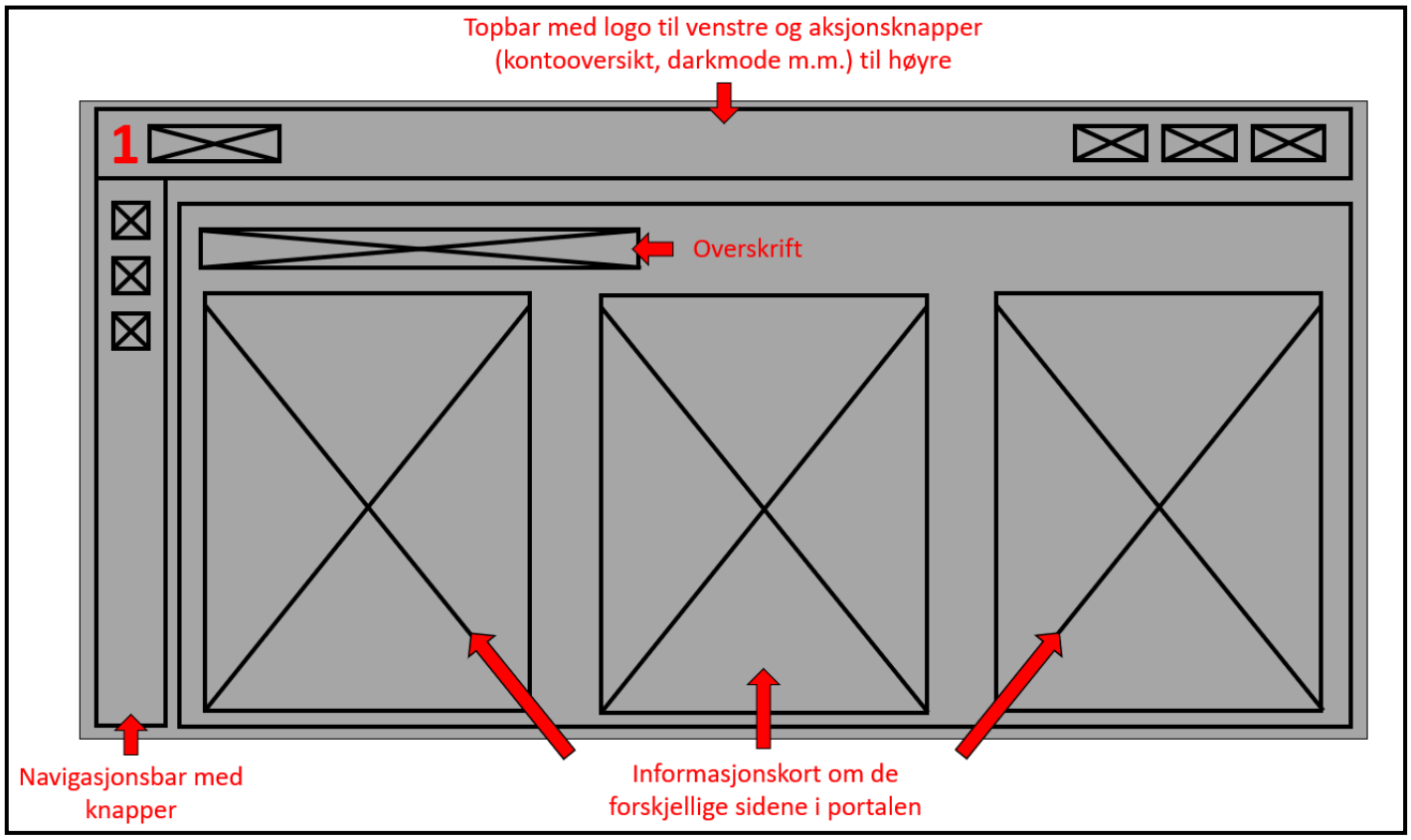
GUI-en gikk igjennom mange forandringer og forbedringer. Likevel endte sluttproduktet opp med å likne på wireframe-utkastene som ble utformet i starten av prosjektet. Navigeringen av brukergrensesnittet ble også planlagt med hensyn til navigasjonskartet. De neste kapitlene vil gå i detalj på disse tingene, i tillegg vil det også bli tatt opp hvordan GUI-en til NATS administrasjonsportalen er utformet i henhold til universell utforming.

4.3.1 Wireframes

Et av de aller første stegene i prosessen med å utforme et brukergrensnitt er opprettelsen av wireframes [57]. Wireframes er et sett med konsepttegninger som viser strukturen og oppbygningen av de ulike komponentene som skal utgjøre brukergrensnittet i et informasjonssystem. Dermed er dette et artefakt som visualiserer den grove utformingen av skjermbildene i det fullførte systemet.

Oppgaven til en wireframe er å vise alle objektene som skal være til stede på skjermen på et avgrenset område i løsningen. Objektene som vises i en wireframe skal ikke inneholde detaljer som forventes av et fullført design. Dette betyr at wireframes ikke fokuserer på ting som skrifttyper, farger, størrelser, nøyaktig plassering, ikoner og annen kosmetikk [58]. Årsaken til dette er at slike ting ofte vil gjennomgå mange forandringer i prosessen fram mot det endelige designutkastet.

Figur 4-5 viser en wireframe som ble utarbeidet i forbindelse med administrasjonsportalen. Totalt ble det opprettet seks individuelle wireframes som kan finnes i vedlegg F.



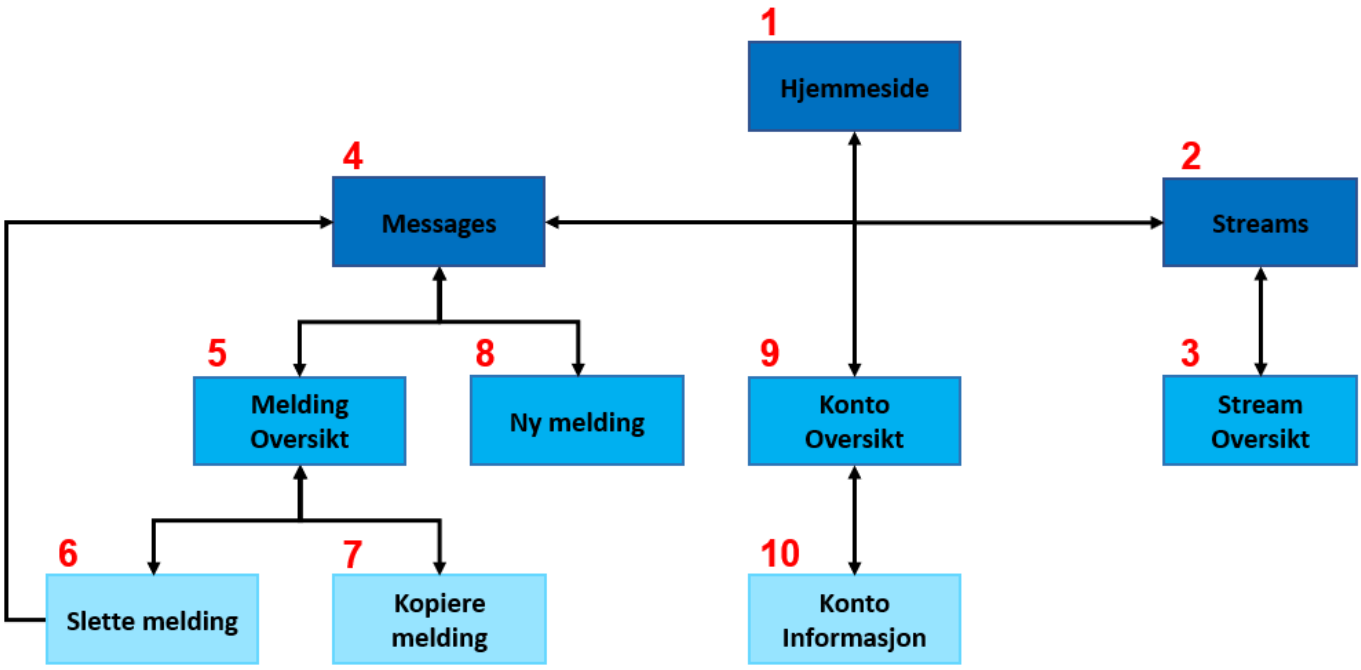
Figur 4-5: Wireframe for hjemmesiden

4.3.2 Navigasjonskart

Et annet artefakt som ble opprettet i forbindelse med designet av administrasjonsportalen var et navigasjonskart [58]. Dette er et verktøy som hjelper utviklere med å visualisere hvordan sluttbrukere kan navigere seg fram i et system eller en applikasjon. Dermed gir navigasjonskartet en oversikt over hvordan sluttbrukeren vil oppleve og oppfatte løsningen.

Hver boks i et navigasjonskart symboliserer et eget skjermbilde, og pilene som forbinder disse skjermbildene viser flyten mellom dem. Det er hensiktsmessig å vise både framover og bakover navigering ved hjelp av pilene, slik at man raskt kan identifisere blindveier som brukere kan ende opp i.

Navigasjonskartet som ble utformet i samspill med wireframes er vist i *Figur 4-6*. Designet til NATS administrasjonsportalen baserer seg på tre ulike nivåer med hjemmesiden og sidene for messages og streams i toppen. Det er gjensidig navigasjon mellom disse sidene og kontooversikten. Neste nivå inneholder en rekke med ulike modaler og oversikter som bare kan aksesseres fra sine respektive sider. På det dypeste nivået finnes det noen flere modaler som bare kan aksesseres fra den overordnete oversikten. Merk at tallene i navigasjonskartet referer til et wireframe, som gjør det lettere å se sammenheng mellom kartet og wireframes.



Figur 4-6: Navigasjonskart

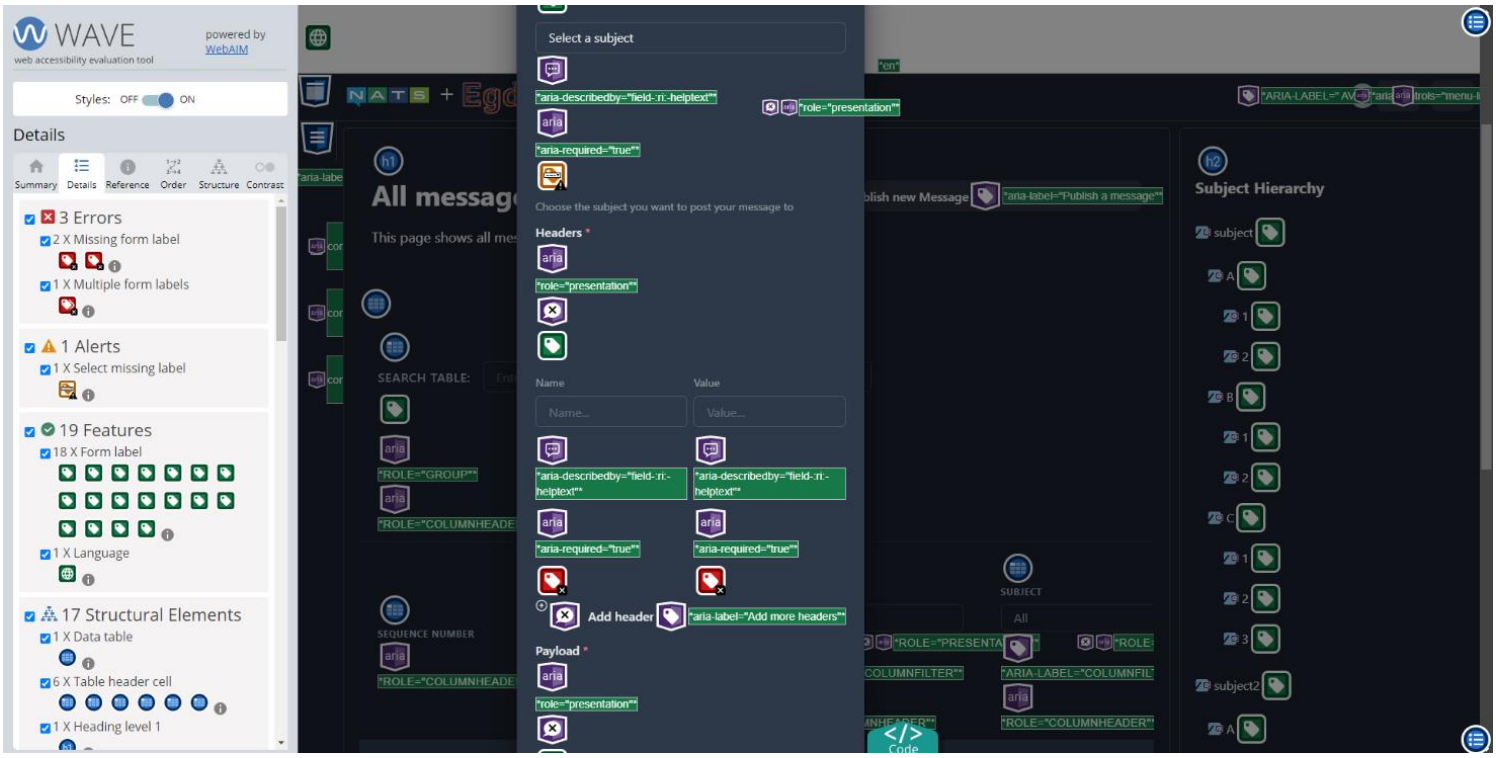
4.3.3 Universell utforming

Universell utforming (UU) [59] innebærer å utforme produkter og tjenester slik at de kan brukes av personer uavhengig av deres psykiske og fysiske evner. Dette betyr at UU tilrettelegger for at produkter og tjenester også kan brukes til fullt potensial av brukere med nedsatte funksjonsevner. Hensikten med dette er å unngå diskriminering av enkelt personer som for eksempel sliter med dårlig syn eller hørsel, språklige vansker, kulturelle eller politiske forskjeller og mye mer.

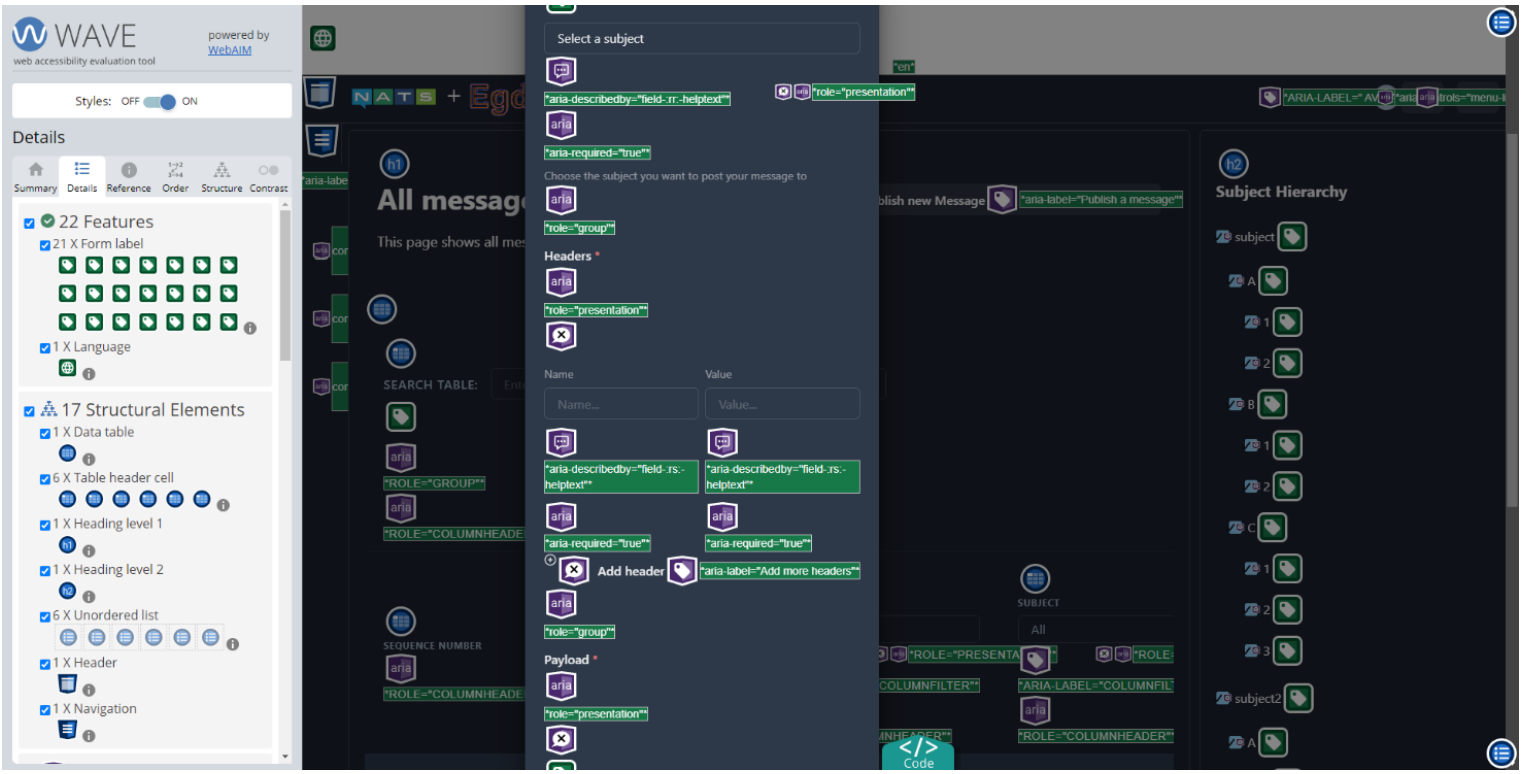
Arbeidet med å utforme et digitalt system med hensyn til universell utforming kan organiseres på ulike måter med varierende grad av suksess. For å garantere en god implementasjon av universell utforming finnes det WCAG [60], som er en internasjonal standard som beskriver hvordan universal utforming kan implementeres i webbaserte løsninger.

Siden et av kravene i prosjektet var å opprette et brukergrensesnitt som la til rette for grunnleggende UU, ble det tatt hensyn til WCAG under hele designfasen. Dette betydde at designet for brukergrensesnittet ble utformet med fokus på fargekontrast, skalerbarhet, utheving, tilbakemelding, støtte for skjermleser og mer.

For å verifisere at designet faktisk overholdt UU-kravene, ble WAVE-verktøyet [61] brukt. Dette er et evalueringsverktøy som sjekker at webbasert innhold er utformet på en måte slik at personer med nedsatte funksjonsevner også kan bruke applikasjonen uten hindringer. Selve WAVE-verktøyet er en nettleserutvidelse som kan åpnes direkte på nettsider og som deretter gir tilbakemelding om designfeil eller eventuelle forbedringer, se *Figur 4-7* og *Figur 4-8*.



Figur 4-7: WAVE-utvidelsen med feil



Figur 4-8: WAVE-utvidelsen uten feil

5 Implementasjon

Etter planleggings- og designfasen kommer implementasjonsfasen der selve applikasjonen eller løsningen opprettes basert på input fra de forrige fasene. Implementeringsfasen starter vanligvis med oppsett og konfigurasjon av et utviklings- og produksjonsmiljø [62]. Dette er viktig fordi all utvikling og publisering av kildekode må foregå på en strukturert og veldefinert måte. Et annet viktig steg i implementasjonsfasen er valget av tech-stack. Her må utviklingsteamet velge hvilke teknologier, rammeverk og verktøy som skal brukes for å bygge opp applikasjonen. Bare en godt utvalgt tech-stack vil sørge for at applikasjonen har god funksjonalitet, ytelse og kompatibilitet [63].

NATS administrasjonsportalen skulle være en full-stack applikasjon. Derfor måtte oppsett av utviklingsmiljø og valg av tech-stack ta hensyn til både front- og backend. De neste kapitlene vil forklare hvordan løsningen rundt administrasjonsportalen ble satt opp.

5.1 Verktøy

Programvareutvikling krever bruk av ulike verktøy for å sikre at implementasjonen av kildekoden skjer på en effektiv og kvalitativ høyverdig måte. Valg av verktøy bør skje etter at teknologier og rammeverk for prosjektet har blitt bestemt. Slik vil bare verktøy med rett funksjonalitet bli brukt [64].

De neste kapitlene vil gi en oversikt over verktøyene som ble brukt i dette prosjektet under implementasjonsfasen.

5.1.1 Git

Git [65] er et distribuert versjonskontrollsystem lagd for alle typer utviklingsprosjekter som inkluderer kildekode på en eller annen måte. Det at Git er et distribuert system betyr at all kildekode er lagret på et sentralt repository, samtidig som at det kan eksistere kloner av dette oppbevaringsstedet som fungerer som egne og uavhengig repository.

Hensikten med versjonskontrollsystemer er å kontrollere og sikre at nye endringer i kildekoden i et utviklingsprosjekt ikke kolliderer med eksisterende kode, altså for å unngå

merge-konflikter. Denne funksjonaliteten sørger for at flere utviklere kan jobbe parallelt med samme kode. Siden målet med prosjektet var å produsere en fullstack-applikasjon, var det helt essensielt å organisere utviklingsprosessen på en effektiv måte. Derfor ble Git brukt under prosjektet, og i dette tilfelle ble GitHub brukt som leverandør av versjonskontrollsystemet.

5.1.1.1 Trunk-modellen

Siden utviklingen av NATS administrasjonsportalen skulle foregå ved hjelp av Git, måtte det velges en passende modell for organisering av arbeidet mot kildekode-repository. Modellen som ble brukt i dette prosjektet, var Trunk-modellen [66].

Trunk-modellen definerer en rekke med metoder som angir hvordan opprettelse og merging av ulike branches i et Git-repository skal foregå. Modellen baserer seg på konseptet om å ha en trunk i form av en hoved-branch, der den mest oppdaterte og stabile versjonen av kildekoden ligger. Ideen bak Trunk-modellen er at selve utviklingen skal foregå på funksjons-brancher som kommer ut av hoved-branchen. På disse funksjons-branchene skal enkelte funksjoner først implementeres i kildekoden før de deretter kan merges med hoved-branchen. Denne organiseringen sørger for en mer kontrollert og effektiv utviklingsprosess, samtidig som håndtering av merge-konflikter og bugs er mer oversiktlig.

Årsaken til at håndtering av merge-konflikter er mer lettvint med Trunk-modellen, henger sammen med at utviklingen bare foregår i små deler. Dette sørger for at omfanget på potensielle konflikter ikke blir for stort. Dessuten sørger strukturen med funksjons-branches også for at opphavet til feil i kildekoden lett kan identifiseres. Når en bug oppdages, må denne være knyttet til kildekodeendringer som ble introdusert av en nylig merge mellom hoved-branch og en funksjons-branch.

En annen grunn til hvorfor Trunk-modellen ble valgt for dette prosjektet, er fordi den er velegnet for et mindre utviklingsteam. Dette henger sammen med at denne modellen tilrettelegger for isolert og parallell utvikling, som igjen fører til effektiv ressursutnyttelse [67]. Dette var helt nødvendig med tanke på størrelsen av prosjektgruppen.

5.1.2 Visual Studio Code

Visual Studio Code [68], også kjent som VS Code, er en kodeeditor utviklet av Microsoft som er åpen kildekode. Dette betyr at programmet er gratis og tilgjengelig på de vanligste plattformene. Rett ut av boksen kommer VS Code med støtte for debugging, innebygd Git kontroll, syntaksutheving, intelligent kodefullføring (IntelliSense) og refactoring. I tillegg til dette kan funksjonaliteten og støtte for flere programmeringsspråk legges til i VS Code gjennom en rekke ulike utvidelser. Dermed kan denne editoren tilpasses til bruk i mange ulike programmeringssammenhenger inklusiv webutvikling, vitenskapelig databehandling, data analyse og mye mer.

