Høgskulen på Vestlandet

BACHELOROPPGAVE

Videomøterom for lokale nettverk

Video conference room for local networks

Espen Kuvås - 151188

Espen Roll Karlsen - 133846

Kristian Åsnes - 181187

Dataingeniør/Informasjonsteknologi

Institutt for data- og realfag

Avdeling for ingeniør og økonomi

Innleveringsdato: 03.06.2019

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap

Institutt for data- og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel:	*Dato:*
Videomøterom for lokale nettverk	*03.06.2019*
Forfatter(e):	*Antall sider u/vedlegg: 52*
Espen Kuvås, Espen Roll Karlsen og Kristian Åsnes	*Antall sider vedlegg: 3*
Studieretning:	*Antall disketter/CD-er:*
Dataingeniør/Informasjonsteknologi	0
Kontaktperson ved studieretning:	*Gradering:*
Bjarte Kileng	Ingen
Merknader:	
Ingen	

Oppdragsgiver:	*Oppdragsgivers referanse:*
FieldCom AS	Ingen
Oppdragsgivers kontaktperson:	*Telefon:*
Terje Furenes	99692262

Sammendrag:

Denne rapporten beskriver prosessen for planlegging og utvikling av en applikasjon med videomøterom til bruk i lokale nettverk. Denne applikasjonen skal integreres i FieldCom sitt dokumentkontrollsystem og erstatte nåværende videoløsning som er avhengig av internettforbindelse.

Stikkord:

React	OpenVidu	Spring

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitenskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

Forord

Denne rapporten dokumenterer arbeidet på prosjektet "Videomøterom for lokale nettverk". Prosjektet ble fullført våren 2019, og er den avsluttende oppgaven på bachelorstudiet informasjonsteknologi og datateknologi ved Høgskolen på Vestlandet. Prosjektet er gjennomført av Espen Kuvås, Kristian Åsnes og Espen Roll Karlsen.

Prosjektet innebærer å utvikle en web-applikasjon der registrerte brukere kan opprette videosamtaler med hverandre. Applikasjonen er utviklet for å fungere på et lokalt nettverk, uten tilgang til internett. For å få dette til har vi benyttet flere eksisterende teknologier, rammeverk og utviklingsverktøy som vil bli nærmere beskrevet i denne rapporten.

Prosjektgruppen vil gjerne takke FieldCom for denne unike muligheten, og for god oppfølging underveis i prosjektet. Dette prosjektet har gitt oss verdifull kompetanse innenfor teknologier og rammeverk som vi ikke hadde fra før.

Vi vil også takke vår veileder fra Høgskulen på Vestlandet, Bjarte Kileng, for god veiledning underveis i utviklingsprosessen og med utarbeidelsen av bachelorrapporten.

Innholdsfortegnelse

1. Introduksjon

Dette kapittelet vil gi en beskrivelse av bakgrunnen for prosjektet. I de neste avsnittene blir mål og motivasjon, kontekst, begrensninger, ressurser og organisering av rapporten gått dypere i detalj.

1.1. Mål og motivasjon

Bacheloroppgaven er gitt i regi av FieldCom AS. FieldCom AS leverer verktøy for å optimalisere og forenkle dokumenthandtering, informasjonsflyt og samarbeid ved hjelp av smart bruk av web-teknologi. Det som står i senter for FieldCom er at løsningene deres skal være mobile og redigerbare, det skal være muligheter for samarbeid i sanntid og at de er skreddersydde og integrert. De har kunder innen skipsbygging, bygg og anlegg, energi og i form av leverandører. De leverer løsninger til både inn- og utland. Fellesnevner for prosjektene deres er at den største utfordringen er omkring informasjonsflyt og ineffektiv kommunikasjon.

Selskapet består av tre ansatte og ble stiftet i 2016. De har på denne korte tiden allerede opprettet seg god likviditet, lønnsomhet og soliditet. Firmaet er registrert med hovedkvarter i Måløy, men har også et kontor Bergen, hvor prosjektgruppens kontaktperson er stasjonert.

1.1.1. Problemstilling

FieldCom har diverse løsninger for å forbedre arbeidsflyten til bedrifter i hverdagen. Som for eksempel “Google Docs for PDF” som de har opprettet, der man kan redigere / merke dokumenter i pdf-format i sanntid. Denne prosjektoppgaven omhandler å utvikle en video-løsning i sanntid for lokale nettverk. Denne løsningen er først og fremst ment for lokasjoner der internett ikke er tilgjengelig. For eksempel på en oljeplattform der man på kontoret skal kunne ha videosamtale med en person på dekk for å vise frem noe. Dette er noe som er etterspurt fra kundene deres.

1.1.2. Mål

Målet for prosjektet er å få til en videomøterom-løsning for FieldCom, der alle med brukere på lokasjonen kan delta. Denne løsningen er noe som er etterspurt av kundene deres, de mener at dette er et produkt som kan forbedre informasjonsflyt og samarbeid. Prosjektgruppen har underveis tenkt på og lagt merke til at denne løsningen også er noe som kan forhindre misforståelser. Det er kommet inn rikelig med forslag med

brukstilfeller, men i første omgang er det de viktigste som blir tatt hensyn til. Etterhvert kan det implementeres flere og flere funksjoner.

1.2. Kontekst

Underveis i arbeidet til bedriften har de merket at et slikt produkt er ettertraktet og nødvendig for deres kunder. Produktet vil være en stand-alone tjeneste som ikke er integrert i noen andre produkter. I videreutviklingen av produktet vil bedriften vurdere om det vil ha noe for seg å implementere flere av deres produkter sammen som en felles tjeneste.

Bedriften har opplyst prosjektgruppen om de viktigste brukstilfellene og funskjonene de ønsker at løsningen skal ha. Bedriften kan senere eventuelt videreutvikle tjenesten basert på hva som er oppnådd i løpet av prosjektperioden. Gruppen ønsker selvsagt å utvikle et best mulig produkt for bedriften, og med de funksjonene bedriften mener deres kunder kan ha best nytte av.

1.3. Begrensninger

Prosjektgruppens største begrensning er tiden til rådighet. Når man får tildelt en bacheloroppgave vil det alltid være nye ting en må sette seg inn i. Det kan være spesielle metoder, programmeringsspråk og fremgangsmåter bedriften ønsker at man skal bruke for å utvikle produktet. Bedriften har gitt sine meninger med tanke på valg av verktøy de ønsker prosjektgruppen skal bruke og ikke bruke. Begrensningene som er viktigst er tiden som blir nyttet til utvikling, produkttesting og kunnskap om teknologier.

I startfasen av prosjektet og fram mot forprosjektrapporten har mesteparten av tiden gått til å lære om nye teknologier og den beste måten å sy disse sammen på for å få et best mulig produkt. Det regnes med at mye av tiden fremover vil bli brukt til testing av tjenester og teknologier som det kan være mulig å bruke.

Underveis i prosjektet vil man nok støte på forskjellige problemer som vil gå utover tiden man har beregnet til oppgavene. Dette kan også slå begge veier, altså noen oppgaver kan ta kortere tid enn antatt og noen kan ta lengre tid enn antatt.

De mest grunnleggende teknologiene har prosjektgruppen fått innføring gjennom studiet. Her i bacheloroppgaven er det mange teknologier som peker i riktig retning mot et flott sluttprodukt. Det viktige er å velge de beste samt viktigste teknologiene innenfor en forholdsvis kort tidsramme for at hovedmålet med prosjektet skal nås.

1.4. Ressurser

Ressursene som blir diskutert i dette avsnittet er bedriftens ressurser og litterære ressurser.

1.4.1. Bedriftens ressurser

Bedriften har gitt nokså frie tøyler for prosjektperioden, men med en forståelse for at man informerer dem underveis og er på "samme bølgelengde" underveis i prosjektet. Ressursene som er tilbudt er blant annet å nytte lokalene deres dersom det er ønskelig ved bryggen i Bergen.

Kontaktpersonen i bedriften ga tidlig uttrykk for at han ønsket god kommunikasjon og at det ikke skulle nøles med å ta kontakt dersom det var noe man lurte på eller hadde vansker med. Prosjektgruppen har kommunikasjon med kontaktpersonen både på Skype og via tekstmelding dersom det skulle være noe som haster.

FieldCom har også delt GitLab med prosjektgruppen slik at de kan se over de siste endringene som er utført i koden og lignende, noe som ofte gjør problem underveis i prosjektet lettere å forklare. Databaser, tegninger, brukstilfeller, diagram og diverse tips har de også delt åpent med prosjektgruppen.

1.4.2. Litterære ressurser

Underveis i prosjektet kommer prosjektgruppen til å trenge forståelse for verktøyene som skal brukes i bacheloroppgaven, både for å skrive rapporten og for å utvikle produktet. For utvikling av produktet trenger prosjektgruppen innføring i grunnleggende utviklingsmetoder, programmeringsspråk og verktøy som passer for produktet. I både produktutviklingen og i skriving av bacheloroppgaven, vil prosjektgruppen benytte kunnskapen som er gitt gjennom studiet på best mulig måte.

1.5. Organisering av rapporten

Rapporten består av 9 kapitler

Kapittel 1 gir en introduksjon til prosjektet, hvem prosjektgruppen jobber med, hva de jobber med, hva problemstillingen er og hva målet for prosjektet er. Begrensninger og ressurser blir også diskutert i dette kapittelet.

Kapittel 2 gir et overblikk av prosjektet, og en mer detaljert oversikt i hva produktet skal inneholde.

Kapittel 3 gir en beskrivelse av designet av produktet, hvordan prosjektgruppen skal gå fram og hvordan de skal gjøre det. Kapittelet sier noe om hvilke metoder og verktøy som skal benyttes, og hvordan produktet skal ta form før prosjektet starter.

Kapittel 4 bygger videre på kapittel 3 og gir en mer detaljert informasjon om produktdesignet.

Kapittel 5 gir en gjennomgang av hvilke metoder prosjektgruppen har brukt for å evaluere prosjektet, hvordan man kan vise at prosjektet har blitt løst på best mulig måte og hva som gjør løsningen bra og/eller mindre bra.

Kapittel 6 viser til diskusjon og refleksjon om hvordan prosjektet har gått, forventninger, problemer og utfordringer vil også bli reflektert over.

Kapittel 7 utdyper konklusjonene om prosjektet og veien videre etter prosjektperioden er avsluttet.

Kapittel 8 viser til referanser og litteratur brukt for å kunne svare på spørsmål underveis i prosjektet.

Kapittel 9 viser til alle vedlegg som har blitt brukt for prosjektet, akronymer og ordliste.

2. Prosjektbeskrivelse

I dette kapitlet beskrives bakgrunnen for prosjektet. Avsnittene under ser nærmere på produkteier, arbeidet før prosjektstart, kravspesifikasjon, idéen for løsningen og funksjonalitet.

2.1. Bakgrunn

Bakgrunnen for denne bacheloroppgaven er å utvikle ny funksjonalitet for å forbedre FieldCom sitt allerede eksisterende produkt. Det eksisterende produktet er et dokumentkontrollsystem for skipsdesign. Systemet fungerer som "Google Docs for PDFs" hvor folk i feltet kan arbeide på samme dokument, sette inn notater, markeringer, bilder og videoer i sanntid. Denne funksjonaliteten er kombinert med et video-chattesystem hvor de i dag bruker Twilio-video.