I dette prosjektet ble en standard installasjon av VS Code brukt for utvikling av frontend. For å utvide funksjonalitet og ansvarsområdene til VS Code ble det brukt en rekke utvidelser. Utvidelsene som ble installert i tillegg er som følger:

- C#: legger til støtte for .NET utvikling som inkluderer syntaksutheving, IntelliSense, ressursreferering med mer.
- C# XML Documentation Comments: forenkler opprettelse av standardisert C# klassedokumentasjon.
- HTML CSS Support: legger til støtte for CSS- og HTML- IntelliSense.
- NuGet Gallery: muliggjør enkel administrering av NuGet-pakker.
- Prettier - Code formatter: sørger for automatisk kode formatering av frontend programmeringsspråk som HTML, TypeScript, JSON, CSS med mer.

5.1.3 Rider

Rider [69] er et Integrated Development Environment (IDE) utviklet for .NET. Akkurat som andre IDE-er, tilrettelegger Rider for kodeanalyse, debugging, testing, IntelliSense, refactoring og kildekontroll med Git. Videre hjelper Rider utviklere med å skrive standardisert og effektiv C#-kode. En annen stor fordel med Rider over andre IDE-er som for eksempel Visual Studio, er krysskompatibilitet med tanke på operativsystem. Dette muliggjør utviklingen av applikasjoner på flere plattformer som Windows, macOS og Linux. På toppen av alt dette er Rider også kjent for kjapp ytelse, lavt minnebruk og et moderne brukergrensnitt.

Bruksområdene til Rider for dette prosjektet var utvikling av backend og testing. Dermed ble all C#-kode skrevet og feilsøkt i Rider, samt at all enhetstesting ble gjennomført ved hjelp av Riders innebygde funksjonalitet.

5.2 Oppsett av utviklings- og produksjonsmiljø

Med utgangspunkt i kravspesifikasjonen og bransjestandarder ble utviklings- og produksjonsmiljøet bygget opp av Docker [70] og GitHub. Selve utviklingen og lagring av all kildekode foregikk ved hjelp av GitHub, som forklart i kapittel 5.1.1, og publiseringen av applikasjonen fra utviklingsmiljøet til produksjonsmiljøet skjedde gjennom bruk av GitHub workflows.

GitHub workflows [71] er konfigurerbare og automatiserte prosesser som kan utføres på kildekoden i et repository. I denne konteksten betyr prosess et sett med jobber som utføres i sekvens. Jobbene som utgjør workflowen defineres i en YAML-fil (Yet Another Markup Language) ved hjelp av ulike steg som må gjennomføres for å fullføre en jobb. YAML-filen angir også konfigurasjonen i forbindelse med autentisering og når workflowen skal utføres [72].

Workflowen som ble opprettet i forbindelse med utviklingsmiljøet var satt opp til å kjøre hver gang det ble gjort endringer på kildekoden i hoved-branchen. Første steget i workflowen er kjøring av enhetstester, og andre steget er containerization av applikasjonen og opplastingen av Docker kontainerbildet til GitHubs kontainerregister [73]. Til slutt publiserer workflowen kontainerbildet ut til en Azure App Service. *Figur 5-1* viser YAML-filen som inneholder konfigurasjonen av den omtalte workflowen.

Containerization [74] er en teknikk brukt innenfor programvareutvikling for å pakke sammen kildekode, filer og eksterne biblioteker. Resultatet av innpakkingen er en kontainer som inneholder all nødvendig kildekode og konfigurasjon for å kunne kjøre applikasjoner på hvilke som helst operativsystem og infrastruktur. En ulempe ved krysskompatibilitet til kontainere er at lagringsstørrelsen ofte blir veldig stor. Dette henger sammen med at krysskompatibilitet krever mye ekstra konfigurasjonsdata og liknende. Løsningen på dette problemet er å bruke kontainerbilder som bare inneholder informasjon om hvordan kontaineren kan settes opp.

Slike kontainerbilder kan deretter brukes i produksjonsmiljøer for å sette opp og kjøre selve kontaineren som inneholder en applikasjon [75].

```yaml
jobs:
  test:
    runs-on: windows-latest
    steps:
      - uses: actions/checkout@v3

      - name: Start local NATS server
        working-directory: ${{ env.SERVER_PATH }}
        shell: pwsh
        run: |
          Start-Process -NoNewWindow -FilePath ".\nats-server.exe" -ArgumentList "-c .\server.conf"

      - name: Set up .NET Core
        uses: actions/setup-dotnet@v2
        with:
          dotnet-version: ${{ env.DOTNET_VERSION }}

      - name: run unit tests
        working-directory: ${{ env.DOTNET_PATH }}
        run: |
          dotnet test
  build:
    runs-on: ubuntu-latest
    needs: test
    steps:
      - uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Log in to GitHub container registry
        uses: docker/login-action@v1.10.0
        with:
          registry: ghcr.io
          username: ${{ github.repository_owner }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Lowercase the repo name and username
        run: echo "REPO=${GITHUB_REPOSITORY,,}" >>${GITHUB_ENV}

      - name: Build and push container image to registry
        uses: docker/build-push-action@v2
        with:
          push: true
          tags: ghcr.io/${{ env.REPO }}:${{ github.sha }}
          secrets: |
            "NATS_SERVER_URL=${{ secrets.NATS_SERVER_URL }}"
          file: ./Dockerfile
  deploy:
    permissions:
      contents: none
    runs-on: ubuntu-latest
    needs: build
    environment:
      name: "Development"
      url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}

    steps:
      - name: Lowercase the repo name and username
        run: echo "REPO=${GITHUB_REPOSITORY,,}" >>${GITHUB_ENV}

      - name: Deploy to Azure Web App
        id: deploy-to-webapp
        uses: azure/webapps-deploy@v2
        with:
          app-name: ${{ env.AZURE_WEBAPP_NAME }}
          publish-profile: ${{ secrets.AZURE_NATS_ADMINPORTAL_PUBLISH_PROFILE }}
          images: "ghcr.io/${{ env.REPO }}:${{ github.sha }}"

```

Figur 5-1: YAML-filen med workflow konfigurasjon

I dette prosjektet ble Docker brukt i forbindelse med opprettelse og kjøring av kontainere. Årsaken til dette var et krav fra Invictus, og fordi Docker er bransjestandarden når det kommer til håndtering av kontainere [76]. Alt som må til for å bruke Docker i samspill med et Git-repository er inkludering av en Docker-fil. Denne filen inneholder all informasjon som en GitHub workflow trenger for å konfigurere et kontainerbilde. *Figur 5-2* viser Docker-filen til NATS administrasjonsportalen.

```
# build .NET
FROM mcr.microsoft.com/dotnet/sdk:7.0 as build-net

WORKDIR /src

COPY /src/vite-api/vite-api.csproj .
RUN dotnet restore

COPY /src/vite-api .
RUN dotnet build -c Release
RUN dotnet publish -c Release -o /dist

# build vite
FROM node:latest as build-node

WORKDIR /src

COPY /src/vite-client/package.json .
RUN npm install

COPY /src/vite-client .
RUN npm run build
```

```
# copy results into production container
FROM mcr.microsoft.com/dotnet/sdk:7.0

WORKDIR /app

#RUN --mount=type=secret,id=NATS_SERVER_URL \
#    export NATS_SERVER_URL=$(cat /run/secrets/NATS_SERVER_URL)
#ARG SECRET_URL
#ENV NATS_SERVER_URL $SECRET_URL

ENV ASPNETCORE_ENVIRONMENT Production
ENV ASPNETCORE_URLS http://+:5000

EXPOSE 5000

COPY --from=build-net /dist .
COPY --from=build-node /src/dist /app/wwwroot

CMD ["dotnet", "vite-api.dll"]
```

Figur 5-2: Docker-fil

Produksjonsmiljøet for dette prosjektet var en Azure App Service. Avhengig av hvordan en App Service konfigureres, kan den brukes til å hoste en applikasjon ut fra et kontainerbilde. Dessuten er det verdt å nevne at Azure App Service klassifiseres som en PaaS (Platform as a Service) [35]. Dette betyr at Microsoft stiller med underliggende infrastruktur, programvare som inkluderer operativsystem og rammeverk, verktøy for analyse og vedlikehold, sikkerhet, lagringsplass med mer. Brukeren av tjenesten trenger derimot bare å ha kontroll over egen applikasjon og vedlikehold [77]. Som tidligere nevnt publiserer GitHub workflowen et kontainerbilde til Azure, og med dette bildet blir det satt opp og kjørt en kontainer. Denne kontaineren vil deretter tilgjengeliggjøre den nyeste versjonen av NATS administrasjonsportalen i produksjonsmiljøet.

5.2.1 JSON-server

Et av kravene til utviklingsmiljøet var uavhengighet mellom front- og backend utvikling. Mer spesifikt skulle frontenden kunne utvikles uten at backenden måtte utvikles samtidig. Siden frontend og backend normalt sett jobber sammen, der backenden tilgjengeliggjør data og frontenden presenterer denne til sluttbrukeren. Denne informasjonsflyten er illustrert i arkitekturtegningen i kapittel 4.1. For å oppnå kravet om uavhengig utvikling, ble det brukt en JSON-server.

JSON-server [78] er et verktøy som gjør at utviklere kan lage et kunstig API som bruker en JSON-fil som datakilde. Kunstig betyr i denne sammenheng at API-et kjører lokalt på utviklerens frontend-miljø, istedenfor for å ligge i backenden eller en ekstern server. Formålet til JSON-server er å muliggjøre enkel utvikling, testing og prototyping av applikasjoner.

JSON-filen som JSON-server baseres på fungerer som en database og inneholder all data som ellers hadde kommet fra et API. Serveren kan startes i utviklingsmiljøet med én enkel kommando og kan tilpasses etter behov. Den støtter alle CRUD-operasjoner og gir utviklere muligheten til å definere egne API-endepunkter.

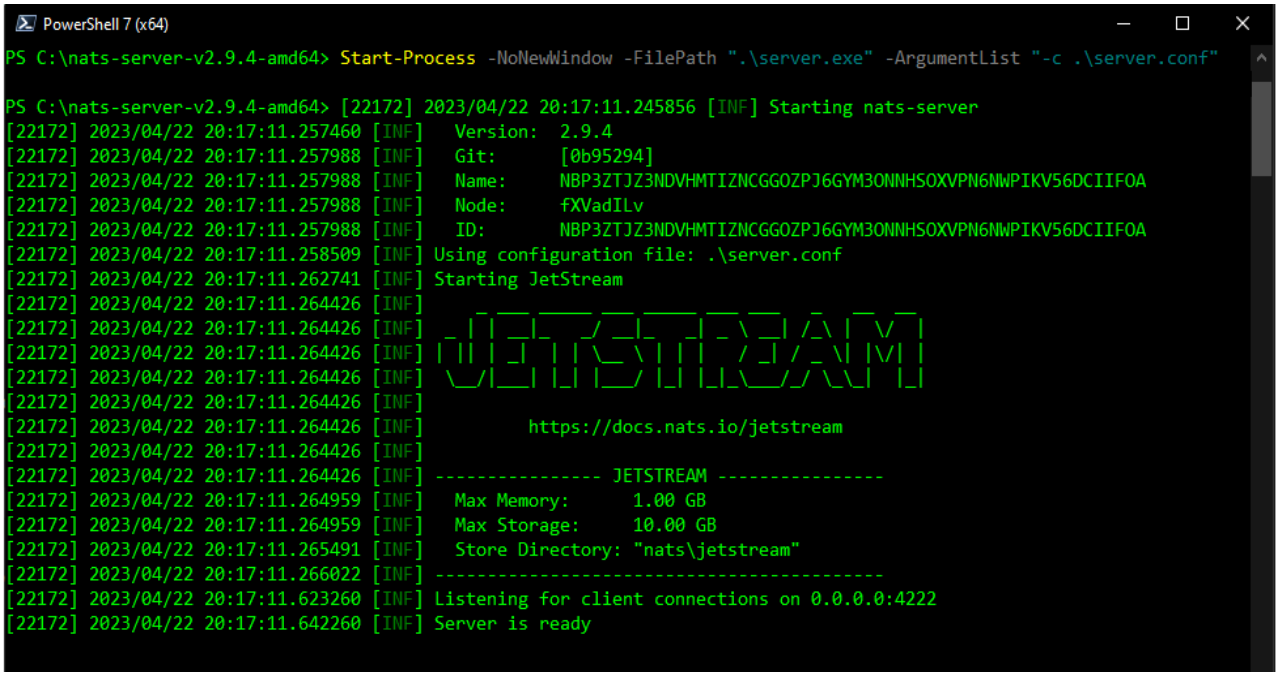
5.2.2 NATS-server

For å kunne utvikle og teste administrasjonsportalen var det nødvendig med en operativ NATS-server [79]. Oppgaven til NATS-serveren var å lagre test-data og svare på forespørsler fra NATS-applikasjon ved å levere test-data. En av Azure sine IaaS (Infrastructure as a Service) [80] ble brukt for å hoste en virtuell maskin som kjørte NATS-serveren [81]. IaaS er et samlebegrep for alle tjenester relatert til hosting av IT-infrastruktur i skyen. Typiske eksempler på IaaS er opprettelse og administrasjon av virtuelle nettverk med tilhørende brannmur, hosting og overvåkning av virtuelle maskiner eller servere, lagring og sikring av informasjon i databaser og mye mer.

Grunnen til at serveren kjørte i en virtuell maskin var for å etterligne Egde Health Gateway mest mulig. Denne plattformen er satt sammen av mange virtuelle servere ved hjelp av Kubernetes. Kort forklart er Kubernetes [82] et system for automatisering og håndtering av containerized applikasjoner og løsninger. I Egde Health Gateway finnes det også en rekke egne NATS-servere som betjener ulike endepunkter i plattformen, og planen er at NATS

administrasjonsportalen skal kommunisere med disse i fremtiden. Informasjonen som lagres på NATS-serveren i Egde Health Gateway er sensitiv, og derfor måtte det brukes en egen NATS-server under utvikling for dette prosjektet.

Det ble brukt en kjørbar applikasjon for å starte og kjøre NATS-serveren på den virtuelle maskinen. Nærmere forklart er applikasjonen en kjørbar fil utviklet av CNCF, som starter opp NATS-serveren i en lokal prosess [83]. For å konfigurere NATS-serveren på en effektiv måte, ble det brukt en konfigurasjonsfil som inneholdt informasjon om subjekter, minnebruk, støtte for JetStream, streams og mer. Støtte for JetStream var en viktig funksjonalitet som serveren måtte ha, siden kravspesifikasjonen etterspurte håndtering av streams. Måten NATS-serveren fikk tak i konfigurasjonsfilen var ved hjelp av en argument-parameter under oppstart. *Figur 5-3* viser hvordan NATS-serveren ble startet opp på den virtuelle maskinen ved bruk av Powershell.



Figur 5-3: Oppstart av NATS-serveren

5.3 Frontend

Frontenden i administrasjonsportalen er bygd opp av React [84] med Typescript [85], Vite [86] og Chakra-UI [87]. React er et populært JavaScript bibliotek som blir brukt til å bygge brukergrensesnitt basert på komponenter. Dette er små og gjenbrukbare moduler som settes sammen for å danne et brukergrensesnitt. Avhengig av mengden komponenter som benyttes

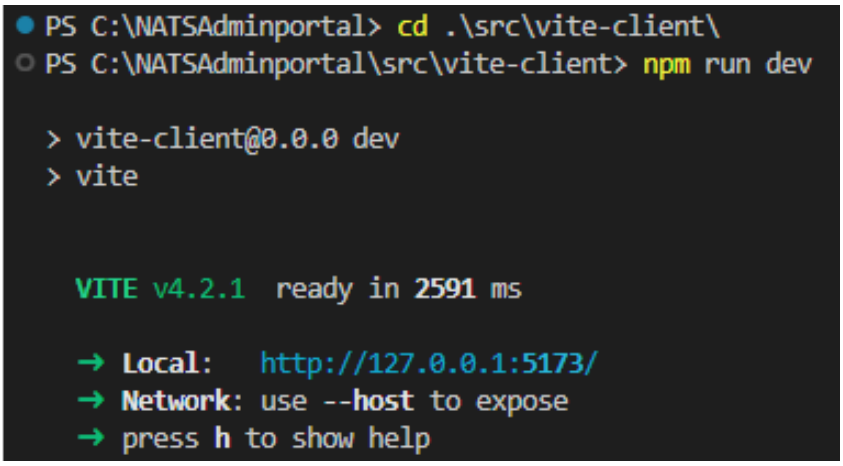
og hvordan disse samarbeider vil kompleksiteten til web-applikasjoner variere mye. En React-komponent ser ut som en metode, men i motsetning til vanlige metoder returnerer den en React-komponent sin HTML-kode. *Figur 5-4* viser en React-komponent som har blitt utviklet og brukt i NATS Administrasjonsportalen.

TypeScript [85] er et åpen-kilde programmeringsspråk som gir ytterligere syntaks til JavaScript. En av fordelene med TypeScript er at det krever angivelse av datatyper til variabler, parametere, funksjoner og mer. Dette hjelper både med å forebygge feil relatert til datatyper og lar brukere definere valgfrie parametere. I tillegg sørger TypeScript for at kode blir lettere å lese, vedlikeholde og feilsøke.

```
1  import { Flex, Spacer, useColorMode, Box, HStack } from "@chakra-ui/react";
2  import { ColorModeButton, AccountMenu } from "components";
3  import { EgdeLogo, NatsLetterIcon } from "assets";
4  import { AddIcon } from "@chakra-ui/icons";
5
6  function Topbar() {
7    const { colorMode } = useColorMode();
8    return (
9      <Flex
10       maxH={"100px"}
11       borderBottom={"1px solid"}
12       borderColor={colorMode === "dark" ? "whiteAlpha.300" : "gray.200"}
13       role={"banner"}
14     >
15       <HStack ml={3} w={"300px"}>
16         <NatsLetterIcon />
17         <Box>
18           <AddIcon mt={-3} />
19         </Box>
20         <EgdeLogo />
21       </HStack>
22       <Spacer />
23       <Box>
24         <ColorModeButton />
25       </Box>
26       <Box>
27         <AccountMenu />
28       </Box>
29     </Flex>
30   );
31 }
32
33 export { Topbar };
```

Figur 5-4: React-komponenten for Topbar

Vite [86] er et verktøy som hjelper ved byggingen av frontend-applikasjonen. Det er mange fordeler knyttet til bruken av Vite, den største er at den bygger applikasjoner lynraskt sammenliknet med andre verktøy. En annen fordel er at Vite støtter automatisk oppdatering av applikasjonen i nettleseren sekundet det blir utført endringer i kildekoden, ellers kjent som «hot reload» [88]. Sistnevnte skjer fordi Vite bruker en utviklingsserver som utnytter nettleserens innlastingsmodul. *Figur 5-5* viser oppstart av Vite servere ved å kjøre kommandoen «npm run dev» i terminalen.



Figur 5-5: Oppstart av frontend i Visual Studio Code

Videre er brukergrensesnittet til administrasjonsportalen basert på komponent-biblioteket Chakra-UI [87]. Chakra inneholder et sett med flere ferdiglagde React komponenter som følger de samme design-prinsippene. I tillegg er komponentene utformet med konsekvente fargetema noe som hjelper med å gi applikasjonen et gjennomført design. Chakra komponenter kan også enkelt modifiseres og tilpasses ved hjelp av properties. Et stort pluss med å bruke Chakra-UI, er at alle komponenter følger retningslinjene for UU [89]. Dette forenklet utviklingen av administrasjonsportalens brukergrensesnitt betydelig.

All frontend relatert kildekode ligger tilgjengelig i GitHub-repository som ble opprettet av prosjektgruppen [90].

5.4 Backend

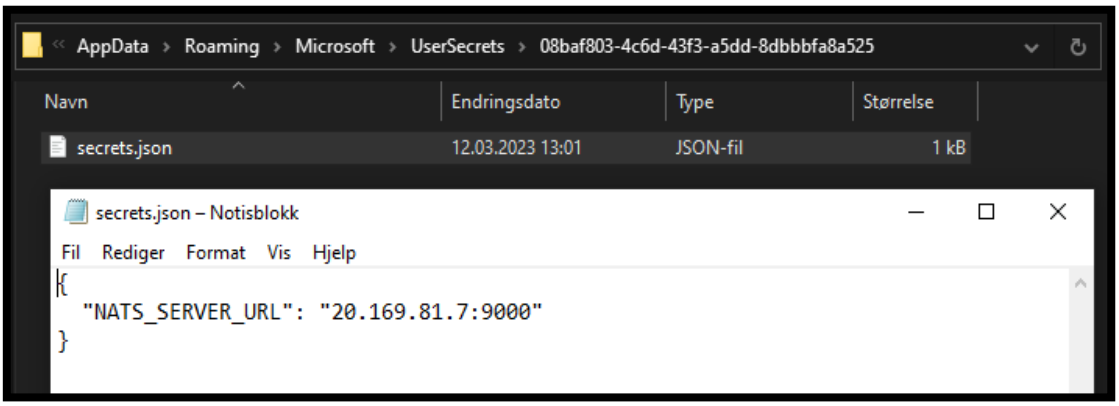
NATS-applikasjonens backend er bygd på .NET 7 rammeverket [91] og bruker ASP.NET MVC-modellen [92]. Videre blir NATS sitt offisielle C#-bibliotek [53] brukt for lette integrasjonen

mot NATS-servere. Dette biblioteket gjør det mulig å utføre CRUD operasjoner mellom administrasjonsportalen og NATS-serveren. Bruken av ASP.NET MVC sørger for en klar utskilling av modeller og kontrollere i backenden.

Backenden har en kontroller som er knyttepunktet mellom frontend og backend. Dermed foregår all kommunikasjon ved hjelp av forhåndsdefinerte endepunkter, og det brukes DTO-er (Data Transfer Object) for mellomlagring av dataen som blir sendt eller mottatt. Kontrolleren sender dataen til NATS-serveren fra modellene, definert i klassene, som igjen bruker NATS sitt C#-bibliotek for å kommunisere direkte med NATS-serveren.

Det er tatt i bruk .NET 7 siden dette er den nyeste versjonen av .NET rammeverket som er offisielt utgitt. Denne versjonen av .NET rammeverket tilbyr økt ytelse, mindre minnebruk og bedre integrasjon med andre teknologier som Docker og Kubernetes [91] [93]. Det siste punktet er viktig for Invictus.

Ettersom prosjektets GitHub repository skulle være åpent for alle, mer om dette i kapittel 7.1, var det viktig å skjule sensitiv informasjon fra kildekoden. Under utvikling ble sensitiv informasjon som for eksempel IP-en til NATS-serveren skjult ved hjelp av user-secrets. User-secrets inngår i Microsofts NuGet pakke «Microsoft.Extensions.Configuration» [94] og tillater applikasjonen å hente hemmelig informasjon fra lokal datamaskin. Dette betyr at hemmelig data ikke ligger åpent i koden, men lokalt på utviklerens datamaskin [95]. Forutsetningen er at utvikleren har konfigurert lagring av user-secrets på lokal datamaskin [96]. *Figur 5-6* viser lokal lagring av hemmelig data og *Figur 5-7* viser hvordan user-secrets er brukt i kildekoden.