Twilio-video er avhengig av servere ute på internett. FieldCom ønsker å utvide sitt system med et videomøterom som kan benyttes i et lokalt nettverk (LAN), uten internettilgang. Dette er mulig å utvikle ved hjelp av en teknologi kalt WebRTC og STUN/TURN-servere. Dette er tjenester som installeres på en server og plasseres i det lokale nettverket.

2.2. Produkteier

FieldCom har alle rettighetene til sluttproduktet og kan bruke det i sin kommersielle virksomhet. Løsningen er tenkt å bli en integrert del av en allerede eksisterende applikasjon og eies av bedriften. Prosjektgruppen har ingen eierskap til produktet, men har lov til å omtale og vise det frem i oppgaven. Aktuelle kodeutsnitt i den endelige rapporten må godkjennes av FieldCom. Bedriften har selv utledet retningslinjene og kravspesifikasjonen for produktet. Dersom løsningene viker fra kravspesifikasjonen må kontaktperson ved bedriften konsulteres.

2.3. Tidligere arbeid

Denne bacheloroppgaven går ut på å forbedre et allerede eksisterende produkt, "Google Docs for PDF's", gjennom utvikling av ny funksjonalitet. Prosjekt er bygget på en idé av oppdragsgiver FieldCom, som ønsker å forbedre sitt produkt. Det er oppdragsgiver som har informert om aktuelle teknologier og API'er som kan benyttes i en slik videoløsning. Det er ikke blitt jobbet med denne ideen i firmaet tidligere, så dette er ganske nytt for begge parter.

Store deler av bachelorprosjektet blir bygget på open source kode, eks. Coturn og OpenVidu. Dette betyr at man kan bruke deler eller hele koden gratis, men at man må følge lisensbetingelsene som medfølger prosjektet. Dette kan innebære å gjøre den nye koden tilgjengelig for andre.

2.4. Kravspesifikasjon

I prosjektbeskrivelsen er det gitt generelle krav fra bedriften som omhandler løsningen, utviklingsmetoder og verktøy. Det anses som viktig at disse kravspesifikasjonene blir fulgt for å oppnå et resultat som har ønsket funksjonalitet og som lett kan videreutvikles. Disse kravene er beskrevet i punktene under.

2.4.1. Backend

Kravspesifikasjonene for backend delen av applikasjonen innebærer design og utvikling av en videokonferanse/WebRTC for et system på lokalt nettverk (ikke koblet til internett). For STUN/TURN servere kan den eksisterende løsningen Coturn brukes. Til utvikling skal Java frameworks som DropWizard eller Spring brukes.

Krav:

- Opprette p2p videokonferanse
- Opprette videokonferanse med flere brukere
- Generere URI'er til videorom
- Autorisere brukere ved bruk av jwt
- Kontroll på online/offline brukere
- Rest-API, eks. Sette opp videosamtaler, autorisering etc
- Websockets for håndtering av online brukere og videokonferanse oppdateringer
- Bonus: Undersøke kompatibilitet med skype for business

2.4.2. Frontend

Kravspesifikasjonene for frontend delen av applikasjonen innebærer design og utvikling av en web/native app til video løsningens backend. React.js eller react-native er foretrukne rammeverk for frontend.

Krav:

- Se online brukere
- Se og opprette videokonferanse med online "FieldCom brukere"
- Del link til p2p eller videokonferanse til andre
- Lager CSS og styling selv inntil videre. Minst mulig bruk av farge.

2.4.3. Dokumentasjon

Skriv klar og tydelig kode med beskrivende kommentarer. Kommentering i koden skal være på engelsk. Flytdiagrammer/tegninger må lages i tilfeller der kompleksiteten tilsier det.

2.5. Løsningsidè

I møte med oppdragsgiver FiedCom har blant annet kravspesifikasjonene, brukstilfeller og arkitektur blitt diskutert. Dette var viktig for å planlegge tilsvarende i pre-prosjektfasen og vite hvilke utfordringer som skal håndteres først.

2.5.1. Funksjonalitet

Figuren under er utformet av FieldCom og viser brukstilfellene i applikasjonen. Brukstilfellene implementeres etter prioritet og brukes som en del av fremdriftsplanen. Brukstilfellene prioriteres fra øvers til nederst på figuren under. Prosjektgruppen vurderer brukstilfellene som omfatter login/logout og brukerhåndtering som de viktigste. Når disse er på plass vil det være en kjørende applikasjon som kan lastes opp på testserveren som FieldCom har opprettet for oss (AWS).

Figur 2.1: Brukstilfeller. Prioriteres fra topp til bunn.

Prosjektgruppen og oppdragsgiver har blitt enig om følgende overordnede fremdrift forslag:

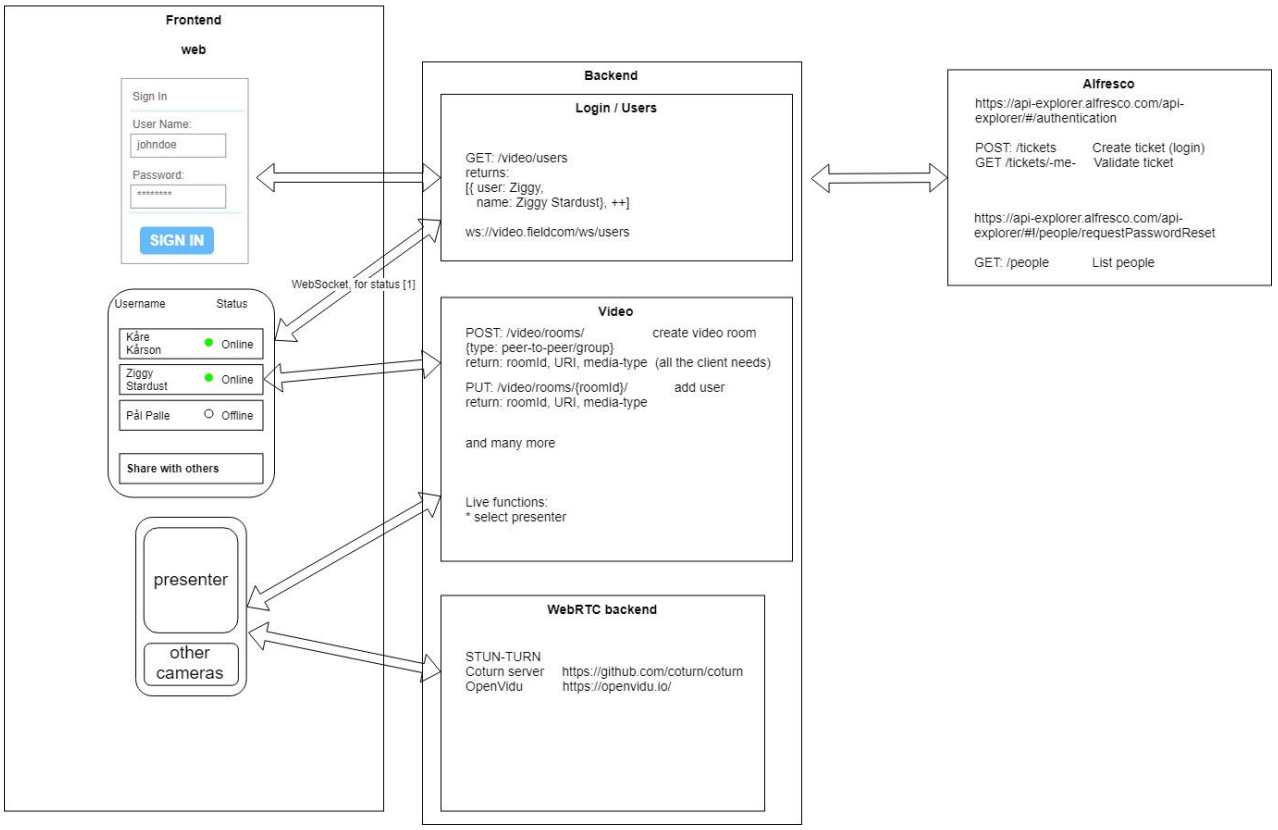
1. Opprette p2p video mellom to klienter
2. Login og brukerhåndtering
3. Sy sammen.

De ulike i brukstilfellene i figur 2.1 kan sorteres etter punkt nr.1 og punkt nr.2 i denne fremdriftsplanen. På punkt nr.1 vil prosjektgruppen i første omgang kun fokusere på opprettelse av videokommunikasjon mellom 2 klienter, og de tekniske utfordringene dette innebærer. På punkt nr.2 forventer prosjektgruppen å implementere brukstilfellene "login",

"logout" og "view other users" før man kan gå videre til punkt nr.3. Når disse 3 punktene er gjennomført kan resterende brukstilfeller implementeres.

2.5.2. Arkitektur

Figuren under er utformet av FieldCom. Prosjektgruppen har gjort egne valg og tilpasninger der kravspesifikasjonene tilsier det. Løsningen blir en web-applikasjon bestående av en backend-modul og en frontend-modul. I kapittel 4 om detaljert design blir det gitt en nærmere beskrivelse av de ulike komponentene i figuren og kommunikasjonen mellom dem.



Figur 2.2: Initiell skisse av modulene i applikasjonen

3. Prosjektdesign

I løpet av den innledende delen av prosjektet har det blitt undersøkt hva som allerede eksisteres av applikasjoner, og vurdert om disse kan brukes i produksjonen av programvaren. I denne prosessen har det også blitt testet ut teknologier for hvordan dette kan implementeres på egenhånd. Dette kapittelet beskriver noen av fremgangsmåtene prosjektgruppen har vurdert, og detaljer rundt programmeringsspråk og metoder.

3.1. Mulige fremgangsmåter

I den utsendte prosjektbeskrivelsen ble det oppgitt visse krav til spesifikasjoner for backend og frontend av prosjektet beskrevet i 2.4. Med disse kravene i bakhodet, tok det lenger tid å finne et passende prosjekt å bruke til integrering av funksjoner.

3.1.1. Alternativ fremgangsmåte 1: SimpleWebRTC

SimpleWebRTC er et JavaScript-bibliotek for bygging av WebRTC applikasjoner basert på React-moduler. Denne løsningen krever nettilgang da bruken baserer seg på en API-nøkkel mot deres server. Denne løsningen er ikke gratis og gir ikke tillatelse til videresalg. (SimpleWebRTC, u.å.)

3.1.2. Alternativ fremgangsmåte 2: NextRTC

NextRTC er et Java-bibliotek med egen signaling-server. Man kan kjøre den som egen web-applikasjon eller legge til i et eksisterende Spring-prosjekt. NextRTC bruker WebSockets til kommunikasjon mellom klienter. Dette prosjektet blir ikke lenger oppdatert (NextRTC, u.å.).

3.1.3. Alternativ fremgangsmåte 3: Jitsi

Jitsi er en JavaScript-applikasjon for produksjon av videokonferanserom. Den er gratis og open source, og kommer i en egen installeringspakke for Debian/Ubuntu-server med Node.js som runtime-system (Jitsi, u.å.).