Figur 5-6: Lagring av user-secrets på lokal datamaskin

```
using System.Text.Json;
using vite_api.Config;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;
using NATS.Client;
using vite_api.Classes;
using Options = NATS.Client.Options;

var builder = WebApplication.CreateBuilder(args);
builder.Configuration.AddUserSecrets<Program>();

builder.Services.AddOptions<AppConfig>().BindConfiguration("");
```

```
namespace vite_api.Config;

2 references | ...
public class AppConfig
{
    [ConfigurationKeyName("NATS_SERVER_URL")]
    1 reference
    public string? NatsServerUrl { get; set; }
}
```

Figur 5-7: Bruk av user-secrets i kildekoden

Det tidligere omtalte NATS C#-bibliotek ble også lagt til backend prosjektet i form av en NuGet pakke. NuGet pakker [97] er enestående samlinger med kode, filer og avhengigheter som gjør det mulig å legge til ny funksjonalitet i .NET prosjekter. Det med at pakkene er enestående betyr at de ikke er avhengig av andre pakker eller ekstern konfigurasjon for å fungere som tiltenkt. Måten NuGet pakker kan legges til, fjernes eller oppdateres i et .NET-prosjekt er gjennom NuGet pakkebehandler. Denne pakkebehandleren kan også brukes til å publisere egne prosjekter til .NET utviklingsplattformen slik at disse blir tilgjengelig som NuGet pakker for andre [98].

All backend relatert kildekode ligger tilgjengelig i GitHub-repositorien som ble opprettet av prosjektgruppen [99].

6 Testing

Prosessen å evaluere programvare, tjenester eller digitale systemer for feil eller mangler kalles for programvaretesting. Hensikten med testing er å sikre at kvaliteten av eksempelvis programvare er bevart. I denne sammenheng betyr kvalitet programvarens evne til å tilfredsstille de funksjonelle og ikke-funksjonelle kravene som sluttbrukeren eller kunde har satt [100].

Når testing utføres blir følgende elementer undersøkt med hensyn på kvalitet: design, brukbarhet, kompatibilitet, ytelse og feiltoleranse [101]. Selve testingen gjennomføres ved hjelp av forskjellige teknikker, og organiseres på ulike måter. Her er to mye brukte teknikker enhetstesting og UAT (User Acceptance Testing), siden de er både enkle og svært effektive. Dermed ble også disse to brukt i forbindelse med testing av NATS administrasjonsportalen. De etterfølgende kapitlene vil gjøre rede for hvordan gjeldende testing teknikker fungerer, og hvilke resultater de hadde i prosjektet.

6.1 User Acceptance Testing

User Acceptance Testing [102] er en form for manuell testing med formål å verifisere at programvare eller systemer tilfredsstiller kravene og forventingene til kunden. Denne typen testing gjennomføres vanligvis mot slutten av utviklingsprosessen når annen testing og kvalitetssikring allerede har blitt gjennomført.

Grunnlaget i all UAT er et sett med testsaker. Hver av disse tar opp et bruksscenario av programvaren eller systemet som skal testes. Scenarioene baseres normalt på de funksjonelle og ikke-funksjonelle kravene som kunden har definert. Når selve testen gjennomføres vil en sluttbruker gå gjennom en rekke steg som står forklart i de ulike testsakene. Etter at sluttbrukeren har fullført alle stegene må det avklares om testens faktiske resultat samsvarer med det forventede resultatet. Hvis resultatene er like kan gjeldende testsak ansees som godkjent, ellers skal avvik dokumenteres og tas med videre i utviklingsprosessen [103].

Tidligere ble det nevnt at UAT er manuell testing, og som forklart innebærer dette at mennesker, altså sluttbrukerne, gjennomgår applikasjonen for å avdekke eventuelle feil og

mangler. Slik type testing er veldig viktig, da den gjør det mulig å identifisere feil i bruksmønster, logisk systemoppsett, designvalg eller liknende.

Det ble gjennomført en UAT på administrasjonsportalen i slutten av fjerde sprint. I forkant av gjennomføringen ble det utarbeidet en liste med testsaker basert på kravspesifikasjonen, se vedlegg G. Under selve testen gikk noen teammedlemmer fra Invictus gjennom alle stegene som var definert i testsakene og ga tilbakemelding mens prosjektgruppen observerte. Tilbakemeldingene fra de individuelle testene ble også dokumentert i testsakene, igjen se vedlegg G. Målet for den femte sprinten var å forbedre kvaliteten på administrasjonsportalen basert på de dokumenterte tilbakemeldingene.

6.2 Enhetstester

Innenfor programvareutvikling er enhetstester automatiserte tester som sjekker at én bestemt funksjon eller metode fungerer riktig. Nærmere forklart undersøkes det at output fra en funksjon eller metode er som forventet basert på kjente inputverdier. Det at enhetstester er automatisert innebærer at det brukes egne rammeverk som maskinelt kjører en rekke med forhåndsdefinerte tester. Fordelene med automatisert testing overfor manuell testing, er at det er mindre tids- og ressurskrevende. På den andre siden er automatisert testing ikke like grundig som manuell testing. Ulempen med dette er at for eksempel enhetstesting ikke avdekker ting som logiske feil eller andre dyptliggende problemer [100].

Rammeverket som ble brukt i dette prosjektet var xUnit [104]. Årsaken til dette var at xUnit er ett av de fremste verktøyene innenfor enhetstesting av C# kode. Framgangsmåten ved skriving av enhetstester i xUnit er ganske standard, det vil si at det opprettes én eller flere fakta-metoder for hver metode som skal testes. Fakta-metoder har oppgaven med å undersøke om en påstand blir oppfylt av en gitt metode basert på kjente inputverdier. Forklart i mer detalj, brukes Assert-klassen for å sjekke at eksempelvis en verdi er sann eller usann, to verdier er like eller ulike, to objekter er identiske eller bare like [105]. Hvis påstanden i en fakta-metode ikke oppfylles vil gjeldende enhetstest ikke godkjennes, og dette vil xUnit varsle om.

Figur 6-1 viser hvordan en av fakta-metodene i NATS administrasjonsportalen ser ut, og *Figur 6-2* viser forskjellige resultater fra en enhetstest hvor testene både passerte og feilet.

All kildekode relatert til testing ligger tilgjengelig i GitHub-repository som ble opprettet av prosjektgruppen [106].

```
32      [Fact]
        0 references | Run Test | Debug Test
33      public void GetPayload_ReturnsSamePayload()
34      {
35          const ulong sequenceNumber = 1;
36          var subscriber = CreateDefaultJetStreamSubscriber();
37
38          var expectedPayload = _fixture.MsgDataDtos[(int)sequenceNumber - 1].Payload.Data;
39          var actualPayload = subscriber.GetPayload(sequenceNumber).Data;
40
41          Assert.Equal(expectedPayload, actualPayload);
42      }
43
44      [Fact]
        0 references | Run Test | Debug Test
45      public void GetPayload_ThrowsInvalidOperationException()
46      {
47          const ulong sequenceNumber = 100;
48          var subscriber = CreateDefaultJetStreamSubscriber();
49
50          void ActualAction() => subscriber.GetPayload(sequenceNumber);
51
52          Assert.Throws<InvalidOperationException>(ActualAction);
53      }
```

Figur 6-1: Kildekode til en fakta-metode

```
PS C:\NATSAdminportal\src\vite-api.Tests> dotnet test
  Determining projects to restore...
  All projects are up-to-date for restore.
  vite-api -> C:\NATSAdminportal\src\vite-api\bin\Debug\net7.0\vite-api.dll
  vite-api.Tests -> C:\NATSAdminportal\src\vite-api.Tests\bin\Debug\net7.0\vite-api.Tests.dll
Test run for C:\NATSAdminportal\src\vite-api.Tests\bin\Debug\net7.0\vite-api.Tests.dll (.NETCoreApp,Version=v7.0)
Microsoft (R) Test Execution Command Line Tool Version 17.4.0 (x64)
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.

Passed!  - Failed:     0, Passed:    24, Skipped:     0, Total:    24, Duration: 35 s - vite-api.Tests.dll (net7.0)
```

```
PS C:\NATSAdminportal\src\vite-api.Tests> dotnet test
  Determining projects to restore...
  All projects are up-to-date for restore.
  vite-api -> C:\NATSAdminportal\src\vite-api\bin\Debug\net7.0\vite-api.dll
  vite-api.Tests -> C:\NATSAdminportal\src\vite-api.Tests\bin\Debug\net7.0\vite-api.Tests.dll
Test run for C:\NATSAdminportal\src\vite-api.Tests\bin\Debug\net7.0\vite-api.Tests.dll (.NETCoreApp,Version=v7.0)
Microsoft (R) Test Execution Command Line Tool Version 17.4.0 (x64)
Copyright (c) Microsoft Corporation.  All rights reserved.

Starting test execution, please wait...
A total of 1 test files matched the specified pattern.
[xUnit.net 00:00:21.12]     vite_api.Tests.VerifyStreamManagerTests.DeleteMessage_ReturnsSameCount [FAIL]
  Failed vite_api.Tests.VerifyStreamManagerTests.DeleteMessage_ReturnsSameCount [10 s]
  Error Message:
   Assert.Equal() Failure
Expected: 4
Actual:   100
  Stack Trace:
     at vite_api.Tests.VerifyStreamManagerTests.DeleteMessage_ReturnsSameCount() in C:\NATSAdminportal\src\vite-api.Tests\VerifyStreamManagerTests.cs:line 36
   at System.RuntimeMethodHandle.InvokeMethod(Object target, Void** arguments, Signature sig, Boolean isConstructor)
   at System.Reflection.MethodInvoker.Invoke(Object obj, IntPtr* args, BindingFlags invokeAttr)

Failed!  - Failed:     1, Passed:    23, Skipped:     0, Total:    24, Duration: 25 s - vite-api.Tests.dll (net7.0)
```

Figur 6-2: Resultater fra enhetstesting. Øverst passerer alle tester og nederst feiler noen.

7 Diskusjon

Utviklingen av applikasjonen hadde stødig framdrift gjennom hele prosjektet. Ved å bruke agile utviklingsmetoder og scrum-rammeverket var det enkelt å fordele og bryte ned store oppgaver til mindre, mer overkommelige oppgaver. Kanban-brettet i GitHub hjalp mye i forbindelse med å holde oversikt over ferdigstilte og gjenstående arbeidsoppgaver. Bruk av GitHub ellers gjorde parallell utvikling av kildekoden veldig lett og godt organisert. Alle gruppemedlemmer hadde til enhver tid oversikt over hele kildekoden og endringer som hadde blitt gjort. Det må påpekes at hele applikasjonen, både frontend og backend, lå i én repository.

Det kan diskuteres om dette var det rette valget med tanke på organisering av kildekode. Et slikt Monorepo [107] har sine fordeler og ulemper, men i dette prosjektet fungerte denne løsningen utmerket. Spørsmålet om frontend og backend bør separeres i forskjellige repositories eller ikke, er et mye diskutert tema i programmeringsmiljøet [108].

7.1 Opphavsrett og Open Source

Med utgangspunkt i opphavsretten blir dataprogrammer ansett som åndsverk. Dette betyr at skaperen av et dataprogram har eneretten til å fremstille og distribuere eksemplarer av programvaren [109]. I de aller fleste bransjer er det ganske enkelt å følge opphavsrett, men innenfor programvareutvikling er ikke dette bestandig tilfelle. Årsaken til dette kan snevres ned til at mye «ny» kildekode allerede eksisterer fra før på en eller annen måte. Eneste forskjellen kan være at oppsettet i koden varierer, men logikken og fremgangsmetoder er helt like. Med andre ord vil nærmest all kildekode i to liknende dataprogrammer være like, noe som kan skape problemer med tanke på opphavsretten.

Gjennom dette prosjektet ble det utviklet kildekode som dannet grunnlaget for NATS administrasjonsportalen. Som tidligere omtalt tar kildekoden utgangspunkt i eksisterende løsninger (NATS-Client biblioteket) og derfor måtte det tas hensyn til riktig lisensiering av kildekoden for å ivareta opphavsretten. For å unngå potensielle konflikter i forbindelse med opphavsrett ble NATS administrasjonsportalen publisert med en Open Source lisens.

Det med at programvare er merket som Open Source [110], betyr at kildekoden er fritt tilgjengelig til både privatpersoner og virksomheter. «Fritt tilgjengelig» innebærer at brukeren kan få fullstendig innsyn i kildekoden, kan modifisere den, og har rett til å re-distribuere kildekoden.

7.1.1 Lisensiering

Å gjøre kildekode fritt tilgjengelig innebærer at en Open Source lisens knyttes til distribusjonen av koden. Nærmere forklart skal opphavsrettserklæringen til gjeldende Open Source lisens inkluderes på en eller annen måte i kildekoden. Bare når dette har blitt gjort kan sluttbrukere være 100% sikker på at kildekoden er Open Source. En ting som kan by på utfordringer i dette område er valg av lisensen, siden det finnes et titalls ulike Open Source lisenser.

Noen av de meste brukte og offisielt anerkjente Open Source lisenser er MIT, Apache 2.0, ISC, og BSD 3-Clause. Selv om opphavsretterklæringen i hver av disse lisensene er formulert på ulike måter går alle ut på det samme. Det vil si alle gir fri tilgang på kildekoden uten noen særlige restriktive krav [111].

I forbindelse med dette prosjektet ble MIT-lisensen [112] brukt for NATS administrasjonsportalen. Denne lisensen inkluderer ingen restriksjoner, samtidig som at eieren ikke gir noen garanti på kildekoden. Dermed fraskriver Egde seg alt lovlig ansvar som har med innholdet i løsningen å gjøre. MIT-lisensen er knyttet til administrasjonsportalen gjennom en tekstfil med informasjon om lisensen, og denne filen ligger i samme Git-repository som resten av kildekoden [113].

7.2 Kode gjennomgang

Halvveis gjennom utviklingen fikk prosjektgruppen tilbud om å gjennomgå produsert kode til frontend og backend sammen med erfarne utviklere. Backenden ble gjennomgått sammen med en backend utvikler fra Egde, Daniel Vassdal. Denne gjennomgangen gikk ut på at Daniel ga prosjektgruppen innspill og eksempler på forbedringer av eksisterende kode. Forbedringene ville øke kvaliteten og effektiviteten av koden, samtidig ville den også følge

god .NET koding etikette på et høyere nivå. Innføring av Daniels forbedringer krevde drastisk og tidskrevende omstrukturering av backenden, men det var verdt det.

Gjennomgang av frontenden ble gjennomført i samarbeid med prosjektets veileder, Martin Nenseth. Han er en fullstack utvikler med mye erfaring innenfor React-biblioteket, og han gav prosjektgruppen forslag på forbedringsområder i forbindelse med frontenden. Noen av de mest sentrale tilbakemeldingene var endring av mappestruktur, navnekonvensjoner og implementasjon av brukervennlig feilhåndtering. Mesteparten av de foreslåtte endringene ble implementert i frontend, og resten ble dokumentert for fremtidig videreutvikling. Alt i alt var også denne gjennomgangen svært lærerik og hjalp med å forbedre helheten av NATS administrasjonsportalen.

7.3 Forbedringer

Når det gjøres et tilbakeblikk på utviklingsløpet blir det åpenbart at noen ting kunne og burde ha blitt gjort annerledes. Eksempelvis kunne gjennomgangen av backend koden ha blitt utført på et tidligere stadium i utviklingsprosessen. Dermed ville man ha spart tiden som ble brukt på å omstrukturere backenden for å implementere drastiske endringer som påvirket oppsettet av kommunikasjon mellom frontend og backend.

Som diskutert i kapittel 5.2.1 ble det opprettet og konfigurert en JSON-server som kunne erstatte backenden. Dette ble gjort fordi Egde hadde hørt mye bra om JSON-server og ville at prosjektgruppen skulle teste ut verktøyet. I etterkant viste det seg at oppsett av JSON-server og tilhørende filer fylt med «mock»-data (kunstig data som simulerer meldinger, streams og annet NATS-server informasjon) tok tid. Det ville vært gunstig å implementere JSON-server tidligere i utviklingsprosessen. Dette skyldes at den fungerte som et utmerket alternativ til et fullverdig backend. Hvis dette hadde vært på plass kunne man ha utnyttet JSON-server bedre i utviklingen av frontend enn det som ble gjort.

En annen sak som må omtales er «GetAllMessages» metoden. Oppgaven til denne metoden er å hente alle meldingene som ligger på de ulike streams på NATS-serveren, og den gjør dette ved hjelp av temporære consumers. I utgangspunktet byr ikke dette på noen direkte problemer, men de temporære consumers setter spor etter seg i form av flytende antall

consumers som ligger på en stream. Med andre ord vises de temporære consumers i administrasjonsportalen på «Streams»-siden helt fram til de automatisk termineres. Dette er ikke ønskelig oppførsel, men opphavet til problemet ligger dypt i JetStream-logikken til NATS-biblioteket. Siden prosjektgruppen hverken hadde kunnskapen og ressursene til å finne en endelig løsning på problemet, ble det bare implementert visualisering av temporære consumers i frontend ved å tone ned tekstfargen på consumers, se *Figur 7-1*.

Figur 7-1: Visualisering av temporære consumers

Et av kravene for administrasjonsportalen var opprettelsen av enhetstester for metodene i backend. Det ble gjort et forsøk på å lage enhetstester, men det viste seg at det å «mocke» en NATS-server var veldig avansert, tidkrevende og dels umulig. Nærmere forklart er alle metodene i backend avhengig av en aktiv kobling med en NATS-server. Hvis dette ikke er på plass, vil metodene feile. Dermed var enhetstestene avhengig av å opprette en kunstig kobling mot en ikke-eksisterende NATS-server. *Figur 7-2* viser et vellykket forsøk på en mocked enhetstest, mens *Figur 7-3* viser mocking som ikke fungerer. Årsaken til at mocking i *Figur 7-3* ikke fungerer er fordi MetaData er en «sealed» klasse fra NATS biblioteket. Dette sørger for at klassen ikke kan arves fra og dermed kan Moq biblioteket [114] ikke brukes for å mocke den [115].

Løsningen på problemet med å ikke kunne mocke NATS biblioteket var bruk av en lokal NATS-server under enhetstestingen. Denne serveren blir startet opp i begynnelsen av enhetstestingen, og avsluttes når alle tester har kjørt. Ulempen med dette er at testene ligner mer på integrasjonstesting enn enhetstesting. Essensen i integrasjonstesting er testing av kommunikasjonen mellom en levende server og metoder i kildekoden ved å sammenligne returnerte og forventede verdier [116]. Alt dette betyr at testene som ble utviklet for administrasjonsportalen tester litt andre ting enn hva som var forventet.

```
18      private JetStreamSubscriber CreateDefaultJetStreamSubscriber()
19      {
20          var mockConnection = new Mock<IConnection>();
21          var mockJetstream = new Mock<IJetStream>();
22          var mockPullSubscribe = new Mock<IJetStreamPullSubscription>();
23          var mockMessages = new List<Msg> {new("Hei")};
24
25          mockPullSubscribe.Setup(x => x.Fetch(It.IsAny<int>(), It.IsAny<int>()))
26              .Returns<int, int>((batchSize, maxWaitMillis) => mockMessages);
27
28          var primarySubject = "Primary";
29          mockJetstream.Setup(x => x.PullSubscribe(It.IsAny<string>(), It.IsAny<PullSubscribeOptions>())).Returns(mockPullSubscribe.Object);
30          mockConnection.Setup(x => x.CreateJetStreamContext(It.IsAny<JetStreamOptions>())).Returns(mockJetstream.Object);
31          var streamName = "Name";
32
33          var serviceProvider = new Mock<IServiceProvider>();
34          serviceProvider
35              .Setup(x => x.GetService(typeof(IConnection)))
36              .Returns(mockConnection.Object);
37
38          return new JetStreamSubscriber(serviceProvider.Object, streamName, new List<string> {primarySubject});
39      }
```

Figur 7-2 - Mocked enhetstest

```
114     public virtual List<Msg> MockMessages()
115     {
116         var messages = new List<Msg>();
117
118         for (int i = 0; i < MsgDataDtos?.Count; i++)
119         {
120             var header = new MsgHeader { { MsgDataDtos[i].Headers.First().Name, MsgDataDtos[i].Headers.First().Value } };
121             var msg = new Mock<Msg>
122             {
123                 Object =
124                 {
125                     Subject = MsgDataDtos[i].Subject,
126                     Header = header,
127                     Data = Encoding.UTF8.GetBytes(MsgDataDtos[i].Payload.Data!)
128                 }
129             };
130
131             msg.SetupGet(x => x.MetaData.StreamSequence).Returns((ulong)i);
132             messages.Add(msg.Object);
133         }
134
135         return messages;
136     }
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

  Failed vite_api.Tests.VerifyJetStreamSubscriberTests.GetPayload_ReturnsSamePayload [1 ms]
  Error Message:
   System.AggregateException : One or more errors occurred. (Unsupported expression: ... => ....StreamSequence
Non-overridable members (here: MetaData.get_StreamSequence) may not be used in setup / verification expressions.) (The following constructor parameters
ixture)
---- System.NotSupportedException : Unsupported expression: ... => ....StreamSequence
Non-overridable members (here: MetaData.get_StreamSequence) may not be used in setup / verification expressions.
---- The following constructor parameters did not have matching fixture data: MockServerFixture fixture
  Stack Trace:

----- Inner Stack Trace #1 (System.NotSupportedException) -----
```

Figur 7-3 - Mocking som ikke fungerer

7.4 Videreutvikling av administrasjonsportalen

Målet for dette prosjektet var å lage en første versjon av en administrasjonsportal. Planen var dermed at applikasjonen skulle videreutvikles av Invictus i fremtiden. Dette ble det tatt hensyn til når utviklings- og produksjonsmiljøet ble satt opp og konfigurert. Bruken av GitHub workflows og Docker har og vil lette utviklingsarbeidet med hensyn på CI & CD (Continuous Integration & Deployment) [117]. Konseptet bak CI & CD er en automatisert prosess som bygger, tester og vedlikeholder kildekoden til en applikasjon. Dette betyr at utviklere bare kan legge til ny funksjonalitet eller gjøre endringer i kildekoden og CI & CD vil sørge for at endringene lagres og at eventuelle nye versjoner av applikasjonen bygges og publiseres.

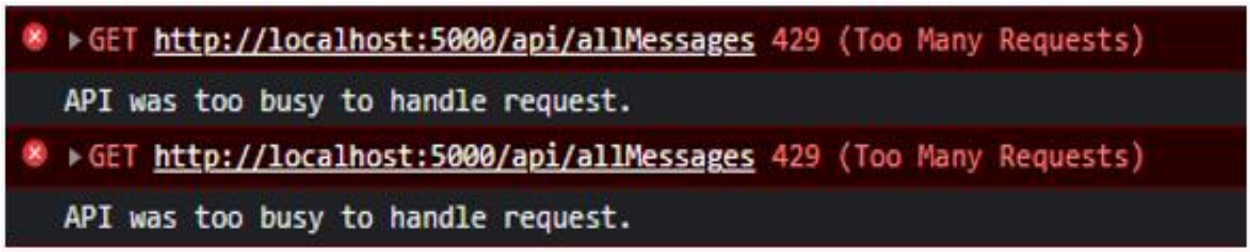
Kildekoden er an annen ting som ble utformet på en måte som gjør det enkelt å videreutvikle applikasjonen. Det ble fulgt navnekonvensjoner for de forskjellige kodespråkene, det ble lagt inn klassedokumentasjon og kildekoden ble organisert med god filstruktur. Disse tingene sørger for at all kildekode er lettleselig og oversiktlig. Videre ble kildekoden utformet til å ha lav coupling og høy cohesion.

Coupling angir hvor mye de ulike modulene eller klassene i kildekoden er avhengig av hverandre. Å ha høy coupling er generelt dårlig siden enkelte endringer i en modul vil kreve endringer i en annen modul grunnet avhengighet, som betyr mye ekstra arbeid. Cohesion er en målestokk på hvor enestående enkelte moduler (kildekode) er. Det vil si at klasser med veldefinerte mål har høy cohesion, siden de bare inneholder metoder relatert til målet. Fordelen med dette er at kildekoden blir lettere å vedlikeholde [118].

Foruten om krav merket med «Could have» og «Won't have» i kravspesifikasjonen, er implementasjon av autorisering og autentisering de neste stegene i videreutviklingen. Dette må være på plass før Egde kan integrere administrasjonsportalen i egen skyplattform. Ellers vil hvem som helst kunne bruke administrasjonsportalen til å lese sensitiv data.

Ved bruk av applikasjonen kan det forekomme at enkelte API-requests feiler som vist i *Figur 7-4*. Feilmeldingene som oppstår på grunn av dette fører ikke til programfeil i applikasjonen og de oppstår veldig sjeldent. Det trengs ikke å finne en løsning på feilende API-requests, men måten feilmeldingene håndteres må forbedres. Per nå logges alle feilmeldinger i forbindelse med API-requests til konsollet i nettleseren. Dette er ikke brukervennlig med tanke på at

sluttbrukeren ikke blir varslet om at slike feil har oppstått. Derfor må korrekt varsling utvikles og implementeres framover.



Figur 7-4: Feilmelding som oppstår når API-et er "opptatt"

En annen viktig ting som må implementeres i NATS administrasjonsportalen er håndtering av kryptert data. Dataen som eksisterer i Egde Health Gateway er sensitiv og blir derfor sikret ved hjelp av kryptering. Slik kan det unngås at utenforstående kan lese dataen, men dette betyr også at NATS meldinger er uleselige for momentan versjon av administrasjonsportalen. Grunnen til at det ikke ble tatt hensyn til kryptering i dette prosjektet, var fordi prosjektgruppen ikke hadde tilstrekkelig med kunnskap og Egde ville håndtere dette selv i fremtiden.

8 Konklusjon

I løpet av dette bachelorprosjektet ble det utviklet en administrasjonsportal for å behandle meldinger for Egde sin helseplattform ved hjelp av NATS-teknologien. Dette var et ønske fra Invictus ettersom de ikke hadde en god måte å håndtere meldinger og streams på.

Å utvikle en full-stack applikasjon er en krevende oppgave. Både utviklingsprosessen og prosjektorganiseringen rundt må kartlegges grundig og gjennomføres som planlagt for at sluttproduktet skal bli en suksess. Produkteieren ga uttrykk for at full-stack applikasjonen som ble utviklet i løpet av dette prosjektet oppfylte alle etterspurte krav.

Valget om å strukturere utviklingsprosessen på en agil måte var en av suksessfaktorene i prosjektet. Denne prosjektorganiseringen tilrettela for kontinuerlig reevaluering og kvalitetssikring av den nyeste utviklede versjonen av NATS administrasjonsportalen. Dermed fokuserte videreutviklingen alltid på krav og tilbakemeldinger, slik at prosjektet ikke sporet av i feil retning.

Grundig forarbeid i prosjektets oppstartsfase var en annen suksessfaktor. Her var arbeidet med å finne relevant referanselitteratur, oppsett av utviklings- og produksjonsmiljø, valg av tech-stack og systemdesign viktige oppgaver. Alle disse tingene sørget for at prosjektgruppen var godt rustet til å iverksette utviklingsarbeidet av selve applikasjonen.

Et annet sentralt aspekt ved utviklingsprosessen var tett samarbeid med sluttbrukerne, Invictus. Dette samarbeidet sørget for at prosjektgruppen alltid fikk tilstrekkelig med tilbakemeldinger i forbindelse med utviklingsprosessen. Tilbakemeldingene som ble innhentet fra Invictus fokuserte på hvorvidt applikasjon tilfredsstilte kravene og hvordan den var utformet designmessig.

Invictus og Egde generelt har vært en strålende samarbeidspartner i dette prosjektet. Alle som var involvert hadde en visjon om hva administrasjonsportalen skulle være og burde kunne gjøre. Samtidig var det tett oppfølging fra Egde i forbindelse med utviklingsprosessen ved hjelp av sprint reviews. Disse tingene sørget for at prosjektet hadde et veldefinert mål, samt at prosjektgruppen alltid fikk hjelp og svar på spørsmål.

Alt i alt var dette et vellykket bachelorprosjekt hvor alle gruppemedlemmene fikk samlet verdifull erfaring og videreutviklet ferdighetene sine. På grunn av jevn fremdrift og godt

samarbeid innad i gruppen oppstod det heller aldri problemer, og uventede utfordringer ble løst på en effektiv måte. Det kan konkluderes med at bachelorens problemstilling ble grundig undersøkt og at det ble utviklet en løsning som Invictus var fornøyd med. Allikevel har Invictus en liten jobb å gjøre med å få implementert tidligere omtalte forbedringer og endringer. Her bør videreutviklingen fokusere på å finne permanente løsninger på problemet med temporære consumers og mocking av NATS biblioteket i forhold til testing. I tillegg må manglende funksjonalitet fra kravspesifikasjonen legges til administrasjonsportalen.

Referanser

[1] Kommunenes Sentralforbund (KS), «Kommunal sektors ambisjoner på e-helseområdet,» 03. Mai 2022. [Internett]. Tilgjengelig: https://www.ks.no/fagomrader/digitalisering/felleslosninger/digitalisering-i-helse-og-omsorgsektoren-e-helse/om-e-helse/. [Funnet 04. Mars 2023].

[2] Egde, «Vi forener mennesker og teknologi,» [Internett]. Tilgjengelig: https://egde.no/vi-jobber-med/. [Funnet 03. Mai 2023].

[3] Egde, «Egde Health Gateway,» [Internett]. Tilgjengelig: https://egde.no/product/egde-health-gateway/. [Funnet 07. Mars 2023].

[4] NATS, «About NATS,» NATS, Oktober 2022. [Internett]. Tilgjengelig: https://nats.io/about/. [Funnet 03. Mai 2023].

[5] Egde, «Kontakt oss,» [Internett]. Tilgjengelig: https://egde.no/kontakt/. [Funnet 04. Mai 2023].

[6] K. Sander, «Prosjektmodell,» 23. Januar 2020. [Internett]. Tilgjengelig: https://estudie.no/prosjektmodell/. [Funnet 09. Mai 2023].

[7] S. Ullsfoss, «Smidig vs. tradisjonell prosjektmetodikk - hva er best?,» [Internett]. Tilgjengelig: https://sprint.no/artikler/smidig-vs-tradisjonell-prosjektmetodikk-hva-er-best. [Funnet 08. Februar 2023].

[8] D. DeClute, «Agile vs. Waterfall: What's the difference?,» 15. September 2022. [Internett]. Tilgjengelig: https://www.theserverside.com/tip/Agile-vs-Waterfall-Whats-the-difference. [Funnet 27. April 2023].

[9] Holte Academy, «Agil eller fossefall – hvilken metode skal vi velge?,» [Internett]. Tilgjengelig: https://www.holteacademy.no/agil-eller-fossefall-hvilken-metode-skal-vi-velge/. [Funnet 28. April 2023].

[10] Scrum.org, «What is Scrum?,» [Internett]. Tilgjengelig: https://www.scrum.org/resources/what-is-scrum. [Funnet 08. Februar 2023].

[11] D. Misevičiūtė, «Scrum Team Size: Easy Steps to Create Ideal Team,» [Internett]. Tilgjengelig: https://teamhood.com/agile/scrum-team-size/. [Funnet 08. Februar 2023].

[12] Adobe Communications Team, «Scrum Product Owner,» 18. Mars 2022. [Internett]. Tilgjengelig: https://business.adobe.com/blog/basics/product-owner. [Funnet 08. Februar 2023].

[13] K. Schwaber og J. Sutherland, «The 2020 Scrum Guide,» November 2020. [Internett]. Tilgjengelig: https://scrumguides.org/scrum-guide.html#the-sprint. [Funnet 08. Februar 2023].

[14] Compete 366, «What is Microsoft Teams and who should be using it?,» [Internett]. Tilgjengelig: https://www.compete366.com/blog-posts/microsoft-teams-what-is-it-and-should-we-be-using-it/. [Funnet 18. Januar 2023].

[15] Microsoft, «Microsoft Planner,» [Internett]. Tilgjengelig: https://www.microsoft.com/en-us/microsoft-365/business/task-management-software. [Funnet 09. Mai 2023].

[16] M. Rehkopf, «What is a kanban board?,» [Internett]. Tilgjengelig: https://www.atlassian.com/agile/kanban/boards. [Funnet 28. Januar 2023].

[17] GitHub, «Let's build from here,» GitHub, [Internett]. Tilgjengelig: https://github.com/about. [Funnet 03. Mai 2023].

[18] GitHub, «About Projects,» [Internett]. Tilgjengelig: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects. [Funnet 04. Mai 2023].

[19] S. Klevengen, D. Eidsheim og T. Günther, «NATSAdminportal,» Egde, 2023. [Internett]. Tilgjengelig: https://github.com/EgdeConsulting/NATSAdminportal. [Funnet 04. Mai 2023].

[20] D. Huether, «Definition of Done,» [Internett]. Tilgjengelig: https://www.leadingagile.com/2017/02/definition-of-done/. [Funnet 28. Januar 2023].

[21] S. Madan, «DONE Understanding Of The Definition Of "Done",» 16. Desember 2019. [Internett]. Tilgjengelig: https://www.scrum.org/resources/blog/done-understanding-definition-done. [Funnet 28. Januar 2023].

[22] A. Rolstadås, «Prosjektplanlegging,» Store Norske Leksikon, 17. April 2020. [Internett]. Tilgjengelig: https://snl.no/prosjektplanlegging. [Funnet 18. Mars 2023].

[23] Cloud Native Computing Foundation, «CNCF Charter,» 28. Juli 2021. [Internett]. Tilgjengelig: https://github.com/cncf/foundation/blob/main/charter.md. [Funnet 02. Februar 2023].

[24] Cloud Native Computing Foundation, «NATS,» [Internett]. Tilgjengelig: https://www.cncf.io/projects/nats/. [Funnet 09. Mai 2023].

[25] GeeksForGeeks, «What is Message Oriented Middleware (MOM)?,» 16. Desember 2021. [Internett]. Tilgjengelig: https://www.geeksforgeeks.org/what-is-message-oriented-middleware-mom/. [Funnet 18. Januar 2023].

[26] Coforge, «Message Oriented Middleware,» 17. Juni 2015. [Internett]. Tilgjengelig: https://www.coforge.com/blog/message-oriented-middleware. [Funnet 20. Februar 2023].

[27] RabbitMQ, «RabbitMQ is the most widely deployed open source message broker.,» [Internett]. Tilgjengelig: https://www.rabbitmq.com/. [Funnet 03. Mai 2023].

[28] Kafka, «Apache Kafka,» [Internett]. Tilgjengelig: https://kafka.apache.org/. [Funnet 03. Mai 2023].

[29] NATS, «Request-reply,» [Internett]. Tilgjengelig: https://docs.nats.io/nats-concepts/core-nats/reqreply. [Funnet 26. April 2023].

[30] NATS, «Official NATS Documentation,» NATS, [Internett]. Tilgjengelig: https://docs.nats.io/. [Funnet 10. Januar 2023].

[31] NATS, «Streams,» [Internett]. Tilgjengelig: https://docs.nats.io/nats-concepts/jetstream/streams. [Funnet 10. Februar 2023].

[32] NATS, «Consumers,» [Internett]. Tilgjengelig: https://docs.nats.io/nats-concepts/jetstream/consumers. [Funnet 10. Februar 2023].

[33] NATS, «JetStream,» [Internett]. Tilgjengelig: https://docs.nats.io/nats-concepts/jetstream. [Funnet 10. Februar 2023].

[34] Microsoft, «What is Azure,» [Internett]. Tilgjengelig: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/. [Funnet 09. Mai 2023].

[35] Microsoft, «App Service overview,» 03. Mars 2023. [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/azure/app-service/overview. [Funnet 03. April 2023].

[36] Red Hat, «What is a REST API?,» 08. Mai 2020. [Internett]. Tilgjengelig: https://www.redhat.com/en/topics/api/what-is-a-rest-api. [Funnet 15. Mars 2023].

[37] M. H. Gregersen, M. Ødegaard og T. Skagen, «Systematiske litteratursøk,» Universitetsbiblioteket i Oslo, Oslo, 2016.

[38] J. Shropshire, J. P. Landry og S. S. Presley, «Towards a consensus definition of full-stack development,» Association for Information Systems, 2018.

[39] N. Mupparaju, «Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware,» University of North Florida, Jacksonville, 2013.

[40] S. Raje, «Performance Comparison of Message Queue Methods,» University of Nevada, Las Vegas, Las Vegas, 2019.

[41] E. Nilsson og P. Victor, «Performance evaluation of message-oriented middleware,» Skolan för kemi, bioteknologi och hälsa, Stockholm, 2020.

[42] P. D og P. J. Bhat, «Modern Messaging Queues - RabbitMQ, NATS and NATS Streaming,» Blue Eyes Intelligence Engineering & Sciences Publication (BEIESP), Bhopal, 2020.

[43] G. Krüger og C. Lane, «How to write a Software Requirements Specification,» 17. Januar 2023. [Internett]. Tilgjengelig: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document. [Funnet 22. Februar 2023].

[44] G. Krüger og C. Lane, «How to Write a Software Requirements Specification (SRS Document),» 17. Januar 2023. [Internett]. Tilgjengelig: https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document. [Funnet 22. Februar 2023].

[45] E. Romani, «What Is a Software Requirement Specification?,» 28. Desember 2022. [Internett]. Tilgjengelig: https://builtin.com/software-engineering-perspectives/software-requirement-specification. [Funnet 22. Februar 2023].

[46] J. Dyson, «Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements,» 07. Januar 2019. [Internett]. Tilgjengelig: https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson/. [Funnet 22. Februar 2023].

[47] Visual Paradigm, «What is Use Case Diagram?,» [Internett]. Tilgjengelig: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/. [Funnet 30. Januar 2023].

[48] Lucidchart, «UML Use Case Diagram Tutorial,» [Internett]. Tilgjengelig: https://www.lucidchart.com/pages/uml-use-case-diagram. [Funnet 30. Januar 2023].

[49] S. Halwai, «SDLC Design Phase – Everything You Need to Know,» 20. Februar 2021. [Internett]. Tilgjengelig: https://www.openxcell.com/blog/design-phase-in-sdlc/. [Funnet 16. Mars 2023].

[50] F. Khalid, «How to Conduct the Design Phase in Software Development?,» [Internett]. Tilgjengelig: https://www.devteam.space/blog/how-to-conduct-the-design-phase-in-software-development/. [Funnet 16. Mars 2023].

[51] Miro, «What are context diagrams?,» [Internett]. Tilgjengelig: https://miro.com/blog/context-diagram/. [Funnet 22. Februar 2023].

[52] Lucidchart, «UML Class Diagram Tutorial,» [Internett]. Tilgjengelig: https://www.lucidchart.com/pages/uml-class-diagram. [Funnet 16. Mars 2023].

[53] NATS, «NATS - .NET C# Client,» [Internett]. Tilgjengelig: https://github.com/nats-io/nats.net. [Funnet 24. Januar 2023].

[54] T. H. Nätt og E. Rossen, «Brukergrensesnitt,» Store Norske Leksikon, 15. Februar 2023. [Internett]. Tilgjengelig: https://snl.no/brukergrensesnitt. [Funnet 25. Mars 2023].

[55] T. Lee, «NATS-WebUI,» 25. Februar 2020. [Internett]. Tilgjengelig: https://github.com/suisrc/NATS-WebUI. [Funnet 10. Februar 2023].

[56] L. Dehon, «Manage & view data inside your Apache Kafka ® cluster,» [Internett]. Tilgjengelig: https://akhq.io/. [Funnet 15. Januar 2023].

[57] J. Hannah, «What Exactly Is Wireframing? A Comprehensive Guide,» 27. Desember 2022. [Internett]. Tilgjengelig: https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/#what-is-a-wireframe. [Funnet 23. Mars 2023].

[58] D. Benyon, Designing user experience: a guide to HCI, UX and interaction design (3. utg.), Harlow: Pearson Education Limited, 2014.

[59] I. M. Lid, «Universell Utforming,» Store Norske Leksikon, 27. Desember 2021. [Internett]. Tilgjengelig: https://snl.no/universell_utforming. [Funnet 23. Mars 2023].

[60] S. L. Henry, «WCAG 2 Overview,» 21. Mars 2023. [Internett]. Tilgjengelig: https://www.w3.org/WAI/standards-guidelines/wcag/#intro. [Funnet 23. Mars 2023].

[61] WebAIM, «WAVE Browser Extensions,» [Internett]. Tilgjengelig: https://wave.webaim.org/extension/. [Funnet 24. Mars 2023].

[62] Amazon Web Services, «What Is SDLC (Software Development Lifecycle)?,» [Internett]. Tilgjengelig: https://aws.amazon.com/what-is/sdlc/. [Funnet 01. April 2023].

[63] MongoDB, «What is a Tech Stack and How Do They Work?,» [Internett]. Tilgjengelig: https://www.mongodb.com/basics/technology-stack. [Funnet 01. April 2023].

[64] smartdraw, «Selecting the Right Software Development Tools for Your Developers,» [Internett]. Tilgjengelig: https://www.smartdraw.com/technology/right-software-development-tools-for-developers.htm. [Funnet 04. Mai 2023].

[65] Atlassian, «What is Git?,» [Internett]. Tilgjengelig: https://www.atlassian.com/git/tutorials/what-is-git. [Funnet 15. Januar 2023].

[66] K. Zettler, «Trunk-based development,» [Internett]. Tilgjengelig: https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development. [Funnet 17. Januar 2023].

[67] P. Hammant, «Trunk-Based Development For Smaller Teams,» [Internett]. Tilgjengelig: https://trunkbaseddevelopment.com/#trunk-based-development-for-smaller-teams. [Funnet 17. Januar 2023].

[68] M. Heller, «What is Visual Studio Code? Microsofts extensible code editor,» 08. Juli 2022. [Internett]. Tilgjengelig: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html. [Funnet 19. Januar 2023].

[69] JetBrains, «Rider; Fast & powerful cross-platform .NET IDE,» [Internett]. Tilgjengelig: https://www.jetbrains.com/rider/. [Funnet 24. Februar 2023].

[70] Docker, «Develop faster. Run anywhere.,» [Internett]. Tilgjengelig: https://www.docker.com/. [Funnet 10. Mai 2023].

[71] GitHub, «About workflows,» [Internett]. Tilgjengelig: https://docs.github.com/en/actions/using-workflows/about-workflows. [Funnet 01. April 2023].

[72] Red Hat, «What is YAML?,» 03. Mars 2023. [Internett]. Tilgjengelig: https://www.redhat.com/en/topics/automation/what-is-yaml. [Funnet 01. April 2023].

[73] GitHub, «Working with the Container registry,» [Internett]. Tilgjengelig: https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry. [Funnet 01. April 2023].

[74] Amazon Web Services, «What Is Containerization?,» [Internett]. Tilgjengelig: https://aws.amazon.com/what-is/containerization/. [Funnet 02. April 2023].

[75] P. Chandrayan, «Docker Image vs Container: The Key Differences,» 03. Februar 2023. [Internett]. Tilgjengelig: https://www.knowledgehut.com/blog/devops/docker-vs-container. [Funnet 02. April 2023].

[76] Docker, «Developing with Docker,» [Internett]. Tilgjengelig: https://www.docker.com/why-docker/. [Funnet 02. April 2023].

[77] W. Chai, K. Brush og S. J. Bigelow, «What is PaaS? Platform as a service definition and guide,» Februar 2022. [Internett]. Tilgjengelig: https://www.techtarget.com/searchcloudcomputing/definition/Platform-as-a-Service-PaaS. [Funnet 30. April 2023].

[78] Typicode, «json-server README,» November 2013. [Internett]. Tilgjengelig: https://github.com/typicode/json-server/blob/master/README.md. [Funnet 22. Mars 2023].

[79] NATS, «Installing, running and deploying a NATS server,» NATS, [Internett]. Tilgjengelig: https://docs.nats.io/running-a-nats-service/introduction. [Funnet 03. Mai 2023].

[80] M. Boisbert, S. J. Bigelow og W. Chai, «Infrastructure as a Service (IaaS),» November 2022. [Internett]. Tilgjengelig: https://www.techtarget.com/searchcloudcomputing/definition/Infrastructure-as-a-Service-IaaS. [Funnet 30. April 2023].

[81] Microsoft, «Azure IaaS (infrastructure as a service),» [Internett]. Tilgjengelig: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure/azure-iaas/. [Funnet 30. April 2023].

[82] Kubernetes, «Kubernetes,» [Internett]. Tilgjengelig: https://kubernetes.io/. [Funnet 22. April 2023].

[83] Cloud Native Computing Foundation, «Nats-Server,» [Internett]. Tilgjengelig: https://github.com/nats-io/nats-server. [Funnet 22. April 2023].

[84] React, «React,» Meta, [Internett]. Tilgjengelig: https://reactjs.org/. [Funnet 10. Februar 2023].

[85] D. Colak, «What are the Advantages of TypeScript over JavaScript?,» 05. Februar 2023. [Internett]. Tilgjengelig: https://www.altogic.com/blog/what-are-the-advantages-of-typescript-over-javascript. [Funnet 13. April 2023].

[86] E. You, «Why Vite,» Vite, [Internett]. Tilgjengelig: https://vitejs.dev/guide/why.html. [Funnet 10. Februar 2023].

[87] S. Adebayo, «Getting Started,» Chakra, [Internett]. Tilgjengelig: https://chakra-ui.com/getting-started. [Funnet 11. Februar 2023].

[88] GeeksForGeeks, «Difference between Hot Reloading and Live Reloading in React Native,» 03. Mars 2022. [Internett]. Tilgjengelig: https://www.geeksforgeeks.org/difference-between-hot-reloading-and-live-reloading-in-react-native/. [Funnet 10. Mai 2023].

[89] R. J. Foss-Pedersen, «Universell Utforming,» Universitet i Oslo, 21. Mars 2017. [Internett]. Tilgjengelig: https://www.usit.uio.no/om/organisasjon/ffu/ux/blogg/2017/universell-utforming.html. [Funnet 07. Mars 2023].

[90] S. Klevengen, D. Eidsheim og T. Günther, «NATSAdminportal Frontend,» Egde, 2023. [Internett]. Tilgjengelig: https://github.com/EgdeConsulting/NATSAdminportal/tree/main/src/vite-client. [Funnet 13. April 2023].

[91] Microsoft, «What's new in .NET 7,» 09. Mars 2023. [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-7. [Funnet 15. Mars 2023].

[92] Microsoft, «ASP.NET MVC Overview,» Microsof, 10. April 2020. [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview. [Funnet 03. Mai 2023].