3.1.4. Alternativ fremgangsmåte 4: OpenVidu

OpenVidu WebRTC Platform gjør det enkelt for utviklere å bygge sitt eget videokonferanserom. Denne inkluderer Kurento Media Server som gjør brukstilfeller lettere å håndtere. Ved bruk av dette prosjektet følger man en enkel API, det man trenger å lage er en klientside og en serverside, resten blir tatt hånd om av OpenVidu-serveren (OpenVidu, u.å.) med oppsett av Coturn og Kurento Media Server. Installasjonsveiledning finnes på dems nettside.

3.1.5. Alternativ fremgangsmåte 5: Egenproduksjon

En alternativ løsning er å produsere backend og frontend selv basert etter de spesifikasjonene som FieldCom ønsker. Dette inkluderer implementering mot WebRTC

API, Alfresco API og oppsett av egen signaling-, STUN/TURN-server og håndtering av media. Dette er en kompleks og tidkrevende prosess.

3.1.6. Diskusjon for valg av fremgangsmåte

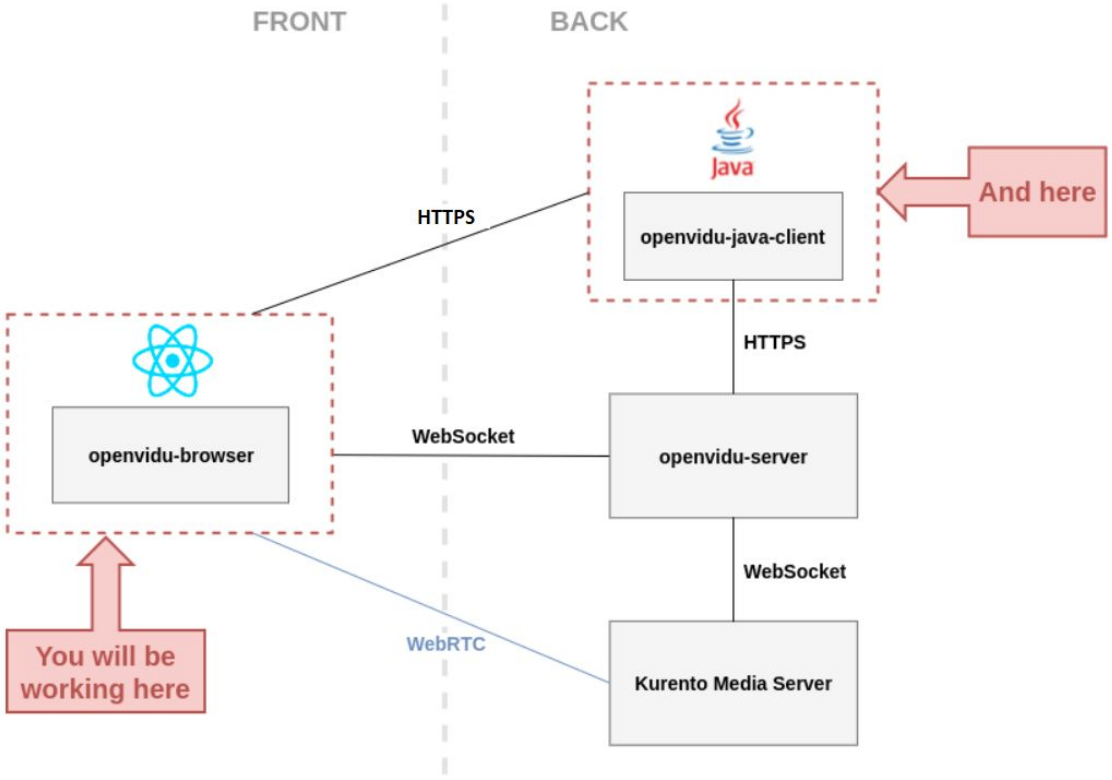
Prosessen for valg av fremgangsmåte har bestått av mye testing av eksisterende prosjekter, noen av de er nevnt over, for å kontrollere om det ville være mulig å integrere det mot programvaren som skal brukes. For å følge spesifikasjonskravet til FieldCom er det mange prosjekter som er blitt utelatt.

Prosjektgruppen så på hvordan det ville være å utvikle vår egen signaling-server integrert med Spring Boot. Ettersom det er begrensning på tid og erfaring kunne dette føre til at man ikke kommer i mål, eller kunne i så fall være begrenset med brukstilfeller i forhold til ønsker fra FieldCom. Det ble vurdert om NextRTC ville være en god løsning, men ettersom det prosjektet ikke lenger blir oppdatert ble det valgt å se videre hvor det til slutt havnet på løsningen fra OpenVidu. Dette vil være tidsbesparende og på OpenVidu sine nettsider har de utviklingseksempler med både Spring Boot og React som gir prosjektgruppen et godt utgangspunkt til videre utførelse, selv om disse er veldig enkle.

3.2. Spesifikasjoner

Dette prosjektet vil bygge på spesifikasjoner fra FieldCom og valg som gjøres underveis i prosessen. Hovedsakelig vil applikasjonen bli brukt sammen med eksisterende rammeverk, dette vil si Spring Boot i Java for backend og React i JavaScript for frontend. FieldCom har satt opp en AWS-server til dette formålet hvor alle servere og tjenester vil bli implementert. WebRTC er et prosjekt som tilbyr nettlesere og mobilapplikasjoner RTC-kapasiteter via API'er (WebRTC, u.å). Applikasjon for håndtering av media vil bli håndtert mot OpenVidu's egen server-API og Kurento Media Server installert på AWS som kommer i egen pakke.

For at to klienter skal kunne starte en videosamtale, må det etableres en forbindelse mellom de. Dette blir gjort gjennom "signaling" som lar to klienter sende metadata for å koordinere kommunikasjonen (Frozen Mountain, u.å.). I dette tilfellet blir det opprettet en WebSocket fra klientens side mot OpenVidu-serveren som kjører på AWS'en som håndterer utvekslingen av informasjonen til de aktuelle motpartene.



Figur 3.1: Redigert bilde fra OpenVidu Tutorials MVC Java

For at WebRTC skal kunne kommunisere med klienter bak brannmurer og NAT-nettverk, behøves det en tjeneste som samler inn informasjonen om klientens sti slik at to datamaskiner kan nå hverandre (Frozen Mountain, u.å.). Her vil det bli brukt en egen Coturn-server som er en implementasjon av STUN- og TURN-server i ett.

3.3. Valg av verktøy og programmeringsspråk

Ubuntu Server
Dette prosjektet vil benytte seg av tjeneste fra AWS som server. For å kunne kommunisere med den og kjøre kommandoer benyttes det av PuTTY som er en SSH- og telnetklient for Windows (PuTTY, u.å). Prosjektgruppen har begrenset med tilgang til brukeren på AWS sine nettsider, men har i senere tid fått tildelt tilgang til å restarte serveren og åpne porter.

Alfresco
FieldCom benytter seg av Alfresco for håndtering av brukerdata. Prosjektgruppen har fått sin egen instans med brukere som kan brukes underveis i utviklingen og til testing.

GitLab
For deling av kode vil det bli brukt versjonskontrollsystemet til GitLab, hvor hver bruker som har tilgang kan laste opp eller ned kode som gjør det lettere for prosjektgruppen å samarbeide på samme "repository". GitLab er valgt av FieldCom hvor det har blitt gitt tilgang til FieldCom sine prosjekter for backend og frontend.

IntelliJ IDEA
IntelliJ er en Java IDE for utvikling av programvare. Dette programmet er som standard en betalingsvare, men det gis ut gratis lisenser til studenter. IntelliJ har mange innebygde ferdigheter som for eksempel kodedeling via versjonskontrollsystemer, integrert Spring Boot og React, og tilgang til et marked med plugins. Den har også støtte for flere programmeringsspråk slik at prosjektgruppen kan forholde seg til ett IDE for frontend i JavaScript og backend i Java.

3.4. Prosjektets utviklingsmetode

I dette avsnittet blir det gitt kjennskap til utviklingsmetoden som er valgt for prosjektet, prosjektplanen med GANTT-skjema og en risikovurdering via et risikoanalyseskjema.

3.4.1. Utviklingsmetoden Scrum

Som utviklingsmetode i dette prosjektet vil det bli fulgt metoden Scrum. Scrum er et rammeverk der man kan takle komplekse adaptive problemer, samtidig som man produktivt og kreativt leverer produkter av høyest mulig verdi (Scrum, u.å.). Metoden er iterativ hvor man fordeler oppgaver som skal utføres over en tidsperiode på to uker, hvor man så implementerer resultatet inn i den fungerende versjonen og lager en ny plan for neste iterasjon, denne perioden kalles en “sprint”. På denne måten har man alltid en kjørende versjon tilgjengelig som vil vokse over tid og man kan utvikle brukstilfeller på en effektiv måte.

3.4.2. Prosjektplan

Det er opprettet et GANTT-skjema som viser planlegging og framdrift for prosjektet.

3.4.2.1. Tidsbruk

Tidsplanen viser til uker som er planlagt i prosjektet. Hver uke representerer en full arbeidsuke. De vil si at hvert medlem i prosjektet regner med å bruke 37,5 timer i uken på prosjektet. Dermed vil medlemmene til sammen bruke 112,5 timer hver uke det er oppført arbeid.

Sprinter	Tidsplan (Uke nr)													
	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Innlevering M2														
Sprint 1 - Orientering og planlegging														
Statusmøte														
Forprosjektrapport og presentasjon														
Sprint 2														
Statusrapport														
Sprint 3														
Blog innhold														
Sprint 4														
Sprint 5														
Sprint 6														
Bachelorskriving											U		E	
Presentasjon av bachelor														
EXPO														

Forklaring av Gantt-diagram:	
Lys gule / beige felt =	Oppgaver opprettet fra skolen
Lys blå felt =	Oppgaver opprettet fra oppdragsgiver eller av eget initiativ
Rødt felt =	Periode uten spesielle oppgåver
Forklaring av Gantt-diagram:	U = Utkast, E = Endelig innlevering

3.4.2.2. Utdypning av Sprinter

Prosjektgruppen har valgt å benytte "sprinter" som er store oppgaver i blokker som forventes å være ferdig innenfor de gitte tidsperiodene. Forklaringen for de forskjellige sprintene vil bli gitt under.

Sprint 1
I løpet av denne perioden vil det settes mål å slå fast hvilke teknologier og verktøy som skal brukes i bacheloroppgaven. Det vil være viktig å velge riktige løsninger med tanke på kompatibilitet og ytelse.

Sprint 2
Sprint 2 vil bli brukt til å teste ut litt forskjellige teknologier, gjøre enkle tutorials for å få en dypere forståelse for verktøyene som skal anvendes. Dette gjør det lettere å støte på problemer tidlig som omhandler kompatibilitet og ved feilsøking.

Sprint 3
I denne perioden blir det å teste deres kundedata for login og de simpleste funksjonene. Dette vil da si å implementere deres kundedata(Alfresco) i backend utviklingen. Samtidig vil det utvikles frontend arkitektur, lage en layout og få implementere de mest essensielle funksjonene her. Her er altså backend og frontend sine mest essensielle funksjoner prioritert.