[93] C. Husk, «Announcing built-in container support for the .NET SDK,» 25. August 2022. [Internett]. Tilgjengelig: https://devblogs.microsoft.com/dotnet/announcing-builtin-container-support-for-the-dotnet-sdk/. [Funnet 15. Mars 2023].

[94] Microsoft, «Microsoft.Extensions.Configuration Namespace,» [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration?view=dotnet-plat-ext-7.0. [Funnet 27. April 2023].

[95] Microsoft, «UserSecretsConfigurationExtensions.AddUserSecrets Method,» [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/dotnet/api/Microsoft.Extensions.Configuration.UserSecretsConfigurationExtensions.AddUserSecrets?view=dotnet-plat-ext-7.0&viewFallbackFrom=net-7.0. [Funnet 06. Februar 2023].

[96] R. Anderson og K. Larkin, «Safe storage of app secrets in development in ASP.NET Core,» 04. November 2023. [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets. [Funnet 25. April 2023].

[97] Microsoft, «An introduction to NuGet,» [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/nuget/what-is-nuget. [Funnet 25. April 2023].

[98] Microsoft, «NuGet Package Manager,» [Internett]. Tilgjengelig: https://marketplace.visualstudio.com/items?itemName=NuGetTeam.NuGetPackageManager. [Funnet 25. April 2023].

[99] S. Klevengen, D. Eidsheim og T. Günther, «NATSAdminportal Backend,» Egde, 2023. [Internett]. Tilgjengelig: https://github.com/EgdeConsulting/NATSAdminportal/tree/main/src/vite-api. [Funnet 18. April 2023].

[100] S. Pittet, «The different types of software testing,» [Internett]. Tilgjengelig: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing. [Funnet 01. April 2023].

[101] IBM, «What is software testing?,» [Internett]. Tilgjengelig: https://www.ibm.com/topics/software-testing. [Funnet 01. April 2023].

[102] A. S. Gillis, «user acceptance testing (UAT),» Mars 2022. [Internett]. Tilgjengelig: https://www.techtarget.com/searchsoftwarequality/definition/user-acceptance-testing-UAT. [Funnet 05. April 2023].

[103] T. Hamilton, «What is User Acceptance Testing (UAT)?,» 11. Mars 2023. [Internett]. Tilgjengelig: https://www.guru99.com/user-acceptance-testing.html. [Funnet 05. April 2023].

[104] xUnit.net, «About xUnit.net,» [Internett]. Tilgjengelig: https://xunit.net/#documentation. [Funnet 06. April 2023].

[105] Microsoft, «Assert Class,» [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert?view=visualstudiosdk-2022. [Funnet 06. April 2023].

[106] S. Klevengen, D. Eidsheim og T. Günther, «NATSAdminportal Testing,» Egde, 2023. [Internett]. Tilgjengelig: https://github.com/EgdeConsulting/NATSAdminportal/tree/main/src/vite-api.Tests. [Funnet 18. April 2023].

[107] Nrwl, «What is a Monorepo?,» [Internett]. Tilgjengelig: https://monorepo.tools/#what-is-a-monorepo. [Funnet 20. April 2023].

[108] V. Prazarkevicius, «Git repos for Frontend and Backend parts: one or several?,» 05. Januar 2020. [Internett]. Tilgjengelig: https://vytotas.medium.com/git-repos-for-frontend-and-backend-parts-one-or-several-c0aa3eb304. [Funnet 18. April 2023].

[109] B. S. Lassen, A. M. Lund og K. M. Tande, «Opphavsrett,» i Store norske leksikon, 11. Februar 2022. [Internett]. Tilgjengelig: https://snl.no/opphavsrett. [Funnet 15. Januar 2023].

[110] T. Gramstad, «Åpen kildekode,» i Store norske leksikon, 13. April 2021. [Internett]. Tilgjengelig: https://snl.no/%C3%A5pen_kildekode. [Funnet 16. Januar 2023].

[111] Synopsys, «Top open source licenses and legal risk for developers,» 13. Juli 2022. [Internett]. Tilgjengelig: https://www.synopsys.com/blogs/software-security/top-open-source-licenses/. [Funnet 18. Januar 2023].

[112] Open Source Initiative, «The MIT License,» [Internett]. Tilgjengelig: https://opensource.org/license/mit/. [Funnet 18. Januar 2023].

[113] S. Klevengen, D. Eidsheim og T. Günther, «NATSAdminportal Lisens,» Egde, 2023. [Internett]. Tilgjengelig: https://github.com/EgdeConsulting/NATSAdminportal/blob/main/LICENSE. [Funnet 18. April 2023].

[114] Clarius Consulting, Manas & InSTEDD, «moq,» [Internett]. Tilgjengelig: https://github.com/moq/moq4. [Funnet 02. Mai 2023].

[115] Microsoft, «sealed (C# Reference),» 15. September 2021. [Internett]. Tilgjengelig: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/sealed. [Funnet 02. Mai 2023].

[116] T. Hamilton, «Integration Testing: What is, Types with Example,» 04. Mars 2023. [Internett]. Tilgjengelig: https://www.guru99.com/integration-testing.html. [Funnet 26. April 2023].

[117] RedHat, «What is CI/CD?,» 11. Mai 2022. [Internett]. Tilgjengelig: https://www.redhat.com/en/topics/devops/what-is-ci-cd. [Funnet 27. April 2023].

[118] I. Montiel, «Low Coupling, High Cohesion,» 17. September 2018. [Internett]. Tilgjengelig: https://medium.com/clarityhub/low-coupling-high-cohesion-3610e35ac4a6. [Funnet 27. April 2023].

[119] NATS, «Subject-Based messaging,» [Internett]. Tilgjengelig: https://docs.nats.io/nats-concepts/subjects. [Funnet 10. Februar 2023].

[120] S. L. Henry, «WCAG 2 Overview,» 25. Januar 2023. [Internett]. Tilgjengelig: https://www.w3.org/WAI/standards-guidelines/wcag/#intro. [Funnet 22. Februar 2023].

[121] Egde, «Definition of Done SMIA,» Upublisert, 2022.

Vedlegg

Vedlegg A – Bachelor Prosjektbeskrivelse

TS3000 Bacheloroppgave

Studieretning: Elektroingeniør – Informatikk og automatisering

Prosjektgruppe: IA6-3-23

Tittel: Utvikling av en administrasjonsportal for køsystemer basert på NATS

Veileder: Martin Nenseth

Samarbeidspartner: Egde

Prosjektets bakgrunn:

Et av Egdes Smia teams (Invictus) jobber med vedlikehold, drift og videreutvikling av en helseplattform og tilhørende applikasjoner. Disse kommuniserer med hverandre vha. meldingsutveksling. Begrepet «melding» omfatter i denne konteksten data i form av f.eks. JSON-objekter som inneholder applikasjonsinformasjon, helsejournaler, brukerdata og mer. Selve utvekslingen av meldinger organiseres og styres vha. et køsystem som baserer seg på NATS teknologien.

Utfordringen som Invictus per dags dato opplever, er at det ikke finnes noen administrasjonsverktøy for NATS-køer. Dette betyr at Invictus ikke kan overvåke meldinger som kommer inn, går ut eller ligger i en bestem kø på en enkel måte. Dessuten har de ingen måte for å se hvilke NATS-køer som er operative eller inaktive på en NATS-server. Resultatet av dette er tungvint vedlikehold og feilsøking av problemer relatert til meldinger på NATS-køer. I tillegg er videreutvikling av den eksisterende helseplattform negativt påvirket av manglende administrasjonsverktøy.

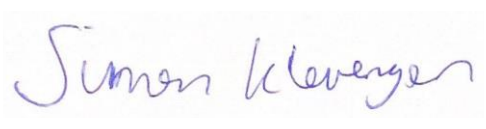
Målbeskrivelse for prosjektet:

Det overordnede formålet med prosjektet er å utvikle en løsning som skal være en fungerende første versjon av en administrasjonsportal for NATS-køer. Dette innebærer at løsningen skal tillate grunnleggende administrasjon og vedlikehold av en eller flere NATS-køer med tilhørende meldinger. Alt dette sørger for at prosjektet skal inkludere følgende:

- Undersøkelse og analyse av eksisterende løsninger og relevant faglitteratur som kan hjelpe med å forbedre eller gi inspirasjon til utviklingsprosessen.
- Opprettelse og konfigurasjon av et utviklings og produksjonsmiljø som baserer seg på .NET 7, React, TypeScript og Vite. I tillegg skal også settes opp en egen NATS-server som det skal jobbes mot. Utviklingsmiljøet skal tilgjengelig gjøres vha. skytjenester (f.eks. Azure Cloud).
- Utforming (design) og implementasjon av en administrasjonsportal som gir mulighet for å se meldinger som ligge i en bestemt NATS-kø, samtidig som at meldinger skal kunne flyttes blant ulike køer eller slettes fra en kø. På toppen av dette skal portalen også kunne vise alle NATS-køer som kjører på en NATS-server.
- Dokumentering av løsningen med hensyn på ting som installasjon og virkemåte av implementasjonen av administrasjonsportalen.
- Opprettelse av rutiner for automatisert testing.
- Kartlegging og redegjøring av hvordan videreutvikling av administrasjonsportalen kan foregå. Dette innebærer en diskusjon som tar opp fremtidig funksjonalitet som kan/burde implementeres i løsningen.

For å kunne utvikle denne løsningen og sikre at den fungerer optimalt skal prosjektet struktureres etter agil utviklingsmetodikken. Dette innebærer at gjeldende fem faser: planlegging, design, koding, testing og tilbakemelding skal gjennomgås fortløpende i små sykluser (sprinter) på 3 uker. Denne organisering skal tillatte for tett samarbeid mellom prosjektgruppen og prosjekteieren (Invictus) slik at prosjektfremdriften med tilhørende arbeidsoppgaver og milepærer kan justeres fortløpende.

Signatur

Veileder (dato og signatur):	16.01.2023	
Studenter (dato og signatur):	18.01.2023	
	18.01.2023	
	18.01.2023	

Vedlegg B – Prosjektgruppen

Dette bachelorprosjektet ble gjennomført av tre studenter i sitt sjette semester ved USN campus Porsgrunn.

Gruppeleder, Tester & Frontend-utvikler

Simen Klevengen

Elektroingeniør, Informatikk og automatisering

QA & Fullstack-utvikler

Daniel Eidsheim

Elektroingeniør, Informatikk og automatisering

Systemarkitekt, Tester & Backend-utvikler

Tobias Günther

Elektroingeniør, Informatikk og automatisering

Vedlegg C – Medforfattererklæring

Medforfattererklæring - bacheloroppgave

Dette skjemaet skal fylles ut og signeres av alle studentene i prosjektgruppen. Ferdig utfylt og signert skjema skal ligge som et vedlegg i rapporten.

Tittel på oppgaven	Utvikling av en administrasjonsportal for køsystemer basert på NATS
Veileder fra USN	Leila Ben Saad

Beskriv hva hver student i prosjektgruppen har bidratt med i bacheloroppgaven.

Daniel Eidsheim har bidratt med følgende ting i prosjektet. Merk at oversikten også inkluderer felles oppgaver hvor hele prosjektgruppen har vært involvert like mye:

Oppgavetittel	Fullført av	Prioritert	Fullført	Samling ↓
~~Sette opp repository~~	TG Tobias Günther		13.1.	Uke 2 (09.01 - 15.01)
~~Definere Prosjekt "DoD"~~	TG Tobias Günther		14.1.	Uke 2 (09.01 - 15.01)
~~Definere Prosjektbeskrivelse~~	DE Daniel Eidsheim		18.1.	Uke 2 (09.01 - 15.01)
~~Utforme Kravspesifikasjon (v1.0)~~	S simenklevengen		25.1.	Uke 2 (09.01 - 15.01)
~~Konfigurer Kode-formatterer i VS Code~~	DE Daniel Eidsheim		18.1.	Uke 3 (16.01 - 22.01)
~~Finne relevante og vitenskapelige artikler~~	DE Daniel Eidsheim		18.1.	Uke 3 (16.01 - 22.01)
~~Gå gjennom feedback fra kravspesifikasjon~~	TG Tobias Günther		20.1.	Uke 3 (16.01 - 22.01)
~~Opprette NATS-Server~~	TG Tobias Günther		22.1.	Uke 3 (16.01 - 22.01)
~~Gjør at NATS-server på VM automatisk restarter hver natt~~	DE Daniel Eidsheim		27.1.	Uke 4 (23.01 - 29.01)
~~Opprette veilederavtale~~	DE Daniel Eidsheim	!	25.1.	Uke 4 (23.01 - 29.01)
~~Opprette og signere gruppeavtale~~	S simenklevengen	!	25.1.	Uke 4 (23.01 - 29.01)
~~Implementere JSON-server~~	DE Daniel Eidsheim		26.1.	Uke 4 (23.01 - 29.01)
~~Oppdatere kravspesifikasjon (v1.1)~~	TG Tobias Günther		25.1.	Uke 4 (23.01 - 29.01)
~~Skrive om CNFC~~	DE Daniel Eidsheim		2.2.	Uke 5 (30.01 - 05.02)
~~Legge til subject hierarki på NATS-server~~	DE Daniel Eidsheim		8.2.	Uke 5 (30.01 - 05.02)
~~Undersøke jetstreams dotnet/nats-client~~	DE Daniel Eidsheim		2.2.	Uke 5 (30.01 - 05.02)
~~Sprint 1 Review~~	DE Daniel Eidsheim		1.2.	Uke 5 (30.01 - 05.02)
~~Legge til "New Message Modal"~~	DE Daniel Eidsheim		15.2.	Uke 7 (13.02 - 19.02)
~~Opprette subject-hierarki oversikt~~	TG Tobias Günther		17.2.	Uke 7 (13.02 - 19.02)
~~Skrive om litteratursøkene~~	DE Daniel Eidsheim		15.2.	Uke 7 (13.02 - 19.02)
~~Validering av meldinginputs~~	DE Daniel Eidsheim		26.4.	Uke 8 (20.02 - 26.02)
~~Legge til konfirmasjon for publisering av melding~~	DE Daniel Eidsheim		26.4.	Uke 8 (20.02 - 26.02)
~~Skille ut subject dropdown til egen komponent~~	DE Daniel Eidsheim		26.4.	Uke 8 (20.02 - 26.02)
~~Fikse import struktur~~	DE Daniel Eidsheim		26.4.	Uke 8 (20.02 - 26.02)
~~Utarbeid Streams-Page~~	S simenklevengen		26.4.	Uke 8 (20.02 - 26.02)
~~Sprint 2 Review~~	S simenklevengen		26.4.	Uke 8 (20.02 - 26.02)

Vedlegg C – Medforfattererklæring

~~Fikse usersecrets lokalt på pc~~		simenklevengen		26.4.	Uke 9 (27.02 - 05.03)
~~Gjennomføre Veilederemøte NR1~~		Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Installere C# XML dokumentasjons utvidelse~~		Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Gjennomgå .NET kode med Egde~~		Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Opprette møte med Daniel Vassdal~~		simenklevengen		26.4.	Uke 9 (27.02 - 05.03)
~~Installere WAVE-utvidelsen i nettleser~~		Tobias Günther		26.4.	Uke 9 (27.02 - 05.03)
~~Forbedre publiseringsmodal~~		Daniel Eidsheim		15.3.	Uke 10 (06.03 - 12.03)
~~Sette opp ny VM i Azure~~		Daniel Eidsheim		8.3.	Uke 10 (06.03 - 12.03)
~~Skrive om Rider~~		Daniel Eidsheim		22.3.	Uke 11 (13.03 - 19.03)
~~Skrive om NATS og JetStreams~~		Daniel Eidsheim		22.3.	Uke 11 (13.03 - 19.03)
~~Legge til "tømming" av MsgContext etter sletting av melding~~		Daniel Eidsheim		16.3.	Uke 11 (13.03 - 19.03)
~~Lage komponent for meldings kopiering~~		Daniel Eidsheim		15.3.	Uke 11 (13.03 - 19.03)
~~Legge til meldingskopiering~~		Daniel Eidsheim		15.3.	Uke 11 (13.03 - 19.03)
~~Bytte ut datatypen "any"~~		Daniel Eidsheim		29.3.	Uke 12 (20.03 - 26.03)
~~Legge til dokumentasjon i SubjectManager.cs~~		Daniel Eidsheim		29.3.	Uke 12 (20.03 - 26.03)
~~Legge til dokumentasjon i StreamManager.cs~~		Daniel Eidsheim		29.3.	Uke 12 (20.03 - 26.03)
~~Sprint Review 3~~		simenklevengen		29.3.	Uke 12 (20.03 - 26.03)
~~Rework stream1 and stream2~~		Daniel Eidsheim		31.3.	Uke 13 (27.03 - 02.04)
~~Legge til "Quick start" cards på hjemmesiden~~		Daniel Eidsheim		30.3.	Uke 13 (27.03 - 02.04)
~~Skrive om utviklings- og produksjonsmiljø~~		Daniel Eidsheim		12.4.	Uke 14 (03.04 - 09.04)
~~Fikse kontrast problemer i light-mode~~		Daniel Eidsheim		12.4.	Uke 14 (03.04 - 09.04)
~~Utvide logikk på API-endepunkter~~		Daniel Eidsheim		12.4.	Uke 14 (03.04 - 09.04)
~~Fikse feil i logging av meldingskopiering~~		Daniel Eidsheim		19.4.	Uke 15 (10.04 - 16.04)
~~Skrive enhetstester for SubscriberManager.cs~~		Daniel Eidsheim		19.4.	Uke 15 (10.04 - 16.04)
~~Utvide README.md til å inkludere testing~~		Daniel Eidsheim		19.4.	Uke 15 (10.04 - 16.04)
~~Frontend Workshop~~		simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Sprint Review 4~~		simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Sprint Review 4~~		simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Fix any types~~		Daniel Eidsheim		20.4.	Uke 16 (17.04 - 23.04)
~~Skrive Diskusjonskapitlet~~		Daniel Eidsheim		26.4.	Uke 16 (17.04 - 23.04)
~~Lag Bachelorpresentasjon (v1.0)~~		simenklevengen	!	26.4.	Uke 16 (17.04 - 23.04)
~~Justere Bachelorpresentasjonen basert på tilbakemeldinger~~		simenklevengen		3.5.	Uke 17 (24.04 - 30.04)
~~Ta bilde for gruppeoversikten~~		Daniel Eidsheim		3.5.	Uke 17 (24.04 - 30.04)
~~Korrekturlese Bacheloren~~		Daniel Eidsheim		28.4.	Uke 17 (24.04 - 30.04)
~~Gjennomføre Veiledermøte 2~~		Tobias Günther		3.5.	Uke 17 (24.04 - 30.04)
~~Signere Bacheloren~~		simenklevengen		5.5.	Uke 18 (01.05 - 07.05)
~~Justere Bacheloren basert på tilbakemeldinger~~		Daniel Eidsheim		3.5.	Uke 18 (01.05 - 07.05)
~~Gjennomgå tilbakemeldinger for Bacheloren~~		simenklevengen		3.5.	Uke 18 (01.05 - 07.05)
~~Sprint Review 5~~		Daniel Eidsheim		5.5.	Uke 18 (01.05 - 07.05)
~~Levere førsteutkast av Bacheloren til Leila for tilbakemelding~~		Tobias Günther		3.5.	Uke 18 (01.05 - 07.05)

Vedlegg C – Medforfattererklæring

Tobias Günther har bidratt med følgende ting i prosjektet. Merk at oversikten også inkluderer felles oppgaver hvor hele prosjektgruppen har vært involvert like mye:

Oppgavetittel	Fullført av	Prioritert	Fullført	Samling ↓
~~Sette opp repository~~	Tobias Günther		13.1.	Uke 2 (09.01 - 15.01)
~~Definere Prosjekt "DoD"~~	Tobias Günther		14.1.	Uke 2 (09.01 - 15.01)
~~Definere Prosjektbeskrivelse~~	Daniel Eidsheim		18.1.	Uke 2 (09.01 - 15.01)
~~Endre backend fra .NET6 til .NET7~~	Tobias Günther		13.1.	Uke 2 (09.01 - 15.01)
~~Lage Use-case Diagrammer (v1.0)~~	simenklevengen		25.1.	Uke 2 (09.01 - 15.01)
~~Konfigurer Kode-formatterer i VS Code~~	Daniel Eidsheim		18.1.	Uke 3 (16.01 - 22.01)
~~Opprette et utviklings- og produksjonsmiljø i GitHub og Azure~~	Daniel Eidsheim		19.1.	Uke 3 (16.01 - 22.01)
~~Finne relevante og vitenskapelige artikler~~	Daniel Eidsheim		18.1.	Uke 3 (16.01 - 22.01)
~~Gå gjennom feedback fra kravspesifikasjon~~	Tobias Günther		20.1.	Uke 3 (16.01 - 22.01)
~~Oppdatere Use-case diagram (v1.1)~~	Tobias Günther		29.1.	Uke 4 (23.01 - 29.01)
~~Legge til støtte for secrets~~	Tobias Günther		27.1.	Uke 4 (23.01 - 29.01)
~~Opprette veilederavtale~~	Daniel Eidsheim	!	25.1.	Uke 4 (23.01 - 29.01)
~~Opprette og signere gruppeavtale~~	simenklevengen	!	25.1.	Uke 4 (23.01 - 29.01)
~~Oppdatere kravspesifikasjon (v1.1)~~	Tobias Günther		25.1.	Uke 4 (23.01 - 29.01)
~~Opprette kontekst diagram (v1.0)~~	Tobias Günther		25.1.	Uke 4 (23.01 - 29.01)
~~Liste: Funksjonelle og ikke-funksjonelle krav~~	Tobias Günther		23.1.	Uke 4 (23.01 - 29.01)
~~Utvide JSON-object for meldinger~~	Tobias Günther		4.2.	Uke 5 (30.01 - 05.02)
~~Opprette fleksibel meldingsoversikt~~	Tobias Günther		3.2.	Uke 5 (30.01 - 05.02)
~~Lage en MessageView komponent~~	Tobias Günther		1.2.	Uke 5 (30.01 - 05.02)
~~Skrive referat fra Sprint 1 Review~~	Tobias Günther		1.2.	Uke 5 (30.01 - 05.02)
~~Oppdatere Publisher.cs~~	Tobias Günther		1.2.	Uke 5 (30.01 - 05.02)
~~Lage topbar~~	Tobias Günther		1.2.	Uke 5 (30.01 - 05.02)
~~Sprint 1 Review~~	Daniel Eidsheim		1.2.	Uke 5 (30.01 - 05.02)
~~Konvertere subject-hierarki fra string array til JSON~~	Tobias Günther		12.2.	Uke 6 (06.02 - 12.02)
~~Separere komponentstyling ut til egne filer~~	Tobias Günther		12.2.	Uke 6 (06.02 - 12.02)
~~Skrive om prosjektorganisering~~	Tobias Günther		8.2.	Uke 6 (06.02 - 12.02)
~~Refactor StreamManager og SubjectManager~~	Tobias Günther		17.2.	Uke 7 (13.02 - 19.02)
~~Opprette subject-hierarki oversikt~~	Tobias Günther		17.2.	Uke 7 (13.02 - 19.02)
~~Begynne implementasjon av logging~~	Tobias Günther		17.2.	Uke 7 (13.02 - 19.02)
~~Opprette logging av hendelser~~	Tobias Günther		26.4.	Uke 8 (20.02 - 26.02)
~~Skrive om kravspec i bacheloren~~	Tobias Günther		26.4.	Uke 8 (20.02 - 26.02)
~~Redigere kravspec basert på tilbakemeldinger~~	Tobias Günther		26.4.	Uke 8 (20.02 - 26.02)
~~Sprint 2 Review~~	simenklevengen		26.4.	Uke 8 (20.02 - 26.02)
~~Gjennomføre Veilederemøte NR1~~	Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Gjennomgå .NET kode med Egde~~	Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Installere WAVE-utvidelsen i nettleser~~	Tobias Günther		26.4.	Uke 9 (27.02 - 05.03)
~~Forbedre detaljert meldingsvisning~~	Tobias Günther		15.3.	Uke 10 (06.03 - 12.03)
~~Flytte applikasjonen inn i en docker container~~	Tobias Günther		15.3.	Uke 10 (06.03 - 12.03)
~~Legge til en DeleteMessageModal~~	Tobias Günther		10.3.	Uke 10 (06.03 - 12.03)
~~Legge til en hjemmeside~~	Tobias Günther		18.3.	Uke 11 (13.03 - 19.03)
~~Skrive innledning for Design-kapitlet~~	simenklevengen		16.3.	Uke 11 (13.03 - 19.03)
~~Legg inn MsgCopy komponenten inn i MsgView~~	Tobias Günther		15.3.	Uke 11 (13.03 - 19.03)
~~Skrive om UU~~	Tobias Günther		24.3.	Uke 12 (20.03 - 26.03)
~~Skrive om navigasjonskart~~	Tobias Günther		23.3.	Uke 12 (20.03 - 26.03)
~~Skrive om wireframes~~	Tobias Günther		26.3.	Uke 12 (20.03 - 26.03)
~~Sprint Review 3~~	simenklevengen		29.3.	Uke 12 (20.03 - 26.03)
~~Sette opp enhetstesting for JetStreamSubscriber~~	Tobias Günther		6.4.	Uke 13 (27.03 - 02.04)