Sprint 4
Dersom alt går som planlagt i sprint 3 vil det i denne perioden testes at backend mot frontend fungerer som det skal og eventuelt rette opp i kompatibilitetsproblemer som kan oppstå her. Så hovedfokuset for sprint 4 er å "sy sammen" backend og frontend. Utvikling av videorom for frontend og backend.

Sprint 5
Her vil prosjektgruppen gå frem med å implementere så mangen brukstilfeller som mulig. I denne perioden vil de bli brukt nær kommunikasjon med oppdragsgiver om hvordan de vil ha løsningen rent visuelt. Det er laget brukstilfeller som skal følges, men av erfaring vil slike ting bli tilpasset i praksis.

Sprint 6
Sprint 6 vil i høy grad bli benyttet til testing og finpussing av produktet. Her skal også produktet bli overlevert til oppdragsgiver.

3.4.3. Risikohåndtering

Prosjektgruppen har valgt å bruke en risikoliste for å føre opp de største risikoene og hvordan de skal håndteres. Risikolisten inneholder hva risikoen er, sannsynlighet for at den inntreffer og hvor stor konsekvens den har.

Sannsynlighet og konsekvens har begge en skala på 1-5 der 1 er lav sannsynlighet/konsekvens og 5 er høy sannsynlighet/konsekvens. Disse to multipliseres for å få en risikofaktor mellom 1 og 25, der 1-10 er lav konsekvens, 10-20 er middels og 20-25 er kritisk konsekvens. Det er òg lagt frem forebyggende tiltak dersom risikoen er i ferd med eller allerede har inntruffet, samt hvem som er interessentene og hvilken fase av prosjektet det inngår i.

3.4.3.1. Risikoanalyse

S = Sannsynligheten for at risikoen inntreffer.

K = Dersom risikoen inntreffer, hva er konsekvensene.

RF = Risikofaktor. S multiplisert med K.

1-5 = 1 tilsvarer lav sannsynlighet/små konsekvenser og 5 tilsvarer høy sannsynlighet/store konsekvenser.

Suksessfaktorer	S	K	RF	Tiltak	Interessent	Fase
Oppgaven blir ikke løst i henhold til kravspesifikasjonene fra oppdragsgiver	2	4	8	Tett dialog med oppdragsgiver underveis i prosessen	Oppdragsgiver	Planleggings-fasen
Applikasjonen kan ikke brukes av oppdragsgiver	3	5	15	Kontinuerlig tilbakemelding på status av applikasjonen	Oppdragsgiver	Kontinuerlig
Applikasjonen er ikke brukervennlig	3	3	9	Tilbakemelding på bruken av applikasjonen fra en uavhengig tredjepart	Oppdragsgiver /kunder	Kontinuerlig
For mange brukstilfeller/for stor arbeidsmengde	4	3	12	Begrense brukstilfeller med tanke på tid og ressurser	Oppdragsgiver /gruppen	Kontinuerlig
Skjev fordeling mellom oppgaveskriving og utvikling av applikasjonen	2	4	8	Holde tidsfrister, holde møte med arbeidsfordeling. Hvis nødvendig, omstrukturere arbeidsfordelingen	Gruppen	Kontinuerlig
Mangel på kunnskap i bruk av teknologiene	3	4	12	Sett av mer tid til opplæring, søke ekstern kompetanse	Oppdragsgiver /gruppen	Kontinuerlig

Tabell 3.1: Risikoanalyse

3.5. Evalueringsmetode

Prosjektgruppen ser for seg hyppig kommunikasjon med oppdragsgiver underveis i prosjektperioden. Som nevnt tidligere vil Scrum bli brukt som utviklingsmetode, som en bonus for å bruke Scrum, vil gruppen få tilbakemeldinger fortløpende. Det er disse tilbakemeldingene gruppen ønsker å nytte som sin evalueringsmetode. Testing vil også fungere som en evalueringsmetode, der gruppen selv oppdager og korrigerer eventuelle feil eller mangler.

4. Detaljert design

I dette kapittelet beskrives designet i dypere detalj, hvordan planleggingen og utførelsen av valgt løsning ble utført. Det er delt opp i tre deler, AWS-server, backend-arkitektur og frontend-arkitektur for å gi en lettere forståelse av oppbyggingen.

4.1. AWS-server

AWS-serveren som blir brukt er av typen EC2 (Amazon Elastic Compute Cloud, u.å.). Dette er en server som er satt opp i Amazon's datasenter. Der prosjektgruppen og FieldCom kan nytte seg av serveren så lenge de har internett-tilkobling. Amazon plattformen består av et stort spekter av tjenester. Typen EC2 er en virtuell maskin som er plassert i Amazon sitt nett, der de som kjøper tjenesten får tilgang til serveren for å konfigurere den slik som ønsket.

Det man i første omgang ville få til er enkel kommunikasjon med hjelp av AWS-serveren som oppdragsgiveren har satt opp til prosjektgruppen. Serveren ble i starten satt opp med Ubuntu 18.04. OpenVidu-server's installasjonsveiledning støttet ikke Ubuntu 18.04, så det ble testet med Docker-container først. Ettersom det ikke så ut til å fungere som ønsket, ble det installert Ubuntu 16.04 på AWS-serveren slik at den ble kompatibel med OpenVidu-server sin installasjonsveiledning.

Denne serveren vil bli brukt til å simulere den serveren som tjenesten skal kjøre på ved for eksempel oljeplattformen. Det er på denne AWS-serveren produktet vil bli utviklet og testet på. Her har også FieldCom tilgang slik at de også kan se på og teste løsningen underveis. Dette gjør det enkelt å vise frem og teste ilag med FieldCom slik at de kan gi fortløpende tilbakemeldinger.

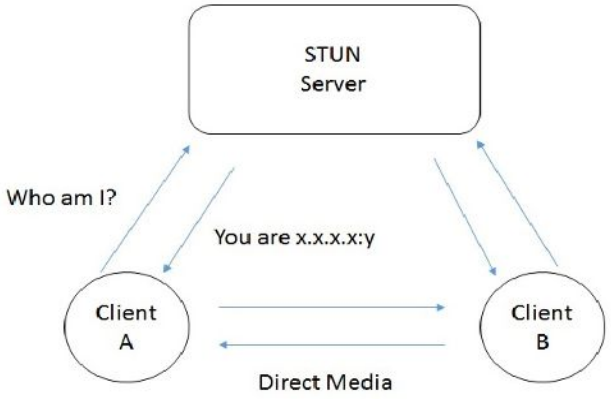
I de neste underkapitlene vil tjenester og teknologier som er brukt på AWS-serveren bli diskutert.

4.1.1. STUN/TURN

Som implementering av STUN og TURN, benyttes Coturn som har en enkel installasjonsveiledning og konfigureres med ekstern IP-adresse og porter.
For å opprette forbindelse og kommunikasjon mellom klientene blir det brukt STUN/TURN-server. STUN/TURN er en teknologi som gjør det mulig for klienter å se hverandres offentlige ip-adresser.

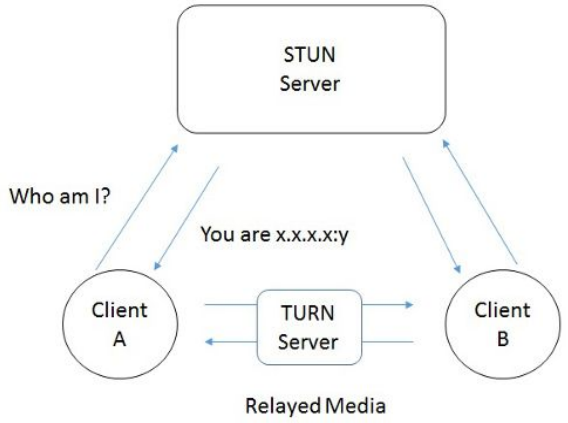
For å forklare nærmere hvordan to eller flere WebRTC klienter kommuniserer med hverandre, må ICE diskuteres (Understanding WebRTC Media Connections, 2014.). ICE er et rammeverk som gjør det mulig for klienter å droppe kompleksiteten fra verdens kompliserte nettverk-infrastruktur. Jobben til ICE omhandler å finne beste veien å gå for at partene kan kobles i lag. Det er ikke alltid det er mulig å etablere en direkte tilkobling mellom klientene, men da fungere ICE likevel selv om de står bak NAT's.

Basert på typen NAT bestemmer ICE hvilken server den skal bruke, altså enten STUN eller TURN. Ved symmetrisk NAT oversettes ikke bare IP-adressen fra privat til offentlig, men også portene. Her vil en TURN-server bli brukt. Ved symmetrisk NAT er begge sider bak NAT, kontra asymmetrisk NAT der kun den ene siden er bak NAT. Ved asymmetrisk NAT vil STUN-server bli brukt, STUN-serveren viser klientenes offentlige IP-adresser og hvilken type NAT de står bak. Som oftest blir en STUN-server brukt i starten for å sette opp tilkoblingen mellom klientene og deretter flyter trafikken direkte mellom klientene som vist på figuren under.



Figur 4.1: STUN-server trafikkflyt
Kilde: Understanding WebRTC Media Connections: ICE, STUN and TURN

Om STUN ikke klarer å etablere tilkoblingen, velger heller ICE å nytte seg av TURN. TURN er et mellomledd for klientene, der data kan lagres før den sendes videre. Ved bruk av TURN-server istedenfor STUN vil trafikken fortsatt gå igjennom serveren istedenfor en direkte link mellom klientene. Når alt dette er sagt kan det sies at det foretrekkes at STUN blir brukt, men dette er ikke alltid mulig. Alle tjenester som bruker WebRTC bør være forberedt på å ha støtte for både TURN og STUN. Under vises en figur som beskriver trafikkflyten ved bruk av TURN.



Figur 4.2 : Turn-server trafikkflyt
Kilde: Understanding WebRTC Media Connections: ICE, STUN and TURN

4.1.2. MySQL

Det ble tidlig klart at det måtte til en database for å håndtere data på serveren. Det er opprettet en database ved navn "fieldcom". Under vises navn og forklaring for de forskjellige tabellene i databasen.

- hibernate_sequence: Denne tabellen inneholder kun en verdi, "next_val" som beskriver neste verdi for enten oppretting av videorom eller bruker-innlogging. Denne verdien er en primary key og Auto_increment altså den øker gradvis når verdien blir brukt. Altså om "next_val" = 5 og en bruker logger inn vil online_message id = 5, dersom brukeren rett etter oppretter et videomøterom, vil videomøterommet få id = 6.
- online_message: Her vises brukere som er pålogget tjenesten for øyeblikket med id og brukernavn.
- user: Ved førstegangs pålogging for brukere blir de lagt til i denne tabellen, denne påloggingen må samsvare med info fra Alfresco-API'et for at påloggingen skal bli validert. For hver gang en bruker logger inn vil verdiene bli oppdatert, selv om de ikke nødvendigvis endrer seg.
- user_roles: De to bruker-rollene som er generert fra Spring er "admin" og "user". Desse blir knytt til brukere som ligger i tabellen "user" for å beskrive hvilken av desse to brukertypene brukerene er.
- video_room: Videomøterommene som blir opprettet ligger her. Id fra "hibernate_sequence" sin "next_val" blir som nevnt tidligere blir id'en til videomøterommet. Videre består tabellen av tittelen som blir valgt til rommet, samt "creator_id" som er id'en til brukeren(fra user) som oppretter rommet.
- video_room_attenders: I denne tabellen vises hvilke videomøterom deltakerne tar del i, med video_room id og attender_id.