Vedlegg C – Medforfattererklæring

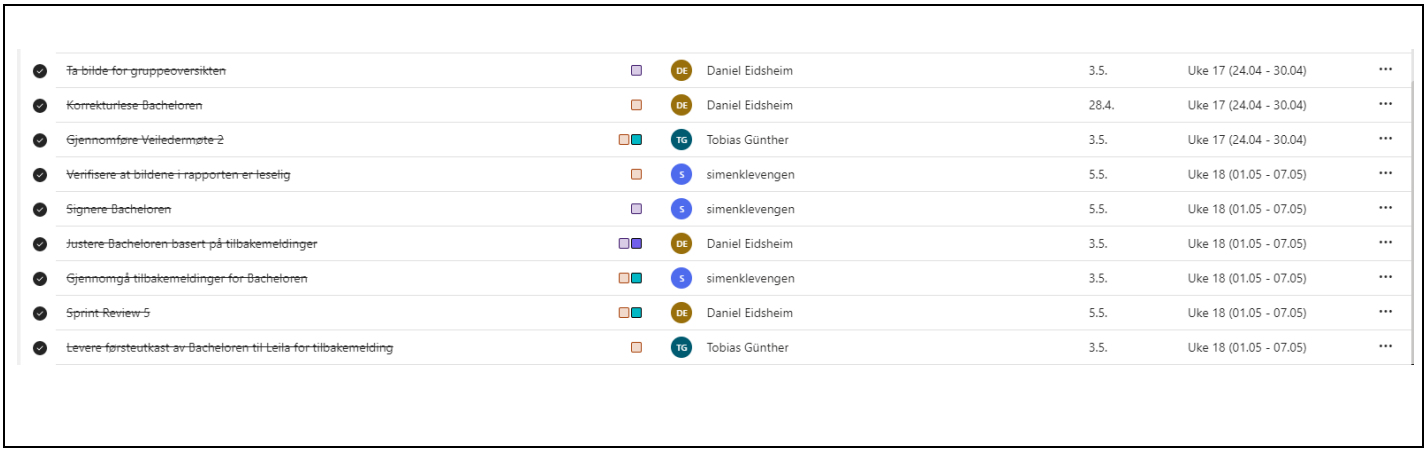
~~Forbedre detaljert meldingsvisning~~	TG	Tobias Günther		15.3.	Uke 10 (06.03 - 12.03)
~~Flytte applikasjonen inn i en docker container~~	TG	Tobias Günther		15.3.	Uke 10 (06.03 - 12.03)
~~Legge til en DeleteMessageModal~~	TG	Tobias Günther		10.3.	Uke 10 (06.03 - 12.03)
~~Legge til en hjemmeside~~	TG	Tobias Günther		18.3.	Uke 11 (13.03 - 19.03)
~~Skrive innledning for Design-kapitlet~~	S	simenklevengen		16.3.	Uke 11 (13.03 - 19.03)
~~Legg inn MsgCopy komponenten inn i MsgView~~	TG	Tobias Günther		15.3.	Uke 11 (13.03 - 19.03)
~~Skrive om UU~~	TG	Tobias Günther		24.3.	Uke 12 (20.03 - 26.03)
~~Skrive om navigasjonskart~~	TG	Tobias Günther		23.3.	Uke 12 (20.03 - 26.03)
~~Skrive om wireframes~~	TG	Tobias Günther		26.3.	Uke 12 (20.03 - 26.03)
~~Sprint Review 3~~	S	simenklevengen		29.3.	Uke 12 (20.03 - 26.03)
~~Sette opp enhetstesting for JetStreamSubscriber~~	TG	Tobias Günther		6.4.	Uke 13 (27.03 - 02.04)
~~Oppdatere README med mer informasjon~~	TG	Tobias Günther		30.3.	Uke 13 (27.03 - 02.04)
~~Legge til filter på Msg- og StreamTable~~	TG	Tobias Günther		26.3.	Uke 13 (27.03 - 02.04)
~~Automatisere publisering av docker container til Azure~~	TG	Tobias Günther		30.3.	Uke 13 (27.03 - 02.04)
~~Skrive om Utviklings- og produksjonsmiljø~~	TG	Tobias Günther		6.4.	Uke 14 (03.04 - 09.04)
~~Gjennomgå UI vha. Wave for å avdekke og fikse problemer ifm. universell utforming~~	TG	Tobias Günther		6.4.	Uke 14 (03.04 - 09.04)
~~Fikse bug ifm. paginatedTable~~	TG	Tobias Günther		13.4.	Uke 15 (10.04 - 16.04)
~~Skrive enhetstester for SubjectManager.cs~~	TG	Tobias Günther		18.4.	Uke 15 (10.04 - 16.04)
~~Frontend Workshop~~	S	simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Sprint Review 4~~	S	simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Opprette klassediagram vha. Visual Studio~~	TG	Tobias Günther		13.4.	Uke 15 (10.04 - 16.04)
~~Skrive om NATS-server~~	TG	Tobias Günther		24.4.	Uke 16 (17.04 - 23.04)
~~Legge til automatisert testing i GitHub~~	TG	Tobias Günther		22.4.	Uke 16 (17.04 - 23.04)
~~Lag Bachelorpresentasjon (v1.0)~~	S	simenklevengen	!	26.4.	Uke 16 (17.04 - 23.04)
~~Justere Bachelorpresentasjonen basert på tilbakemeldinger~~	S	simenklevengen		3.5.	Uke 17 (24.04 - 30.04)
~~Organisere Egde Roll-up for USN EXPO~~	TG	Tobias Günther		26.4.	Uke 17 (24.04 - 30.04)
~~Fjerne mellomtidlige consumers fra stream data~~	TG	Tobias Günther		28.4.	Uke 17 (24.04 - 30.04)
~~Ta bilde for gruppeoversikten~~	DE	Daniel Eidsheim		3.5.	Uke 17 (24.04 - 30.04)
~~Korrekturlese Bacheloren~~	DE	Daniel Eidsheim		28.4.	Uke 17 (24.04 - 30.04)
~~Gjennomføre Veiledermøte 2~~	TG	Tobias Günther		3.5.	Uke 17 (24.04 - 30.04)
~~Signere Bacheloren~~	S	simenklevengen		5.5.	Uke 18 (01.05 - 07.05)
~~Justere Bacheloren basert på tilbakemeldinger~~	DE	Daniel Eidsheim		3.5.	Uke 18 (01.05 - 07.05)
~~Gjennomgå tilbakemeldinger for Bacheloren~~	S	simenklevengen		3.5.	Uke 18 (01.05 - 07.05)
~~Sprint Review 5~~	DE	Daniel Eidsheim		5.5.	Uke 18 (01.05 - 07.05)
~~Fylle ut medforfattererklæringen~~	TG	Tobias Günther		5.5.	Uke 18 (01.05 - 07.05)
~~Levere førsteutkast av Bacheloren til Leila for tilbakemelding~~	TG	Tobias Günther		3.5.	Uke 18 (01.05 - 07.05)

Vedlegg C – Medforfattererklæring

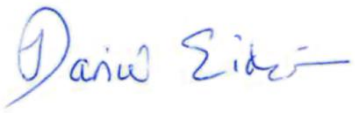
Simen Klevengen har bidratt med følgende ting i prosjektet. Merk at oversikten også inkluderer felles oppgaver hvor hele prosjektgruppen har vært involvert like mye:

Oppgavetittel	Fullført av	Prioritert	Fullført	Samling ↓
~~Sette opp repository~~	Tobias Günther		13.1.	Uke 2 (09.01 - 15.01)
~~Definere Prosjekt "DoD"~~	Tobias Günther		14.1.	Uke 2 (09.01 - 15.01)
~~Opprette Systemtegning (v1.0)~~	simenklevengen		25.1.	Uke 2 (09.01 - 15.01)
~~Definere Prosjektbeskrivelse~~	Daniel Eidsheim		18.1.	Uke 2 (09.01 - 15.01)
~~Konfigurer Kode-formatterer i VS Code~~	Daniel Eidsheim		18.1.	Uke 3 (16.01 - 22.01)
~~Finne relevant litteratur~~	simenklevengen		18.1.	Uke 3 (16.01 - 22.01)
~~Finne relevante og vitenskapelige artikler~~	Daniel Eidsheim		18.1.	Uke 3 (16.01 - 22.01)
~~Gå gjennom feedback fra kravspesifikasjon~~	Tobias Günther		20.1.	Uke 3 (16.01 - 22.01)
~~Opprette NATS-Server~~	Tobias Günther		22.1.	Uke 3 (16.01 - 22.01)
~~Opprette veilederavtale~~	Daniel Eidsheim	!	25.1.	Uke 4 (23.01 - 29.01)
~~Opprette og signere gruppeavtale~~	simenklevengen	!	25.1.	Uke 4 (23.01 - 29.01)
~~Oppdatere kravspesifikasjon (v1.1)~~	Tobias Günther		25.1.	Uke 4 (23.01 - 29.01)
~~Lage sidebar~~	simenklevengen		8.2.	Uke 5 (30.01 - 05.02)
~~Oppdatere Systemtegning (v1.1)~~	simenklevengen		1.2.	Uke 5 (30.01 - 05.02)
~~Sprint 1 Review~~	Daniel Eidsheim		1.2.	Uke 5 (30.01 - 05.02)
~~Lage react routing~~	simenklevengen		12.2.	Uke 6 (06.02 - 12.02)
~~Opprette møteinvitasjon for formell veiledermøte~~	simenklevengen		13.2.	Uke 7 (13.02 - 19.02)
~~Skrive om litteratursøkene~~	Daniel Eidsheim		15.2.	Uke 7 (13.02 - 19.02)
~~Sprint 2 Review~~	simenklevengen		26.4.	Uke 8 (20.02 - 26.02)
~~Fikse usersecrets lokalt på pc~~	simenklevengen		26.4.	Uke 9 (27.02 - 05.03)
~~Gjennomføre Veilederemøte NR1~~	Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Installere C# XML dokumentasjons utvidelse~~	Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Gjennomgå .NET kode med Egde~~	Daniel Eidsheim		26.4.	Uke 9 (27.02 - 05.03)
~~Opprette møte med Daniel Vassdal~~	simenklevengen		26.4.	Uke 9 (27.02 - 05.03)
~~Installere WAVE-utvidelsen i nettleser~~	Tobias Günther		26.4.	Uke 9 (27.02 - 05.03)
~~Oppdatere arkitekturtegning~~	simenklevengen		16.3.	Uke 10 (06.03 - 12.03)
~~Skrive dokumentasjon om JSON-server~~	simenklevengen		22.3.	Uke 11 (13.03 - 19.03)
~~Skrive innledning for Design-kapitlet~~	simenklevengen		16.3.	Uke 11 (13.03 - 19.03)
~~Utvide datasettet til JSON-server~~	simenklevengen		22.3.	Uke 11 (13.03 - 19.03)
~~Fikse på MessageView og StreamView komponentene~~	simenklevengen		29.3.	Uke 12 (20.03 - 26.03)
~~Legge til en LoadingIcon component~~	simenklevengen		24.3.	Uke 12 (20.03 - 26.03)
~~Sprint Review 3~~	simenklevengen		29.3.	Uke 12 (20.03 - 26.03)
~~Fikse kontrast problemer med knapper i tabeller~~	simenklevengen		12.4.	Uke 13 (27.03 - 02.04)
~~Sette opp enhetstesting for JetStreamSubscriber.cs~~	simenklevengen		12.4.	Uke 13 (27.03 - 02.04)
~~Oppdatere SubjectSidebar komponenten~~	simenklevengen		30.3.	Uke 13 (27.03 - 02.04)
~~Legge til Payload modal~~	Daniel Eidsheim		30.3.	Uke 13 (27.03 - 02.04)
~~Bytte ut alle Wikipedia kilder med SNL eller noe annet i Bacheloren~~	simenklevengen		30.3.	Uke 13 (27.03 - 02.04)
~~Sette opp enhetstesting for Publisher.cs~~	simenklevengen		12.4.	Uke 13 (27.03 - 02.04)
~~Skrive konklusjonen~~	simenklevengen		12.4.	Uke 14 (03.04 - 09.04)
~~Lage møteinnkalling for veiledermøte 2~~	simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Skrive enhetstester for StreamManager.cs~~	simenklevengen		19.4.	Uke 15 (10.04 - 16.04)
~~Frontend Workshop~~	simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Sprint Review 4~~	simenklevengen		13.4.	Uke 15 (10.04 - 16.04)
~~Oppdatere JSON-server med de nyeste endringen~~	simenklevengen		21.4.	Uke 16 (17.04 - 23.04)
~~Skrive sammendraget på forsiden~~	simenklevengen		21.4.	Uke 16 (17.04 - 23.04)
~~Lag Bachelorpresentasjon (v1.0)~~	simenklevengen	!	26.4.	Uke 16 (17.04 - 23.04)
~~Justere Bachelorpresentasjonen basert på tilbakemeldinger~~	simenklevengen		3.5.	Uke 17 (24.04 - 30.04)

Vedlegg C – Medforfattererklæring



~~Ta bilde for gruppeoversikten~~	Daniel Eidsheim	3.5.	Uke 17 (24.04 - 30.04)
~~Korrekturlese Bacheloren~~	Daniel Eidsheim	28.4.	Uke 17 (24.04 - 30.04)
~~Gjennomføre Veiledermøte 2~~	Tobias Günther	3.5.	Uke 17 (24.04 - 30.04)
~~Verifisere at bildene i rapporten er leselig~~	simenklevengen	5.5.	Uke 18 (01.05 - 07.05)
~~Signere Bacheloren~~	simenklevengen	5.5.	Uke 18 (01.05 - 07.05)
~~Justere Bacheloren basert på tilbakemeldinger~~	Daniel Eidsheim	3.5.	Uke 18 (01.05 - 07.05)
~~Gjennomgå tilbakemeldinger for Bacheloren~~	simenklevengen	3.5.	Uke 18 (01.05 - 07.05)
~~Sprint Review 5~~	Daniel Eidsheim	5.5.	Uke 18 (01.05 - 07.05)
~~Levere førsteutkast av Bacheloren til Leila for tilbakemelding~~	Tobias Günther	3.5.	Uke 18 (01.05 - 07.05)

Dato	**Signatur**
05.05.2023	Daniel Eidsheim: 
05.05.2023	Tobias Günther: 
05.05.2023	Simen Klevengen: 

Vedlegg D – FURPS-analyse

Kravspesifikasjon

Gjeldende dokument beskriver de overordnede kravene som spesifiserer hvordan NATS administrasjonsportalen skal utvikles og fungere. Det vil si at gjeldende dokument beskriver egenskapene til løsningen som skal utvikles med hensyn på ytelse og funksjonalitet. For å sikkerstille at kravspesifikasjonen er av høy kvalitet benyttes FURPS-modellen.

FURPS-analyse

FURPS-modellen er et mye brukt rammeverk innenfor programvareutvikling som hjelper med å samle og kategorisere krav. Hensikten bak kategoriseringen er å sikre at løsningen oppfyller alle nødvendige kvalitetskrav som kunden eller sluttbrukeren etterspør [1].

Funksjonalitet (Functionality):

Tabell D-1 gir en detaljer oversikt over all funksjonalitet som skal eller burde inkluderes i NATS administrasjonsportalen. Dermed definerer tabellen alle funksjonelle og ikke-funksjonelle krav. I tillegg prioritet settes hvert krav etter MoSCoW metoden for å lettere definere omfanget av prosjektet.

Moscow metoden [1] er en ofte brukt teknikk innen prosjektstyring for å avgjøre viktigheten av krav eller funksjoner med hensyn på prosjektmålet. Selve ordet MoSCoW basere seg på de fire prioriteringsnivåene som teknikken går ut på: «Must have», «Should have», «Could have» og «Won't have».

Alle funksjonalitetskrav definert med «Must have» og «Should have» vil inkluderes i prosjekt. «Could have» kravene vil potensielt inkluderes i utviklingen, hvis tiden strekker til det. Krav definert med «Won't have» er ikke inkludert i dette prosjektet, men kan inkluderes i utviklingen av neste versjon av NATS administrasjonsportalen.

Tabell D-1: Funksjonalitetskrav

Type	Prioritet	Beskrivelse
Funksjonelt	Must have	Vise alle meldinger, som publiseres under (tilhører) et bestemt subject, i en oversikt.
Funksjonelt	Must have	Opprette ny melding på et spesifisert subject.
Funksjonelt	Must have	Se meldingsspesifikk informasjon som meldingsheader og payload.
Funksjonelt	Must have	Backend skal begrense antall tegn i meldingspayload i tilfeller hvor payload er for stor.
Ikke-funksjonelt	Must have	Payload til en melding skal ikke bearbeides eller konverteres.
Funksjonelt	Must have	Vise alle subjects i en oversikt.
Funksjonelt	Must have	Vise alle streams, som er tilgjengelig på en NATS-server eller i et cluster, i en oversikt.
Funksjonelt	Must have	Eksisterende meldinger kan slettes.
Funksjonelt	Must have	Mulighet for å kopiere meldingsheader og payload fra et subject til et annet.
Ikke-funksjonelt	Must have	All behandling av innhold til eksisterende meldinger skal foregå etter «Read only» prinsippet.
Funksjonelt	Must have	Alle handlinger som f.eks. lese-, flytte- og opprette meldinger m.m. skal spores og loggføres.
Funksjonelt	Must have	All loggføring skal knyttes opp mot fiktive brukerkontoer som er definert gjennom miljøvariabler.
Ikke-funksjonelt	Must have	All form for dokumentasjon og navnsetting skal foregå på engelsk, siden prosjektet er åpen kildekode.
Funksjonelt	Should have	Vise subject hierarkiet på en NATS-server.
Ikke-funksjonelt	Should have	Administrasjonsportalen må kunne rulles ut vha. Docker, slik at den kan kjøres i kubernetes.
Ikke-funksjonelt	Should have	Hemmelig informasjon som passord, IP-adresser m.m. skal lagres og håndteres vha. miljøvariabler.
Ikke-funksjonelt	Should have	Alle frontend komponenter skal opprettes uten unødvendig bruk av wrapping-tags (f.eks.).
Funksjonelt	Should have	Legge til JSON-server som et alternativ til backend under utvikling.
Ikke-funksjonelt	Could have	Dokumentere navigasjonsruter og generell navigering av UI vha. wireframes og navigasjonskart.
Ikke-funksjonelt	Could have	API endepunkter skal alltid verifisere at inndata er akseptabel. Dvs. at det ikke mangler obligatorisk data i f.eks. et JSON-objekt.
Funksjonelt	Could have	Melding- og stream oversikter kan filtreres og søkes i vha. subjects, stream, timestamp m.m.
Funksjonelt	Could have	Vise alle consumers i en oversikt.

Funksjonelt	Could have	Optimalisere API-forespørsler ved bruk av pagination.
Funksjonelt	Could have	Meldingspayload som er av datatype base64 skal konverteres til et lesbart format.
Funksjonelt	Won't have	Behandling av subjects (opprettelse og sletting) skal ikke være mulig.
Funksjonelt	Won't have	Administrasjonsportalen skal inkludere tilgangsstyring, slik at brukerrettigheter kan begrenses.
Ikke-funksjonelt	Won't have	Meldinger og deres innhold skal ikke være leselig (kryptert) for utenforstående parter når de ligger i en NATS-kø.

Brukbarhet (Usability):

Med tanke på at dette prosjektet skal produsere en første versjon av administrasjonsportalen vil brukervennligheten av løsningen være grunnleggende. Det vil si at løsningen må kunne representere informasjon på en hensiktsmessig og oversiktlig måte, samtidig som at den i stor grad skal ta hensyn til universell utforming. Dette betyr at det ikke følges retningslinjer som f.eks. WCAG (Web Content Accessibility Guidlines) [2] i denne omgangen, men nettleserutvidelsen WAVE [3] skal brukes for å sikkerstille at brukergrensesnittet kan brukes av individer med nedsatte funksjonsevner.

Videre må portalen også tillate for brukerinteraksjon i form av input. Dataen som kommer ut av denne type interaksjon skal automatisk kvalitetssikres slik at feilaktig eller ulovlig input ikke kan føre til uønsket oppførsel i applikasjonen.

Pålitelighet (Reliability):

Administrasjonsportalen må være robust og nøyaktig. Det vil si at alle deler av applikasjonen må være utstyrt med tilstrekkelig feilhåndtering. Slik kan det unngås at eventuelle programfeil eller uventet data kan føre til uønskede situasjoner eller applikasjonsoppførsel.

På toppen av dette må all data relatert til meldinger eller NATS-køer være pålitelig. Dette innebærer f.eks. at ingen meldinger som ligger i en NATS-kø må oversees/utelukkes fra å bli vist til sluttbrukeren. Et annet eksempel vil være at payload til en melding ikke må endres.

For å sikkerstille pålitelighet på kildekodenivå skal enhetstester brukes på alle .NET-klasser. Disse testene skal åpne for kjapp og effektiv verifisering av kildekode, og være til hjelp i feilsøkingsprosesser.

Et annet viktig krav som administrasjonsportalen må oppfylle, er det at bare bruker initierte aktiviteter faktisk utføres. Nærmere forklart betyr dette at administrasjonsportalen ikke skal utføre noen handlinger på meldinger eller NATS-køer med mindre brukeren har implisitt bedt om det. Derfor bør alle handlinger i portalen kreve en godkjenning av brukeren (f.eks. trykke på en OK-knapp) før den faktisk gjennomføres. I tillegg skal brukeren også få tilbakemelding fra administrasjonsportalen når en handling har blitt utført.