4.1.3. Tomcat og HTTPS

Apache Tomcat er en åpen-kilde Java-web-server hvor Java-teknologier kan utrulles (Apache Tomcat, u.å.). Installeringen er enkel og man behøver kun konfigurere den med stien til Java-installeringen. Tomcat kommer med en egen nettside for administrering, denne må konfigureres med brukernavn og passord før man kan ta den i bruk. Prosjektgruppen vil bruke denne programvaren for å utrulle applikasjonen innpakket som en WAR-fil.

Kryptering er obligatorisk for alle WebRTC-komponenter, og API-ene kan bare brukes fra en sikker opprinnelse, det vil si HTTPS eller localhost (Google Codelabs, u.å.).

For å konfigurere Tomcat til å aktivere HTTPS kreves det en sikkerhetsnøkkel. Deretter må "server.xml"-filen til Tomcat konfigureres med følgende:

```
<Connector port="8443" maxHttpHeaderSize="8192" maxThreads="100"
        minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" disableUploadTimeout="true"
        acceptCount="100" scheme="https" secure="true"
        SSLEnabled="true" clientAuth="false"
        sslProtocol="TLS" keyAlias="VIDEO"
        keystoreFile="/etc/tullball/video.fieldcom.no.jks"
        keystorePass="HEMMELIG_PASSORD" />
```

4.1.4. OpenVidu og Kurento Media Server

På OpenVidu's egen nettside kan man finne installasjonsveiledninger for flere medier, inkludert Ubuntu 16.04 som AWS'en kjører. Følgende programvare må installeres og konfigureres:

4.1.4.1. Kurento Media Server

Kurento er en WebRTC-mediaserver som gjør det enkelt å utvikle avanserte videoapplikasjoner for internett og smarttelefonplattformer. Kurento Media Server-funksjoner inkluderer gruppekommunikasjon, transkoding, opptak, mixing, kringkasting og ruting av audiovisuelle strømmer (What's Kurento, u.å.). Kurento konfigureres med IP-adressen og porten til kjørende Coturn-server.

4.1.4.2. Redis

Redis er et åpent kildekode, datastruktur "in-memory"-lagringssted som blir brukt som database, cache og meldingsmegler (Redis, u.å.). Coturn og OpenVidu benytter Redis til å opprette, lagre og slette brukerlegitimasjoner ved hver sesjonstilkobling (Medium, 2018).

4.1.4.3. OpenVidu-server

OpenVidu-server krever at Java 8 er installert ettersom programvaren er en Java JAR-kjørbar fil, som kan lastes ned fra OpenVidu's GitHub-side.

For å tilby sikkerhetssertifikater i applikasjonen må det opprettes egen nøkkel i form av en ".jks"-fil. Dette er samme fil som ble benyttet til å aktivere HTTPS i Tomcat.

Etter at Coturn, Redis og Kurento er installert og konfigurert, er OpenVidu-serveren klar til start. Konfigureringen av OpenVidu-server gjøres i samme kommandolinje ved oppstart.

```
nohup java -jar -Dopenvidu.secret=MY_SECRET
-Dopenvidu.publicurl=https://video.fieldcom.no:4443/
-Dserver.ssl.key-store=/etc/tullball/video.fieldcom.no.jks
-Dserver.ssl.key-store-password=HEMMELIG_PASSORD
-Dserver.ssl.key-alias=VIDEO openvidu-server-2.8.0.jar &
```

Kommando	**Beskrivelse**
nohup	Ignorerer hangup signal slik at serveren fortsetter å kjøre selv når kommandovinduet stenges
java -jar	Spesifiserer filtypen
-Dopenvidu.secret	Passordet som ønskes for serveren
-Dopenvidu.publicurl	Maskinens offentlige IP-adresse
-Dserver.ssl.key-store	Stien til Java sikkerhetsnøkkel
-Dserver.ssl.key-store-password	Passordet til sikkerhetsnøkkelen
-Dserver.ssl.key-alias	Aliaset til sikkerhetsnøkkelen
openvidu-server-2.8.0.jar	Stien til OpenVidu-server JAR-fil
&	Returnerer kontrollen umiddelbart til deg og tillater kommandoen å fullføre i bakgrunnen

Tabell 4.1: Beskrivelse av kommandoer for start av OpenVidu-server

4.1.5. Porter

For at utrullingen av programvarestrukturen skal fungere fra AWS mot en bruker, må det åpnes porter i brannmuren. Portene listet under er standardsatte, og kan endres etter ønske.

Port	Programvare
4443 TCP	OpenVidu-server
3478 TCP UDP	Coturn
49152-65535 UDP	WebRTC utveksler tilfeldig medier gjennom de
8080 TCP	Tomcat Webserver
8443 443 TCP	Tomcat Webserver HTTPS

Tabell 4.2: Beskrivelse av porter brukt på AWS

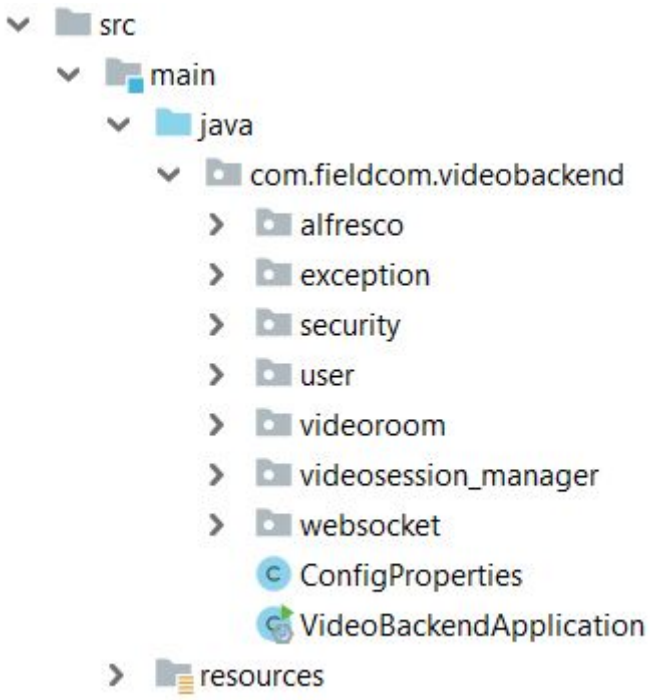
4.2. Backend-arkitektur

4.2.1. Spring

Spring-rammeverket er en åpen kildekode-applikasjon for Java plattformen. Spring er et alternativ til Enterprise JavaBean modellen, som gruppemedlemene har benyttet i kurset DAT104 - Systemutvikling og webapplikasjoner ved HVL. Prosjektgruppen har valgt å bruke Spring-rammeverket til utvikling av backend-modulen i applikasjonen. Dette rammeverket ble valgt fordi det var enkelt å sette seg inn i, mtp grunnlaget prosjektdeltakerne har fra nevnte kurs. Det var også en del av kravspesifikasjonen fra FieldCom.

Spring Boot er en samling av rammeverk, satt sammen for å gjøre det enkelt å sette opp en Spring-webapplikasjon med minst mulig konfigurasjon. Man kan med noen få linjer kode ha en kjørende REST-applikasjon og tilpasse den for ulike behov.

I dette prosjektet har det blitt benyttet Spring Boot til å lage REST-baserte API-er på backend. Disse konsumeres av frontend applikasjonen, laget i React (4.3). Backend-modulen konsumerer også Alfresco sitt REST-API for autentisering ved innlogging til å hente brukere (4.2.2). Spring-Boot har i tillegg rammeverk som støtter håndtering av databaser. Dette har blitt brukt i applikasjonen(Spring Boot – mindre kode, mere applikation, 2017).



Figur 4.3: Mappestrukturen for Backend-applikasjonen

4.2.2. REST API

API (Application Programming Interface) tillater to systemer å kommunisere med hverandre. Et API er i hovedsak språket og kontrakten for hvordan to systemer samhandler. Hvert API har dokumentasjon og spesifikasjoner som bestemmer hvordan informasjon kan overføres.

REST (Representational State Transfer) er en arkitektonisk stil som definerer en rekke begrensninger basert på HTTP. Hvis et API tilfredsstiller disse begrensningene, gir den interoperabilitet mellom datasystemer på internett og kan derfor betegnes som RESTful.

APIer som kategoriseres som REST brukes for å få tilgang til webtjenester. REST bruker webadresser for å motta eller sende informasjon, og benytter fire ulike HTTP 1.1-verb (GET, POST, PUT og DELETE) for å utføre oppgaver. I denne applikasjonen utnyttes alle disse fire metodene GET, POST, PUT og DELETE i HTTP-forespørslene. Dette betyr at web-serveren kan hente, endre, slette eller oppdatere informasjon i databasen. Endepunktene for de ulike api-ene etterlikner databasen. REST-API'et returnerer utdataene fra databasen til klienten på JSON-format (API Endpoints, u.å.).

Annotasjoner:

- @RestController: Brukes til å forenkle etableringen av en RESTful web-tjeneste. @RestController kombinerer @Controller og @ResponseBody. Dermed slipper man å anotere hver metode i kontroller klassen med @ResponseBody.

- @RequestMapping: Brukes til å definere Request URI for å få tilgang til REST endepunkter. Man definerer selv hvilken Request-metode man ønsker. Standard er GET.

- @RequestBody: Brukes til å definere innholdet i request-body.

(Tutoialspoint, u.å.)

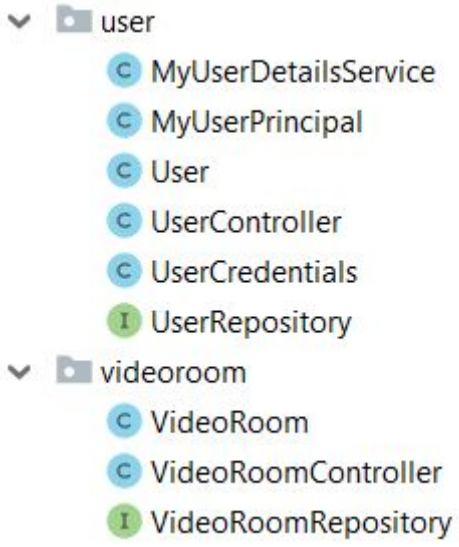
```
@RestController
@RequestMapping("/api/video-rooms")
public class VideoRoomController {

   @Autowired
   private VideoRoomRepository videoRoomRepository;
   @Autowired
   private UserRepository userRepository;

   @Secured({"ROLE_USER","ROLE_ADMIN"})
   @RequestMapping(value = "/new", method = RequestMethod.POST)
   public ResponseEntity<VideoRoom> newVideoRoom(@RequestBody VideoRoom videoRoom){

       if(videoRoom == null || videoRoom.getTitle() == null){
           return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
       }

       String loggedUserName = SecurityContextHolder.getContext().getAuthentication().getName();
       User loggedUser = userRepository.findByUsername(loggedUserName);

       videoRoom.setCreator(loggedUser);

       videoRoomRepository.save(videoRoom);
       videoRoomRepository.flush();

       videoRoom = videoRoomRepository.findById(videoRoom.getId()).get();
       return new ResponseEntity<>(videoRoom, HttpStatus.CREATED);
   }
```

Kodeutsnitt 4.1: Eksempel på bruk av av annotasjoner i VideoRoomController

4.2.3. Databasen

Spring Boot gir en veldig god støtte for å lage en DataSource for Database. Man trenger ikke skrive noen ekstra kode for å lage en DataSource i Spring Boot. Bare å legge til avhengighetene og gjøre konfigurasjons detaljer er nok til å opprette en DataSource og koble til databasen.