Ytelse (Performance):

Responstiden til applikasjonen bestemmes i all hovedsak av systemarkitekturen og kommunikasjonen mellom de ulike delen i løsningen. Tanken er at systemarkitekturen skal basere seg på Azure App Services i forbindelse med hosting av applikasjonen, dermed vil responstiden være kjapt hvis riktig kostnadsplan velges i forbindelse med hosting. Kommunikasjon blant de ulike dele i backend og NATS-server vil ha en optimal responstid, hvis alle disse tjenesten hostes på Azure plattformen.

Med tanke på utvikling vil bruk av Vite sørge for et raskt og effektivt utviklingsmiljø. Nærmere forklart muliggjør Vite opprettelsen av lokal utviklingsserver på maksimalt noen sekunder, samtidig som at hot loading er tilgjengelig [4]. Sammenliknet med andre utviklingsverktøy som tillater «bundling» av React kode, er Vite mye raskere [5].

Støttbarhet (Supportability):

Koden i hele løsningen skal være utformet på en skalerbar og fleksible måte. Dette innebærer at løsningen skal kunne videreutvikles uten at den eksisterende systemarkitekturen blir påvirket på en negativ måte eller er til hindring. Mer konkret betyr dette at ting som hardkoding av verdier skal unngås.

For å sikkerstille lang levetid på applikasjonen med hensyn på teknologi, skal bare ny teknologien benyttes under utviklingen. For backend betyr dette at .NET 7 skal brukes, mens frontend skal utvikles med React. I tillegg skal Vite og Typescript brukes i frontend for å sikkerstille at all utvikling kan foregå effektivt og at kildekoden er av høy kvalitet.

Plan for fremtiden er at administrasjonsportal skal videreutvikles, for å bistå med dette skal prosjektet også ha fokus på dokumentasjon. Nærmere forklart skal all installasjons- og konfigurasjonsarbeid i forbindelse med oppsett av utviklings- og produksjonsmiljøet dokumenters. Dette kommer ved siden av generell dokumentasjon av kildekoden og overordnet virkemåte av løsningen. Dokumentasjonen vil gjøre det lettere å få et overblikk over hva som er gjort og hvordan videreutviklingen kan håndteres.

Referanser

[1] J. Dyson, «Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements,» 07. Januar 2019. [Internett]. Tilgjengelig: https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson/. [Funnet 22. Februar 2023].

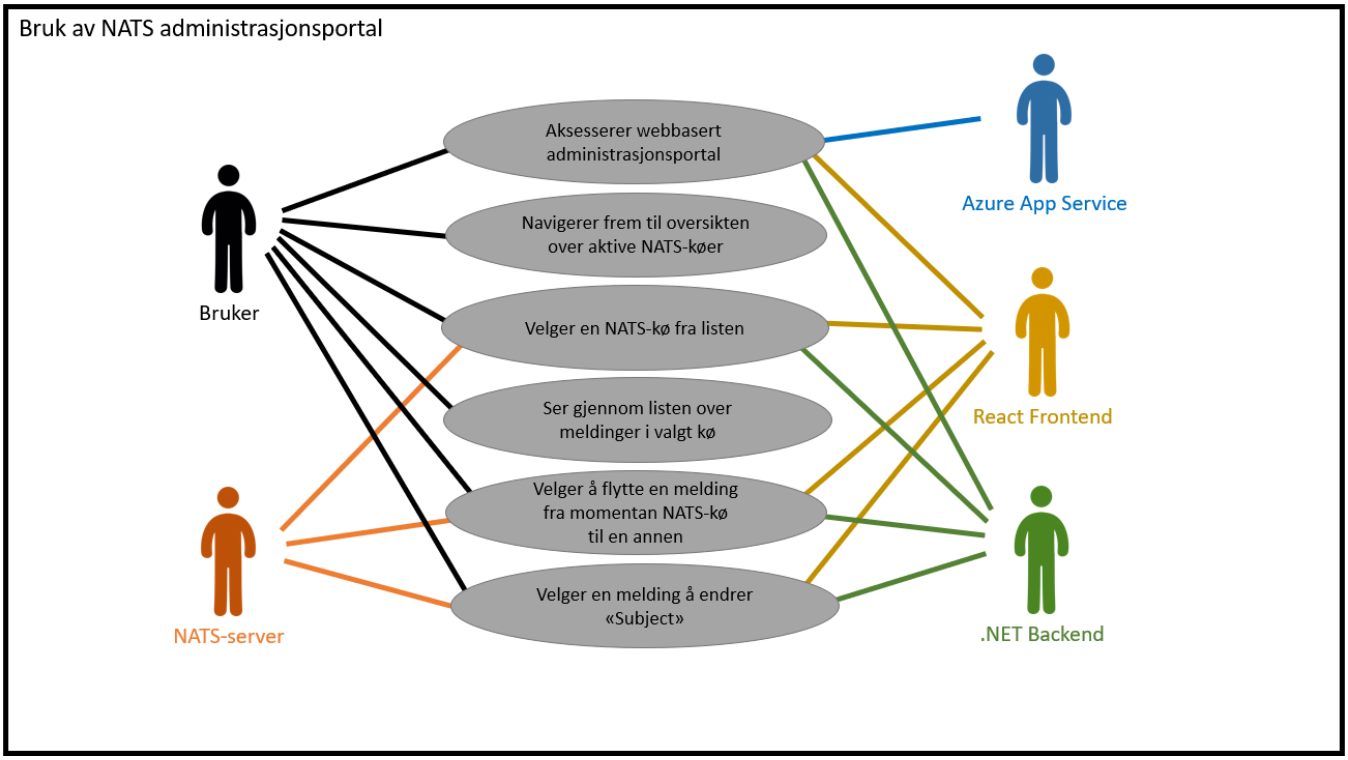
[2] S. L. Henry, «WCAG 2 Overview,» 25. Januar 2023. [Internett]. Tilgjengelig: https://www.w3.org/WAI/standards-guidelines/wcag/#intro. [Funnet 22. Februar 2023].

[3] WebAIM, «WAVE Browser Extensions,» [Internett]. Tilgjengelig: https://wave.webaim.org/extension/. [Funnet 24. Mars 2023].

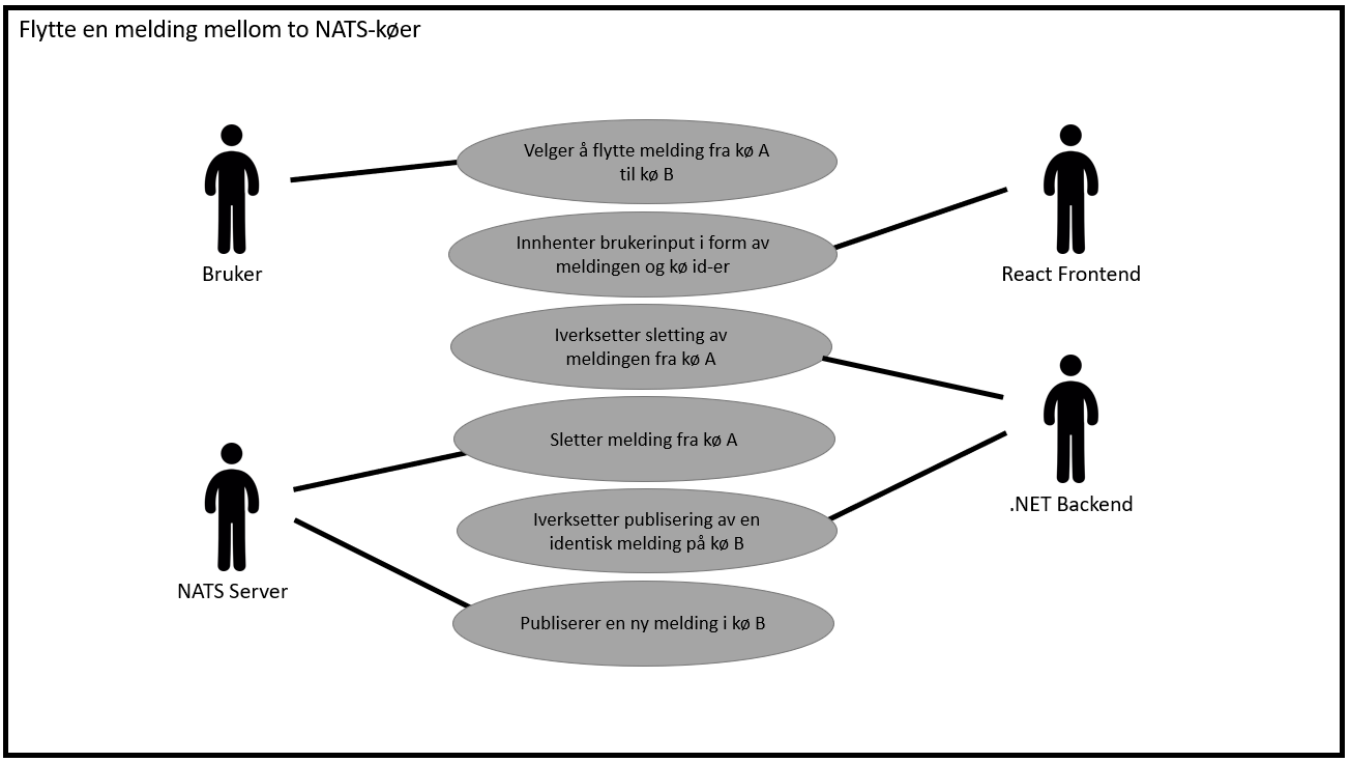
[4] GeeksForGeeks, «Difference between Hot Reloading and Live Reloading in React Native,» 03. Mars 2022. [Internett]. Tilgjengelig: https://www.geeksforgeeks.org/difference-between-hot-reloading-and-live-reloading-in-react-native/. [Funnet 10. Mai 2023].

[5] E. You, «Why Vite,» Vite, [Internett]. Tilgjengelig: https://vitejs.dev/guide/why.html. [Funnet 10. Februar 2023].

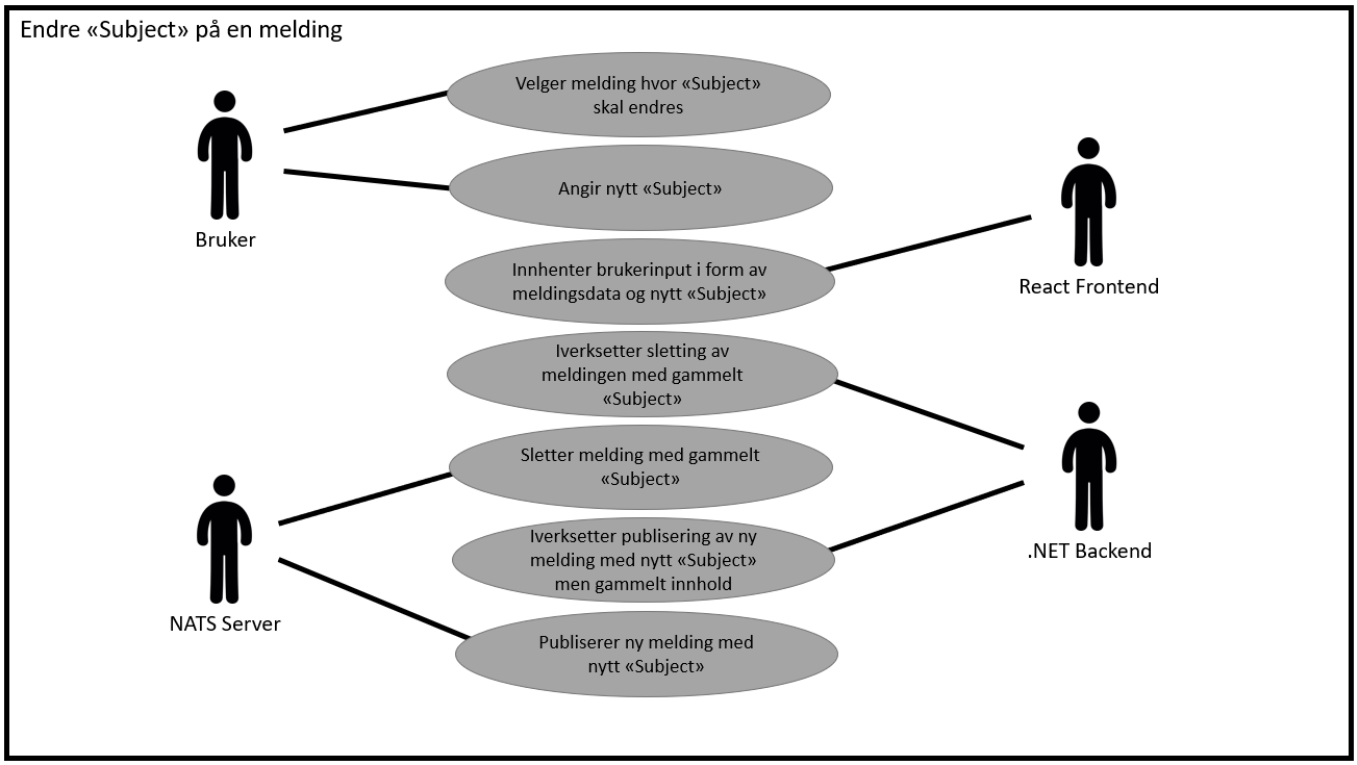
Vedlegg E – Use-case diagrammer



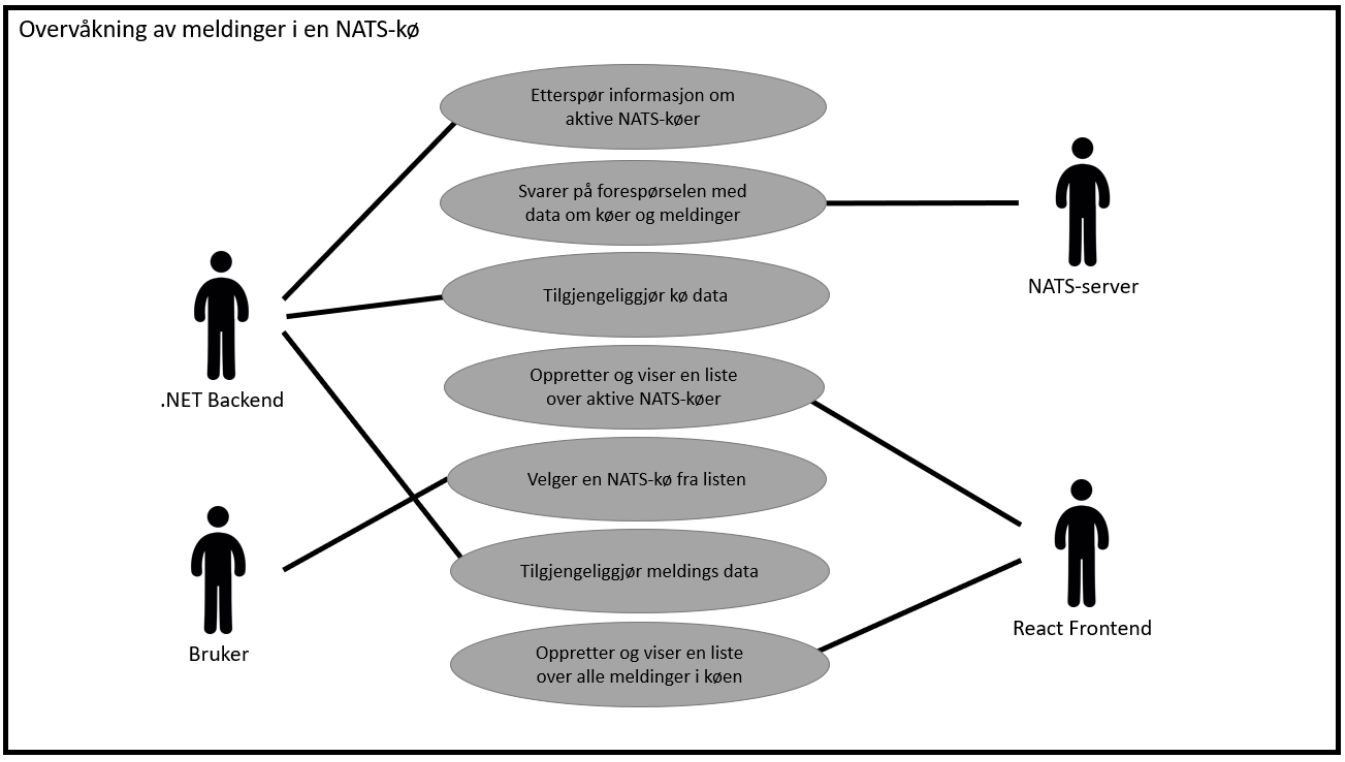
Figur E-1: Use-case diagram for «Bruk av NATS administrasjonsportal»



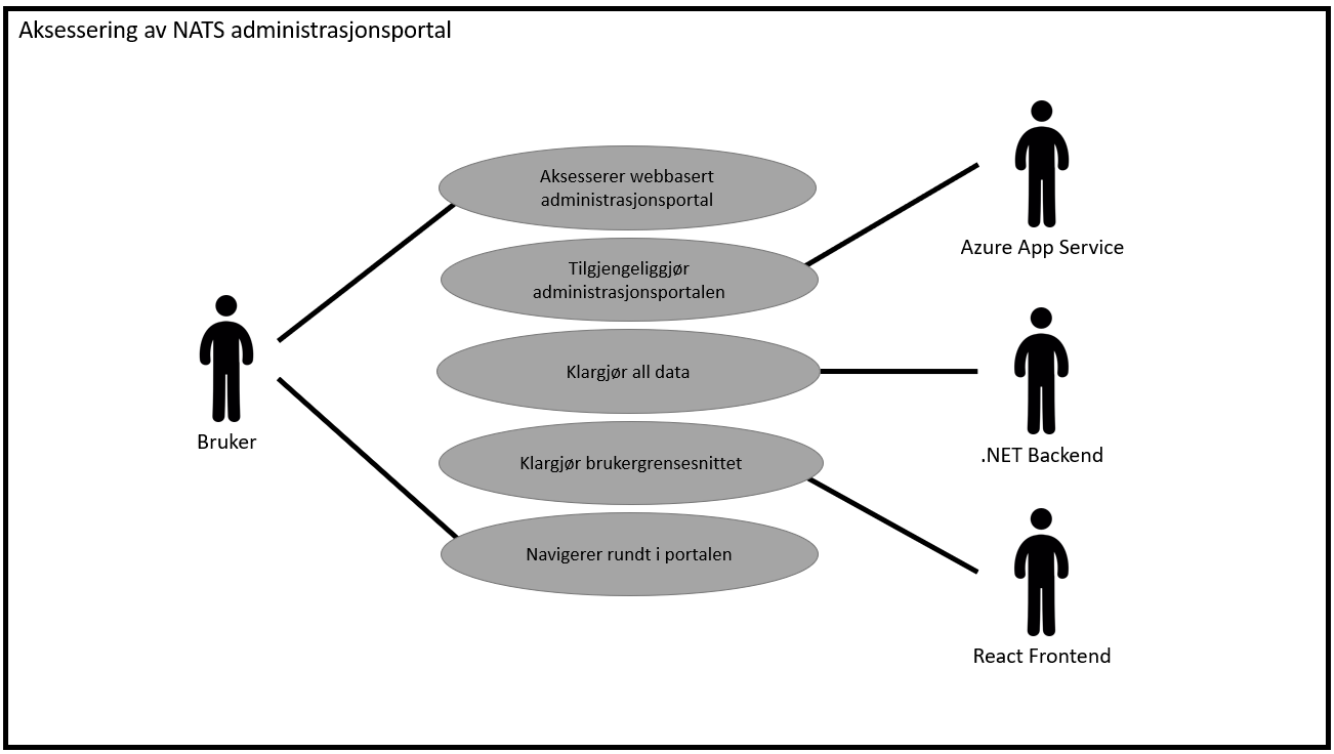
Figur E-2: Use-case diagram for «Flytte en melding mellom to NATS-køer»



Figur E-3: Use-case diagram for «Endre Subject på en melding»

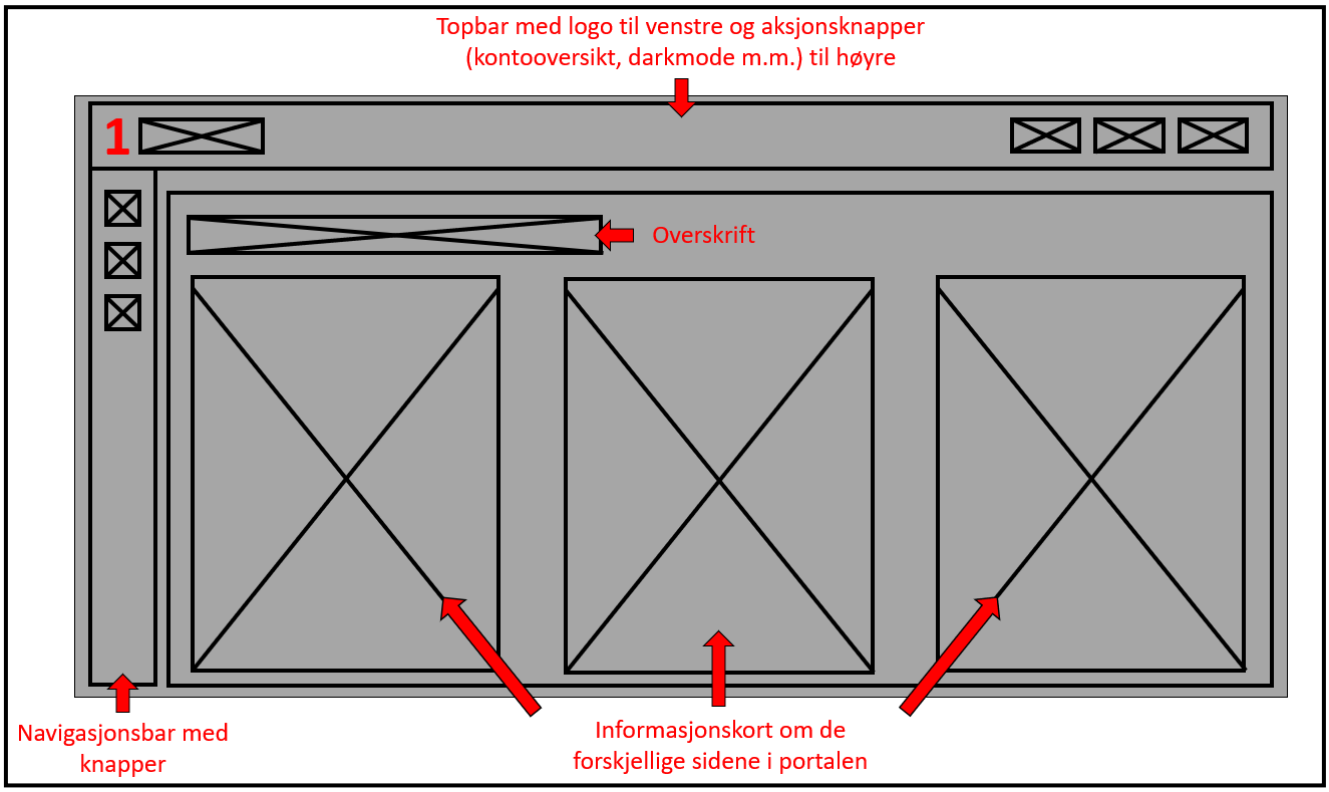


Figur E-4: Use-case diagram for «Overvåkning av meldinger i en NATS-kø»