I modellen for denne applikasjon finnes det to entitetsklasser, User og VideoRoom, som representerer to tabeller med tilsvarende navn i databasen. Disse er implementert på webserveren med en kontroller-klasse, en modell-klasse og et repository-interface for å endre/hente ønsket informasjon fra databasen og levere til klienten. User har i tillegg noen ekstra klasser, fig. 4.8, som benyttes i forbindelse med autentisering og jwt (se avsnitt 4.2.4)



Figur 4.4: Mappestruktur for user og videoroom

Klassene User og VideoRoom er entiteter i modellen som representerer henholdsvis brukerne og Video-rommene i applikasjonen. UserRepository og VideoRoomRepository er ansvarlig for å håndtere persistering av dataene for brukere og video-rom. Disse interfacene er en utvidelse av JpaRepository-som gir tilgang til mange nyttige metoder som f.eks save(), findById() osv. Det er også mulig å definere egne metoder. Figuren under viser blant annet en metode som heter findByUsername i grensesnittet. Denne metoden blir brukt i forbindelse med autentisering(4.2.4).

```
//Annotate a Repository with @RepositoryRestResource to customize export mapping and rels.
//Flag indicating whether this resource is exported at all. Returns: true if the resource is to be
exported, false otherwise.
@RepositoryRestResource(exported = false)
public interface UserRepository extends JpaRepository<User, Long> {

  public User findByUsername(String username);

  public Collection<User> findAllProjectedBy();

  public Collection<User> findByVideoRooms(Collection<VideoRoom> videoRooms);

}
```

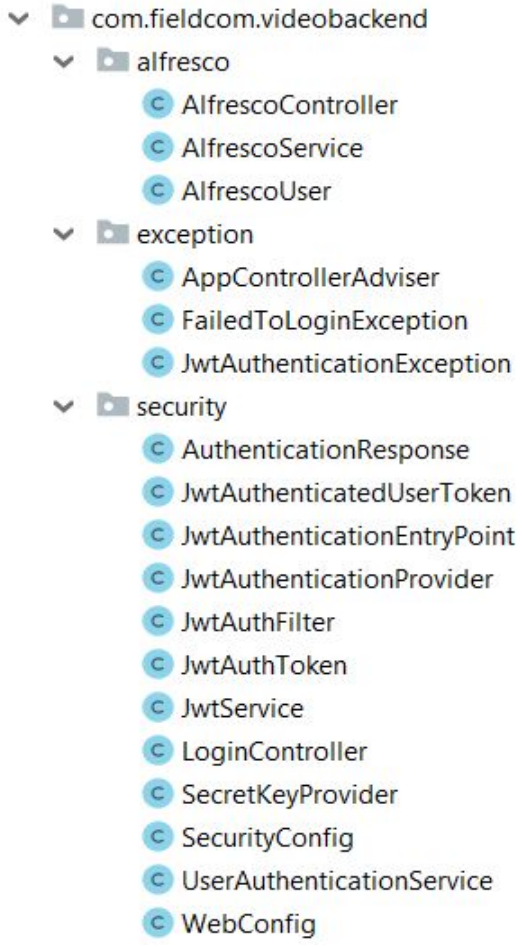
Kodeutsnitt 4.2: Eksempel på bruk av JpaRepository

4.2.4. Autorisering og JWT

JWT (JSON Web Token) er en standardiseringen for bruk av tokens til autentisering for REST-tjenester. En JWT er robust og kan bære mye informasjon, men er fortsatt enkel å bruke, selv om størrelsen er relativt liten. Som alle andre tokener kan JWT brukes til å sende identiteten til godkjente brukere mellom en klient og en tjener. Det kan også bære alle brukerens krav, for eksempel autorisasjonsdata, slik at tjeneren ikke trenger å gå inn i databasen eller eksterne systemer for å verifisere brukerroller og tillatelser for hver forespørsel. Disse dataene hentes fra JWT (Dejan Milosevic, u.å.).

Figur 4.5: Oversikt innlogging med JWT

I applikasjonen er det implementert JWT til håndtering av login, autentisering og tilgangskontroll. Figuren under viser alle filene som benyttes i denne prosessen.



Figur 4.6: Mappestruktur for JWT

Beskrivelse av programflyten ved innlogging:

- Bruker logger inn ved å sende brukernavn og passord i request bodyen til LoginController: POST /api/login
- Brukerdata videresendes til "UserAuthenticationService.java". Her blir det foretatt en test på om brukeren eksisterer i databasen. Hvis brukeren ikke har logget inn i applikasjonen tidligere må brukeren hentes fra Alfresco og legges inn i databasen som tilhører applikasjonen. Filene i Alfresco-mappen håndterer dette ved hjelp av et REST-API.
- Når brukeren er blitt autentisert benyttes klassen JwtService til å generere en token: jwtService.generateToken(username); Klassen "SecretKeyProvider.java" blir brukt i denne prosessen. Når token er generert returneres den til LoginController.
- LoginController oppretter et nytt AuthenticationResponse objekt som inneholde brukernavn og JWT. Dette objektet blir returnert til klienten.
- Frontend lagrer JWT'en og sender den til backend i en autorisasjons header ved alle nye forespørsler.
- For hver nye forespørsel tar tjeneren JWT fra autorisasjons header og dekrypterer den, validerer signaturen, og hvis alt er greit, trekker brukerdata og tillatelsene ut. Basert på disse dataene alene, uten å se nærmere på data i databasen, kan den akseptere eller nekte forespørselen.

4.2.5. WebSocket og brukerstatus

For å kunne sende beskjeder frem og tilbake mellom en nettleser og server kan WebSockets benyttes. WebSockets gir en vedvarende forbindelse mellom en klient og en server som begge parter kan bruke til å begynne å sende data når som helst. Klienten etablerer en WebSocket-tilkobling gjennom en prosess kjent som WebSocket-håndtrykk. Denne prosessen starter med at klienten sender en vanlig HTTP-forespørsel til serveren (An Introduction to WebSockets, 2013). I Spring Boot rammeverket er det innebygget WebSocket-support, for å aktivere dette må man spesifisere dette under "dependencies" i "build.gradle"-filen for prosjektet.

I applikasjonen skal dette konseptet benyttes for å oppdatere status på om en bruker er pålogget eller ikke. For å oppnå dette opprettes det en STOMP-meldingstjeneste som godtar beskjeder i form av et JSON-objekt. STOMP lar klienter og server kommunisere med tekstmeldinger over en WebSocket-protokoll, og tilbyr flere kommandoer som f.eks. connect, subscribe, send og message. Modellen for denne beskjeden er en enkel Java entitetsklasse som inneholder brukernavnet.

```java
@Entity
public class OnlineMessage  {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    private String username;
```

Kodeutsnitt 4.3: Entitetsklassen OnlineMessage.java for statusoppdatering

Endepunktet for kommunikasjonen som mottar beskjeden fra frontend når en bruker logger av eller på, er av en kontroller-klasse. Der henter den ut brukernavnet som ble tilsendt og legger inn eller fjerner brukeren fra databasen. Som svar returnerer den listen over alle påloggede brukere i applikasjonen til alle som er tilkoblet WebSocket-kanalen.

```java
@Controller
public class OnlineController {

   @Autowired
   private OnlineRepository onlineRepository;

   @Autowired
   private SimpMessageSendingOperations messagingTemplate;

   // Add username in websocket session and send online list to all online users
   @MessageMapping("/fieldcom.addUser")
   @SendTo("/topic/public")
   public void addUser(@Payload OnlineMessage onlineMessage, SimpMessageHeaderAccessor headerAccessor) {
       headerAccessor.getSessionAttributes().put("username", onlineMessage.getUsername());
       OnlineMessage online = new OnlineMessage(onlineMessage.getUsername());
       onlineRepository.save(online);
       List<OnlineMessage> allOnline = onlineRepository.findAll();
       messagingTemplate.convertAndSend("/topic/public", allOnline);
   }
}
```

Kodeutsnitt 4.4: Kontroller-klasse som håndtere online brukere

Etter at disse komponentene er opprettet må man konfigurere Spring for å aktivere WebSockets og STOMP-beskjeder. For å gjøre dette opprettes en Java-klasse kalt "WebSocketConfiguration.java" hvor konfigurering for oppsettet spesifiseres, som f.eks destinasjonen og benevninger som frontend behøver for å kunne koble til kanalen.

Det er opprettet en komponent som lytter etter "disconnect"-begivenheter, f.eks. lukking av nettleser. Når en pålogget bruker logger av oppdager denne komponenten dette og finner fram brukernavnet til tilkoblingen. Dette brukernavnet blir så fjernet fra databasen og ny liste over alle påloggede brukere distribueres ut til alle abonnenter på WebSocket-kanalen.

4.2.6. Videorom og video-sesjon med OpenVidu

Etter innlogging har backend-modulen ansvaret for å holde orden på alle video rommene som opprettes, endres eller slettes. Som beskrevet i avsnitt 4.2.3 brukes det en database for lagring av videorom og brukere, og benytter en kontroller (VideoRoomController.java) til å håndterer de ulike forespørslene fra frontend. Stien for å nå kontrolleren er "/api/video-rooms". Tabellen under beskriver de forskjellige metodene.

Sti	**Beskrivelse**
"/new"	POST Parameter: videorom objekt i Request body Lagrer videorommet i databasen Setter innlogget bruker til eier av rommet Returnerer videorommet på JSON-format
"/all"	GET Henter alle videorom fra databasen Returnerer en samling av videorom på JSON-format
"/room/{videoRoomId}"	GET Parameter: Videorom id i urlsti Henter videorommet fra database Returnerer videorommet på JSON-format
"/delete/{videoRoomId}"	DELETE Parameter: Videorom id i urlsti Henter videorommet fra databasen Bare eier av rommet kan slette Fjerner rommet fra alle brukerne Sletter rommet fra databasen Returnerer videorommet på JSON-form

Tabell 4.3: Beskrivelse av kontrollere i Spring backend

Applikasjonen har en kontroller for å opprette video sesjoner ved hjelp av OpenVidu sitt Java API. Denne heter "VideoSessionController.Java" og har metoder for å opprette video-sesjon, genere token og legge til bruker i sesjon, og fjerne bruker fra sesjon. Kontrolleren inneholder to hashmapper. Den ene for å holde orden på hvilke videorom og hvilke OpenVidu-sesjon-objekt som hører sammen. Den andre for å holde orden på hvilke video-sesjoner og hvilke tokens som hører sammen.

```
// Collection to pair video room id and OpenVidu Session objects
private Map<Long, Session> videoRoomIdSession = new ConcurrentHashMap<>();
```

```
// Collection to pair session id and tokens (the inner Map pairs tokens and users)
private Map<String, Map<Long, String>> videoSessionIdUserIdToken = new ConcurrentHashMap<>();
```

Kodeutsnitt 4.5

En sesjon kan ha mange brukere samtidig og hver bruker får generert en egen token.

En OpenVidu-sesjon opprettes i følgende metode:

```
@Secured({"ROLE_USER","ROLE_ADMIN"})
@RequestMapping(value = "/create-videosession", method = RequestMethod.POST)
public ResponseEntity<JSONObject> createVideoSession(@RequestBody String videoRoomId
```

Kodeutsnitt 4.6

Sesjonen opprettes vha. OpenVidu API'et

```
Session session = this.openVidu.createSession();
```

Kodeutsnitt 4.7:

En token blir generert når en bruker joiner et rom. Dette skjer i følgende metode:

```
@Secured({"ROLE_USER","ROLE_ADMIN"})
@RequestMapping(value = "/generate-token", method = RequestMethod.POST)
public ResponseEntity<JSONObject> generateToken(@RequestBody String videoRoomId)
```

Kodeutsnitt 4.8

Koden under oppretter en token og returnerer den til frontend:

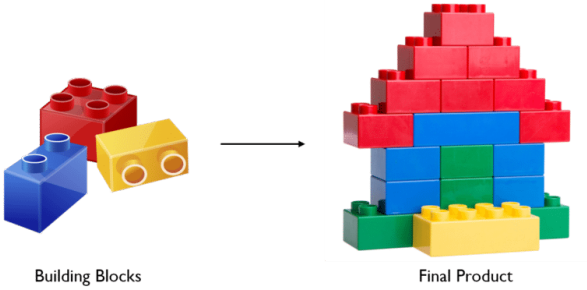
```
Session session = this.videoRoomIdSession.get(id_videoRoom);
OpenViduRole role = OpenViduRole.PUBLISHER;

JSONObject responseJson = new JSONObject();
String serverData = "{\"serverData\": \"" + loggedUserName + "\"}";
TokenOptions tokenOptions = new TokenOptions.Builder().role(role).data(serverData).build();

try{
   String token = this.videoRoomIdSession.get(id_videoRoom).generateToken(tokenOptions);

   this.videoSessionIdUserIdToken.get(session.getSessionId()).put(loggedUser.getId(), token);
   responseJson.put(0, token);

   v.getAttenders().add(loggedUser);

   videoRoomRepository.save(v);
   videoRoomRepository.flush();

   showMap();

   return new ResponseEntity<>(responseJson, HttpStatus.OK);
}catch (OpenViduJavaClientException e1){
```

Kodeutsnitt 4.9

Når en bruker joiner en video-sesjon blir vedkommende lagt til i databasen. Oppdatering av tabellen videoRom skjer derfor i denne metoden for å redusere antall requester fra frontend til backend.

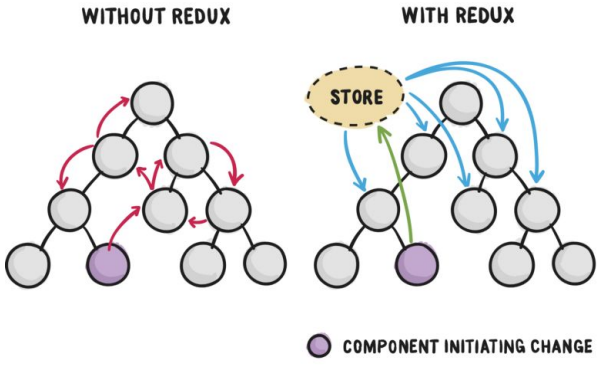
4.3. Frontend-arkitektur

ReactJS er et JavaScript-bibliotek som gjør det mulig å dele opp applikasjonen i flere komponenter for å produsere interaktive grensesnitt. I ReactJS er alt en komponent. Tenk på ett legohus som en hel applikasjon. Sammenlign deretter hver lego-blokk til en komponent som fungerer som en byggestein. Disse blokkene/komponentene er integrert sammen for å bygge en større og dynamisk applikasjon (What Is React, 2019).



Figur 4.7: Beskrivelse av komponentoppbygging i ReactJS.
Bildekilde: (What Is React, 2019)

For å holde oversikt over data (state) og hvordan man håndterer alle hendelsene (action creator) benyttes Redux, som isolerer tilstandsobjektene fra komponenter. (Understanding Redux, 2018).



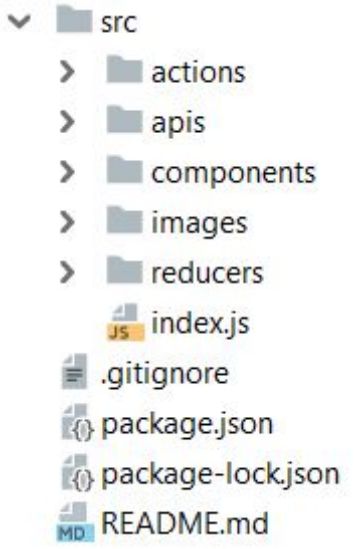
Figur 4.8: Forskjell på bruk med og uten Redux.
Bildekilde: (Understanding Redux, 2018)

Hendelsene i React sender forespørsler til Spring backend API'et og legger svarene tilgjengelig i "store" for videre bruk i applikasjonen vha. "reducers". Reducers angir hvordan programmets tilstand endres som svar på handlinger som sendes til "store" (Redux Reducers, u.å.).

Til utforming av frontend er det benyttet rammeverket Semantic UI. Dette gjør arbeidet raskere og kan også gi støtte for responsiv design for mobile enheter. For å sette et ønsket utseende til en komponent legges det til et klassenavn for den komponenten.

```
<div className="ui card">
   <div className="content">
   </div>
</div>
```

Kodeutsnitt 4.10: Eksempel på spesifisering av komponentdesign med Semantic UI



Figur 4.9: Mappestrukturen for React

4.3.1. Navigering

For å lettere implementere navigering i applikasjonen brukes HashRouter, som bruker "hashen" i nettadressen for å gjengi komponenten som skal brukes, det vil si at den definerer forbindelsen mellom en spesifikk URL og en komponent. HashRouter setter ingen begrensninger på støttede nettlesere eller webserver og server-side-ruting er uavhengig av klient-side ruting (Stack Overflow - HashRouter vs BrowserRouter, 2018). Dette medfører at man vil få en "#" i URL-adressen, men gjør arbeidet med å publisere applikasjonen i en undermappe i Tomcat mye lettere ettersom web-server ikke trenger ekstra konfigurering.

Under følger en liste over stier og hvilke komponenter de er linket til:

Sti	Komponent	Beskrivelse
"/"	Home	Forsiden til applikasjonen
"/home"	Home	Forsiden til applikasjonen
"/community"	Community	Viser liste over alle registrerte brukere
"/login"	Login	Autorisering for tilgang
"/logout"	Logout	Logger ut
"/video/delete/:id"	VideoHomeDelete	Sletter gitt videorom fra databasen
"/video/:id"	VideoSession	Henter videorom og starter en videosesjon
"/video/create/:id"	VideoHomeCreate	Oppretter et nytt viderom
"/onlineuser/:id"	OnlineUserModal	Liste til utførelser på en pålogget bruker

Tabell 4.4: Beskrivelse av URL-stier for frontend

4.3.2. Autorisering og JWT

I oppgaveteksten ble det beskrevet at FieldCom ønsker JWT-basert autorisering. I implementeringen av login-form brukes Redux-form for lettere tilgang til form-tilstanden i Redux "store". Alfresco brukerinformasjon blir sendt til backend API via "action creator" for autoriseringskontroll. Ved vellykket autorisering returneres en JWT-token som blir lagret i Redux "store" og i localStorage i nettleseren for lettere tilgang ved gjenåpning av nettsiden. Ved å lagre token i localStorage slipper man å bli logget ut ved en "refresh" av siden. Alle komponenter utover i applikasjonen blir eksportert rundt en auth-komponent som kontrollerer om det er en autorisert token i "store" ved kall og oppdatering av disse. Hvis brukeren ikke er autorisert blir man sendt tilbake til login-siden.

```
export default ChildComponent => {
   class ComposedComponent extends Component {
       // This component just got rendered
       componentDidMount() {
           this.shouldNavigateAway();
       }
       // This component just got updated
       componentDidUpdate() {
           this.shouldNavigateAway();
       }
       shouldNavigateAway() {
           if (!this.props.auth) {
               this.props.history.push('/login');
           }
       }
       render() {
           return <ChildComponent {...this.props} />;
       }
   }
   function mapStateToProps(state) {
       return { auth: state.auth.authenticated };
   }
   return connect(mapStateToProps)(ComposedComponent);
};
```

Kodeutsnitt 4.11: Kode for kontroll av autorisering i en komponent.
Kodekilde: (Advanced React and Redux Udemy Course, 2018)

4.3.3. Forespørsler til Spring Boot

For å sende forespørsler og lettere håndtere HTTP-requests fra React til backend API, benyttes en plugin kalt Axios som er en av de mest populære løftebaserte HTTP-klientene for nettlesere (How to use Axios with React, 2019). Axios er løftebasert, og dette lar oss skrive async/await kode for å utføre XHR-forespørsler veldig enkelt (HTTP requests using Axios, 2018). Moderne nettlesere har API'et "fetch", som også er løftebasert. En egenskap Axios har over "fetch" er at den utfører automatiske transformasjoner av JSON-data, mens ved bruk av "fetch" er det en to-trinns prosess når du håndterer JSON-data. Den første er å lage den faktiske forespørselen, og den andre er å kalle metoden ".json()" på svaret (Fetch vs. Axios.js for making http requests, 2017).

```
export default axios.create({
    baseURL: 'http://localhost:8080/api'
});
```

Kodeutsnitt 4.12: Oppretter et axios-objekt til backend kjørende lokalt

```
export const videoFetchAll = () => async dispatch => {
    const response = await fieldcom.get('/video-rooms/all', tokenHeader());
    dispatch({ type: VIDEO_ALL, payload: response.data })
}
```

Kodeutsnitt 4.13: Axios-forespørsel for å hente alle videorom

Ved alle forespørsler til backend (bortsett fra login), sendes det med en header med JWT-token som ble returnert ved å logge inn.