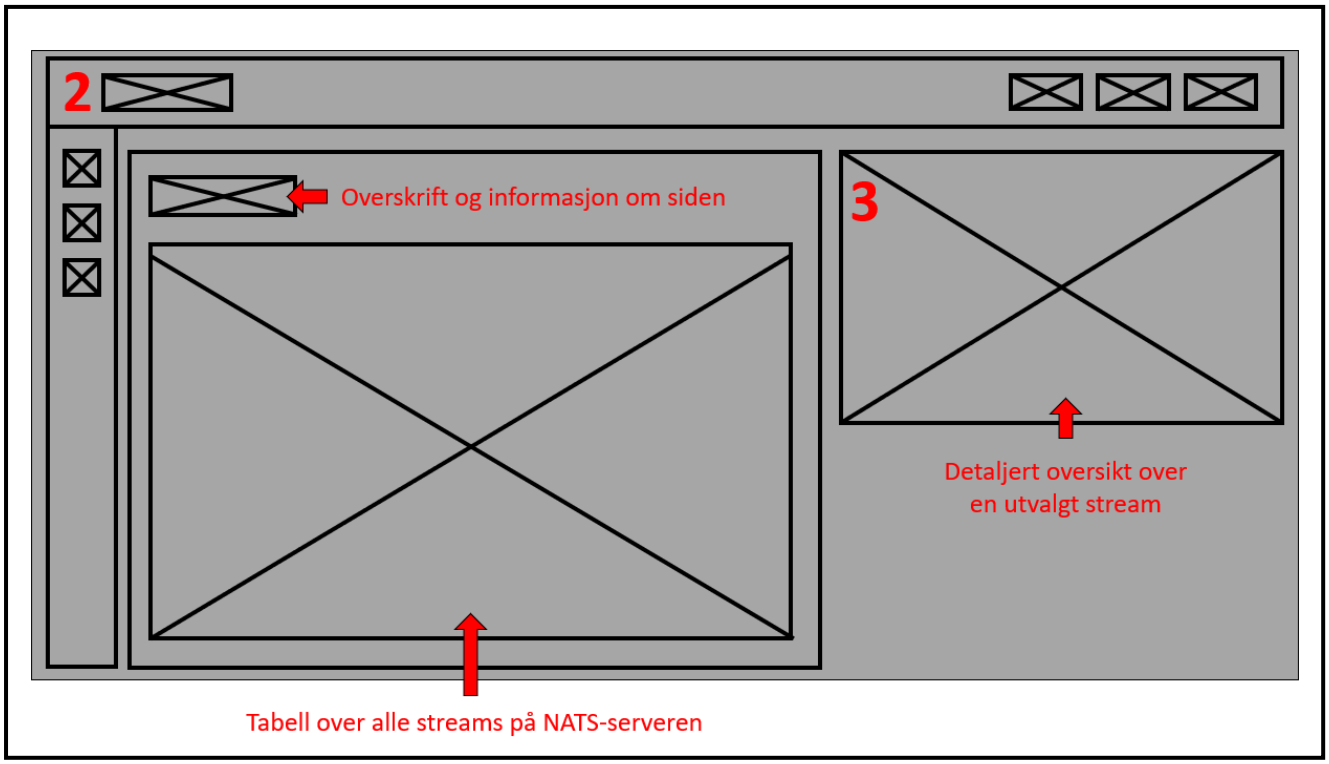


Figur E-5: Use-case diagram for «Aksessering av NATS administrasjonsportal»

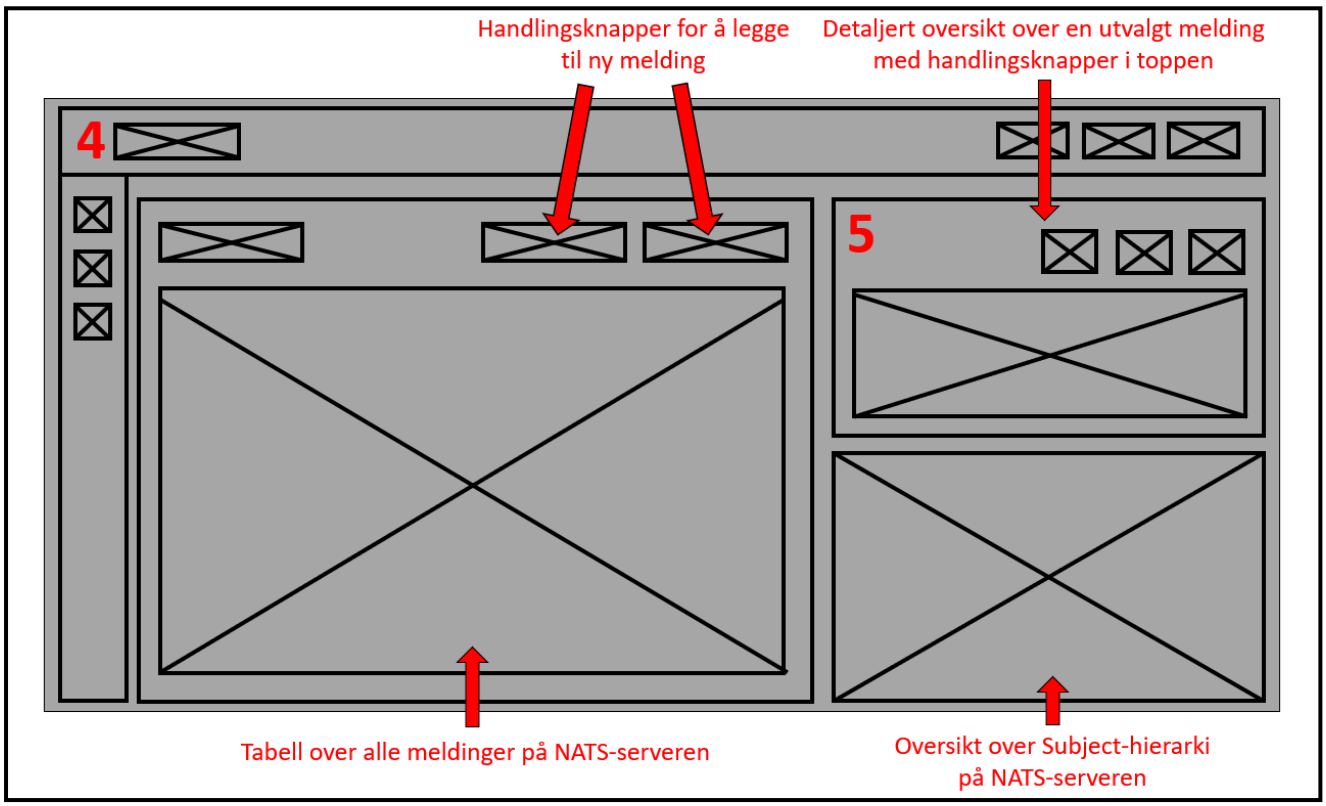
Vedlegg F – Wireframes



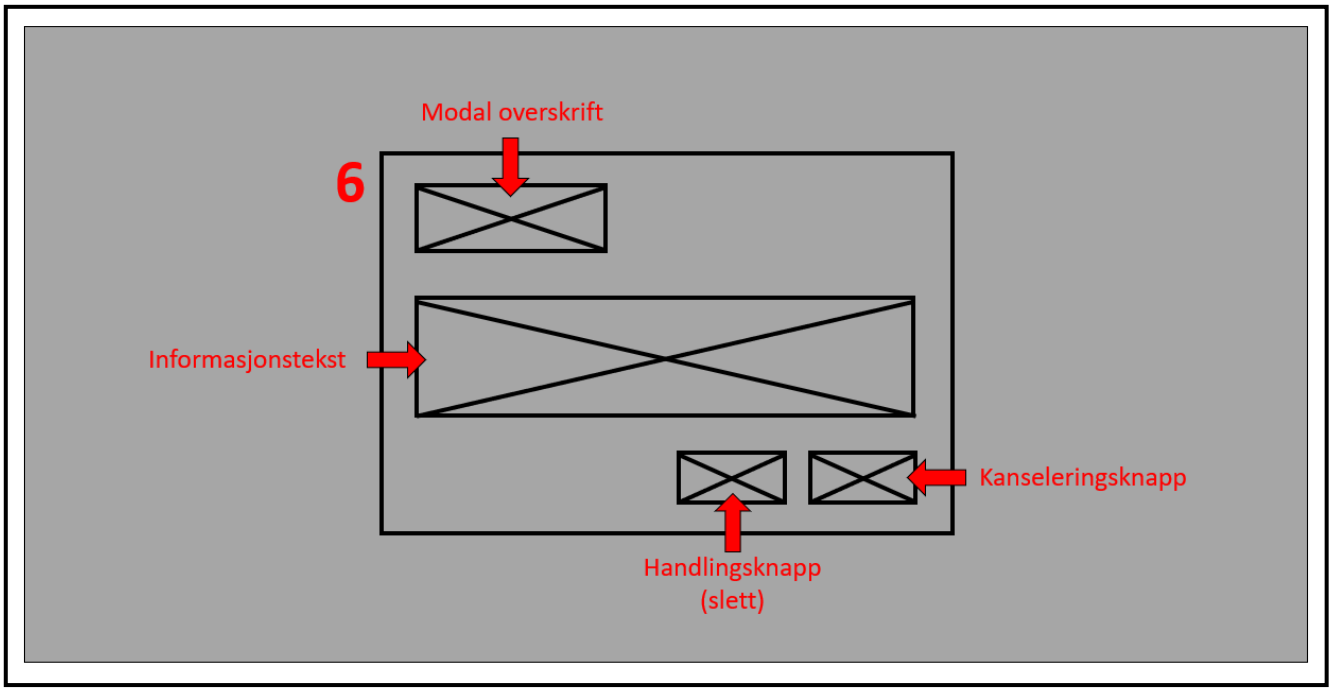
Figur F-1: Wireframe av «Hjemmesiden»



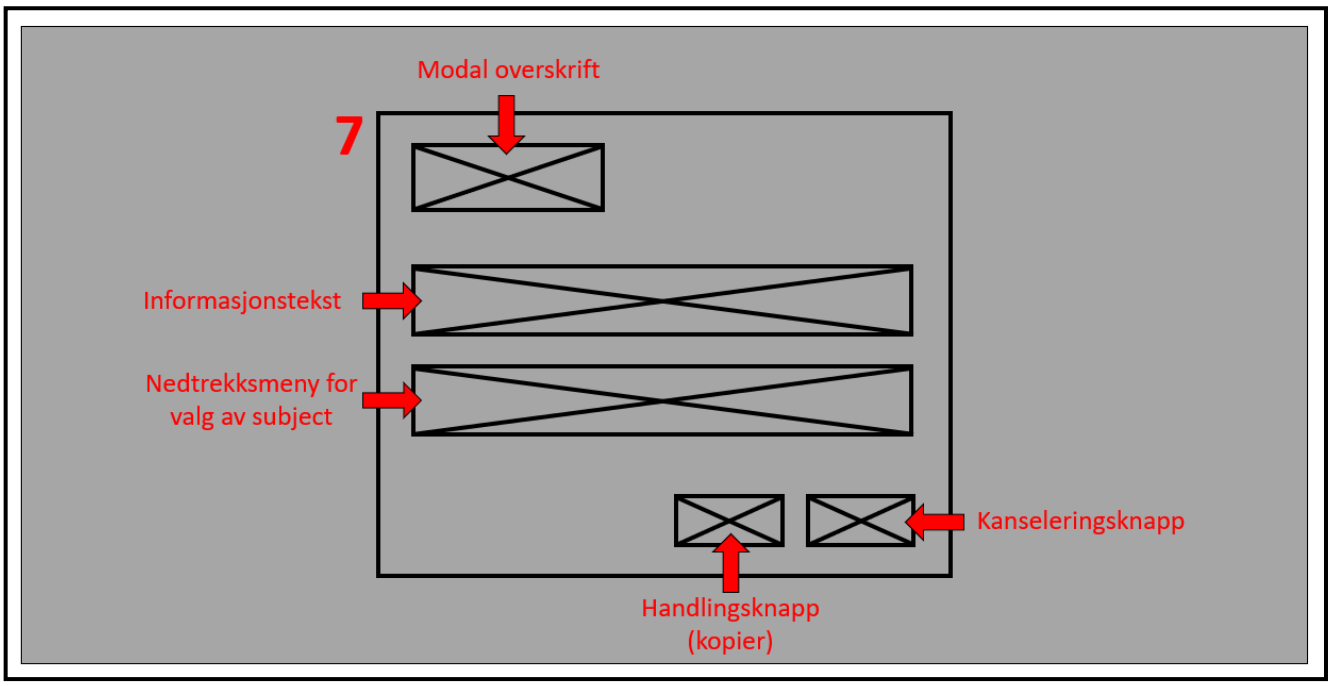
Figur F-2: Wireframe av «Streamsiden»



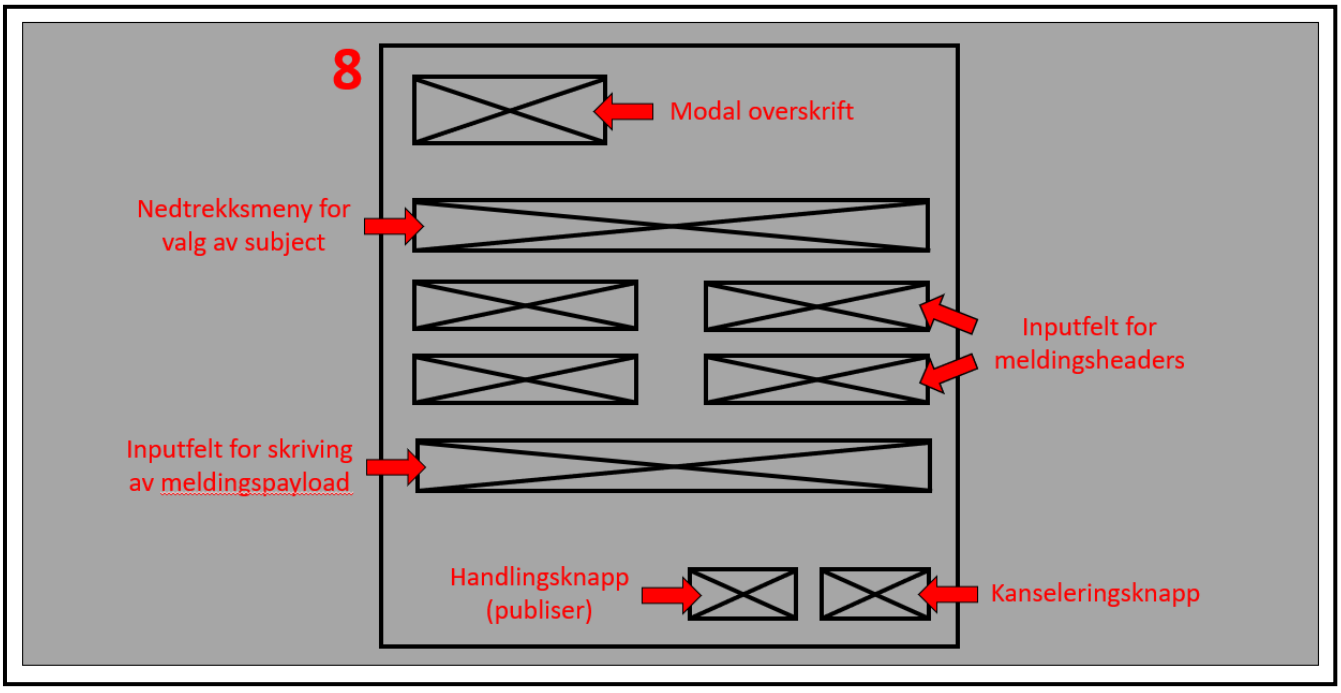
Figur F-3: Wireframe av «Meldingssiden»



Figur F-4: Wireframe av «Modalen for sletting av en meldingen»



Figur F-5: Wireframe av «Modalen for kopiering av en melding»



Figur F-6: Wireframe av «Modalen for publisering av en melding»

Vedlegg G – User Acceptance Tests

Tabell G-1: UAT av «meldingsoversikten»

Test Navn	Vise alle meldinger.		**Test ID**	1
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.			
Test Steg	1. Naviger til meldingssiden. 2. Vent på at alle meldingene laster inn.			
Forventet Resultat	En tabulert oversikt med alle meldingene på NATS-serveren.			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-2: UAT for «filtrering og søk i meldingsoversikten»

Test Navn	Søke etter én eller noen bestemte meldinger.		**Test ID**	2
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.			
Test Steg	1. Naviger til meldingssiden. 2. Vent på at meldingene laster inn. 3. Endre på filtre og bruk søkefelt.			
Forventet Resultat	Meldingstabellen viser bare meldinger basert på filtrene og søkeord.			
Faktisk Resultat	Når filter ble lagt på meldingstabellen ble det ikke vist noe. Dette kom av at tabellens sidenummer ikke ble tilbakestilt til side 1. Bortsett fra dette fungerte filtrering og søke.			
Status	**Godkjent**	**Avvist**		
		X		

Tabell G-3: UAT for «sletting av en melding»

Test Navn	Slette en melding.		**Test ID**	3
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på meldingssiden. • Det finnes minst én melding på NATS-serveren.			
Test Steg	1. Trykk på «View Message»-knappen på en melding i meldingstabellen. 2. Trykk på slettknappen i meldingsoversikt. 3. Velg om meldingen skal overskrives i slettingsmodalen. 4. Fullfør slettingen.			
Forventet Resultat	Den valgte meldingen forsvinner fra meldingstabellen.			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-4: UAT for «kopiering av en melding»

Test Navn	Kopiere en melding.		**Test ID**	4
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på meldingssiden. • Det finnes minst én melding på NATS-serveren.			
Test Steg	1. Trykk på «View Message»-knappen på en melding i meldingstabellen. 2. Trykk på kopieringsknappen i meldingsoversikt. 3. Velg subject i kopieringsmodalen. 4. Fullfør kopieringen.			
Forventet Resultat	En ny melding dukker opp i meldingstabellen under angitt subject og med kopiert meldingsinnhold (headers og payload).			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-5: UAT for «opprettelse av en melding»

Test Navn	Opprette en ny melding.		**Test ID**	5
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.			
Test Steg	1. Naviger til meldingssiden. 2. Trykk på publiseringsknappen i toppen av meldingstabellen. 3. Fyll ut formen med subject, headers og payload. 4. Fullfør publiseringen.			
Forventet Resultat	En ny melding dukker opp i meldingstabellen under angitt subject og med oppgitt meldingsinnhold (headers og payload).			
Faktisk Resultat	Det var ikke mulig å legge inn headers i publiseringsmodal. Dermed var det ikke mulig å opprette en ny melding.			
Status	**Godkjent**	**Avvist**		
		X		

Tabell G-6: UAT av «detaljert meldingsoversikt»

Test Navn	Se detaljert meldingsinformasjon.		**Test ID**	6
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.			
Test Steg	1. Naviger til meldingssiden. 2. Trykk på «View Message»-knappen i meldingsoversikten.			
Forventet Resultat	En oversikt med detaljert meldingsinformasjon åpnes på høyre siden av skjermen.			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-7: UAT for «nedkorting av meldingspayload»

Test Navn	Se nedkortet meldingspayload.	**Test ID**	7
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på meldingssiden. • Vet sekvensnummeret på en melding med stor payload.		
Test Steg	1. Trykk på «View Message»-knappen på meldingen med riktig sekvensnummer. 2. Finn «Payload» i meldingsinformasjonen på høyre siden av skjermen.		
Forventet Resultat	De første 200 tegn av den store payloaden vises etterfulgt av «…».		
Faktisk Resultat	Samme som forventet resultat.		
Status	**Godkjent**	**Avvist**	
	X		

Tabell G-8: UAT for «vising av hele meldingspaylod»

Test Navn	Vise hele meldingspayload.	**Test ID**	8
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på meldingssiden. • Det finnes en melding på med en payload større enn 200 tegn.		
Test Steg	1. Trykk på «View Message»-knappen på en melding med stor payload. 2. Trykk på «View full Payload»-knappen i meldingsoversikt.		
Forventet Resultat	En modal åpnes hvor hele meldingspayload vises.		
Faktisk Resultat	Samme som forventet resultat.		
Status	**Godkjent**	**Avvist**	
	X		

Tabell G-9: UAT av «streamsoversikten»

Test Navn	Vise alle streams.	**Test ID**	9
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.		
Test Steg	1. Naviger til streamsiden. 2. Vent på at alle streams laster inn.		
Forventet Resultat	En tabulert oversikt med alle streams på NATS-serveren.		
Faktisk Resultat	Samme som forventet resultat.		
Status	**Godkjent**	**Avvist**	
	X		

Tabell G-10: UAT av «detaljert streamoversikt»

Test Navn	Se detaljert stream informasjon.		**Test ID**	10
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.			
Test Steg	1. Naviger til streamsiden. 2. Trykk på «View Stream»-knappen i streamsoversikt.			
Forventet Resultat	En oversikt med detaljert stream informasjon åpnes på høyre siden av skjermen.			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-11: UAT av «subject-hierarkiet»

Test Navn	Vise alle subjects.		**Test ID**	11
Forutsetninger	• Logget inn med en konto som har riktige rettigheter. • Står på hjemmesiden.			
Test Steg	1. Naviger til meldingssiden. 2. Vent på at subject hierarkiet laster inn.			
Forventet Resultat	En oversikt over subject hierarkiet på NATS-serveren som inkluderer alle subjects på serveren.			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-12: UAT for «loggføring av lesing av meldinger»

Test Navn	Se loggføring av lesing av meldinger.		**Test ID**	12
Forutsetninger	• Har terminalen til backend åpen mens applikasjonen kjører. • Står på hjemmesiden.			
Test Steg	1. Naviger til meldingssiden. 2. Se i terminalen at ting loggføres. 3. Trykk på «View Message»-knappen-knappen i meldingsoversikten. 4. Se i terminalen at handlingen blir loggført.			
Forventet Resultat	En loggføring som sier at fiktiv brukerkonto ser alle meldinger og at en spesifikk melding sees på.			
Faktisk Resultat	Samme som forventet resultat.			
Status	**Godkjent**	**Avvist**		
	X			

Tabell G-13: UAT for «loggføring av meldingsbehandling»

Test Navn	Se loggføring av handlinger ifm. meldingsbehandling.	**Test ID**	13
Forutsetninger	• Har terminalen til backend åpen mens applikasjonen kjører. • Står på meldingssiden.		
Test Steg	1. Trykk på publiserings-, kopierings- eller slettingsknappen. 2. Se i terminalen at handlingen blir loggført.		
Forventet Resultat	En loggføring som sier at fiktiv brukerkonto enten har publisert-, kopiert- eller slettet en melding.		
Faktisk Resultat	Samme som forventet resultat, men loggføring av meldingskopiering må inkludere sekvensnummer til både opprinnelig og ny melding.		
Status	**Godkjent**	**Avvist**	
	X		

Tabell G-14: UAT for «bytting av aktiv brukerkonto»

Test Navn	Bytter brukerkonto.	**Test ID**	14
Forutsetninger	• Står på hjemmesiden.		
Test Steg	1. Trykk på kontoknappen i toppbaren. 2. Velg en annen brukerkonto i oversikten over andre tilgjengelige kontoer. 3. Fullfør bytting av brukerkonto.		
Forventet Resultat	Navn på den nye brukerkontoen vises under «Logged in as».		
Faktisk Resultat	Samme som forventet resultat.		
Status	**Godkjent**	**Avvist**	
	X		