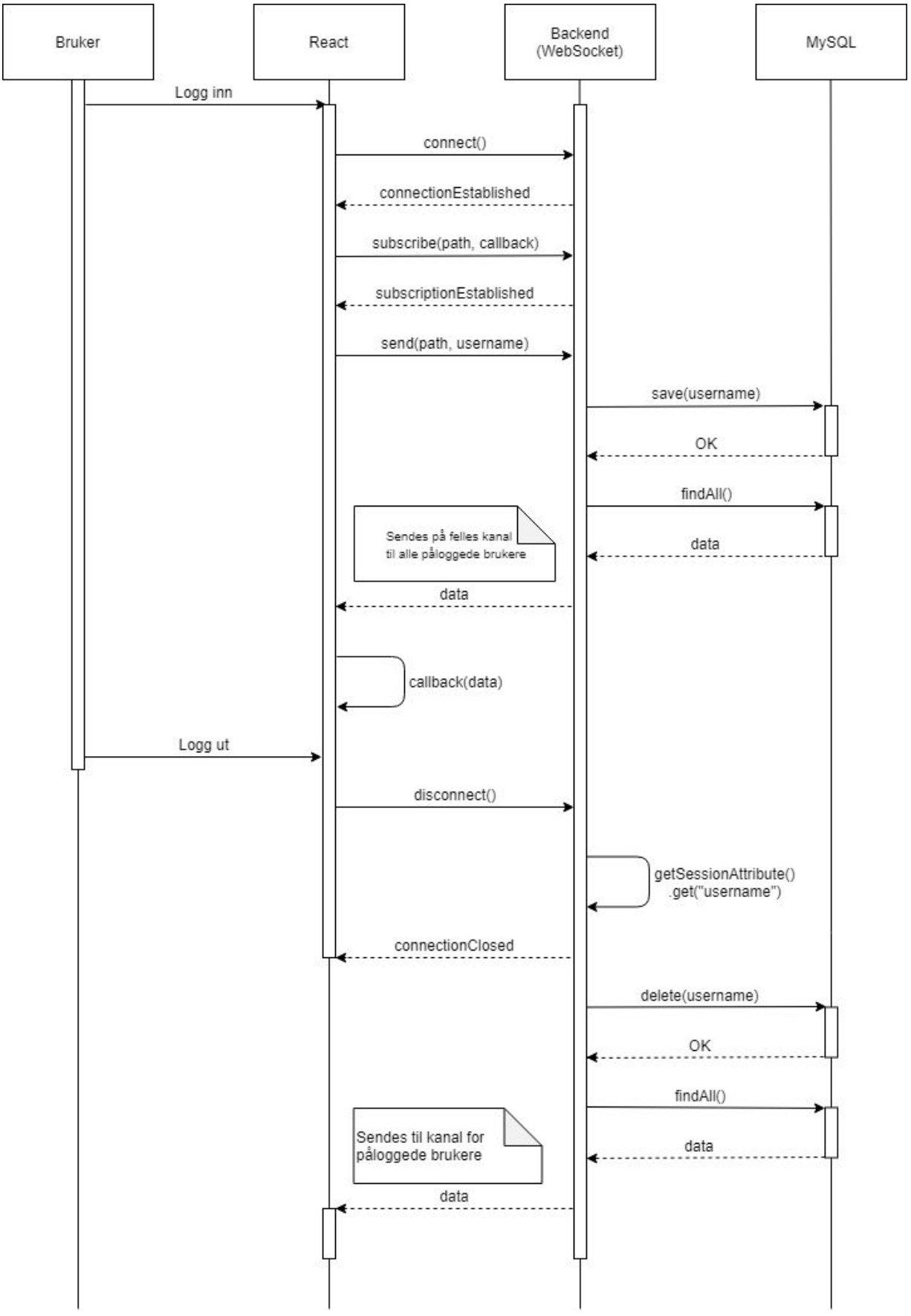
```
const tokenHeader = () => {
    const token = localStorage.getItem('fieldcomToken');
    return {
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer ' + token,
        }
    };
};
```

Kodeutsnitt 4.14: Oppsett av header med JWT for hver API-forespørsel

4.3.4. WebSocket og brukerstatus

Når en bruker logger inn i applikasjonen skal det vises en oversikt over brukere og deres status. For å holde kontroll over hvem som er pålogget, brukes WebSocket med Stomp-meldingstjener som beskrevet i 4.2.5. Ved pålogging blir autoriseringstilstanden i applikasjonen satt til autorisert. Når dette skjer sendes en WebSocket-"handshake" til backend, samtidig som menyen og forsiden blir aktivert for nettsiden. WebSocket-forbindelsen holder seg aktiv så lenge brukeren er innlogget og har nettleseren åpen. Hvis brukeren lukker nettleseren er det en "lifecycle" funksjon kalt componentWillUnmount() som lukker forbindelsen.

Hvis backend godkjenner handshaken abonnerer klienten til en kanal for kommunikasjon av formidling av informasjon. Samtidig sender klienten sitt eget brukernavn til backend via denne kanalen for å bli lagt til i listen over påloggede brukere. Ved hendelser som at en bruker logger av eller på mottas en melding på denne kanalen med brukere som er pålogget, disse blir lagret i "store" for bruk til visning på forsiden.



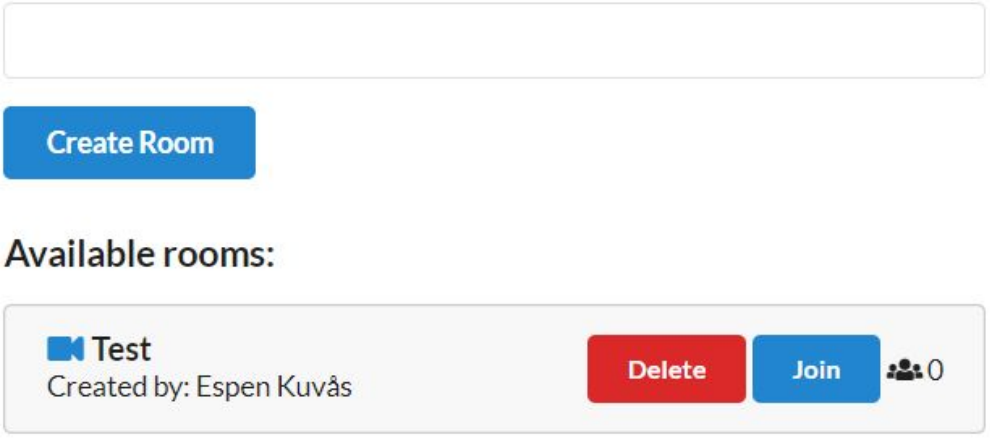
Figur 4.10: Sekvensdiagram for pålogging og avlogging for statusoppdateringer

Figur 4.11: Eksempel på visning av hele listen og enkelte brukere.

Hovedkomponenten OnlineUsers har ansvar for visningen av status for brukere, og henter alle registrerte brukere fra backend som beskrevet i 4.3.3. og lagrer i "store". Når brukere og listen over status er hentet blir disse sendt videre til komponenten OnlineUsersList.js som oppretter visningen av statusoversikten. I OnlineUsersList bestemmes rekkefølgen ved å iterere over påloggede brukere-listen først, som sender brukerinformasjonen videre til OnlineUserItem som tar seg av selve visningen av enkelte brukere. OnlineUsersList itererer så over alle registrerte brukere og sender de brukerne som ikke er pålogget videre til OnlineUsersItem for å visning av brukere som ikke er pålogget.

4.3.5. Videorom og videosesjon med OpenVidu

Etter man har logget inn og kommet til forsiden av applikasjonen, vil man på venstre side se en "form" for opprettelse av videorom. Ved opprettelse av rom sendes "form"-verdiene til Spring backend, for så å bli presentert et modalvindu hvor man kan velge mellom å gå inn i rommet og starte en videosesjon, eller gå tilbake til forsiden.
Under "formen" for opprettelse av rom listes alle rom som er opprettet med informasjon som tittel, oppretter og antall brukere som er aktive i det rommet. Det er kun oppretter som har mulighet til å slette rom, mens alle har tilgang til å gå inn i et rom.



Figur 4.12: Eksempel på visning av "form" og tilgjengelig rom.

Når man trykker på "Join" vil man bli sendt til en sti "/video/roomId". Routingen spesifisert i App.js navigerer brukeren til riktig sti med tilhørende komponent.

```
<Route path="/video/:id" exact component={ VideoSession } />
```

Kodeutsnitt 4.15: Route i App.js

VideoSession-komponenten tar seg av håndteringen av brukere, sesjon, token, og tilhørende lyd og bilde. For å oppnå dette importeres OpenVidu fra "openvidu-browser" inn i komponenten, som gir oss tilgang til en sesjon med muligheten til å starte, sende og motta video med tilhørende data. Innstillinger som f.eks. sesjon-id, sesjonsobjektet, brukernavn og mainStreamManager (hovedvideoen som vises på siden) blir lagret i tilstanden til komponenten slik at komponenten kjører en "render" på komponenten når en verdi endres, som f.eks ved at en annen bruker kommer inn.

Når VideoSession-komponenten starter henter den id'en gitt i stien, og forsøker å hente videorommet fra backend for så å sette id'en som sessionId i tilstanden. Hvis dette er vellykket kjøres funksjonen joinSession(). Der opprettes det et OpenVidu-objekt, starter sesjonen og abonnerer på begivenheter som det ønskes å få informasjon om, som f.eks. "streamCreated" som mottar streams fra sesjonen.

For å kunne koble til en sesjon trengs en OpenVidu-brukertoken, denne blir opprettet ved forespørsel til Spring backend som returnerer en Promise med token. Token er formatert som en WebSocket-link som brukes for å koble til OpenVidu-serveren.

```
wss://video.fieldcom.no:4443?sessionId=8ag05smtaavtmao1&token=29hbx0yt
2wlaaypv&role=PUBLISHER&turnUsername=ZYZROR&turnCredential=xxje7e
```

Figur 4.13: Eksempel på OpenVidu-token

Med OpenVidu-tokenen kobles det til sesjonen sammen med navnet som vises i videovinduet som vist på figur 4.15. I kildekoden kalles eget videostream-objekt en "publisher" og spesifikasjoner for objektet spesifiseres som standard i kildekoden av utvikleren. Her kan man f.eks. endre oppløsning og bildefrekvens. Et forbedringspotensiale er å la brukeren endre noen av disse spesifikasjonene selv.

```
let publisher = this.OV.initPublisher(undefined, {
   audioSource: undefined, // The source of audio. If undefined default microphone
   videoSource: undefined, // The source of video. If undefined default webcam
   publishAudio: true, // Whether you want to start publishing with your audio unmuted or not
   publishVideo: true, // Whether you want to start publishing with your video enabled or not
   resolution: '640x480', // The resolution of your video
   frameRate: 30, // The frame rate of your video
   insertMode: 'APPEND', // How the video is inserted in the target element 'video-container'
   mirror: false, // Whether to mirror your local video or not
});
```

Kodeutsnitt 4.16: Visning av et publisher-objekt

Hvis oppkoblingen er vellykket sendes eget publisher-objekt til sesjonen som er koblet opp mot OpenVidu-serveren, og setter i tilstanden til VideoSession-komponenten som mainStreamManager og publisher. Andre brukere tilkoblet sesjonen vil motta dette stream-objektet og kjøre en "streamCreated" begivenhet som oppdaterer tilstanden med den nye streamen og vises på skjermen. Tabellen under viser begivenhetene som benyttes ved en videosession.

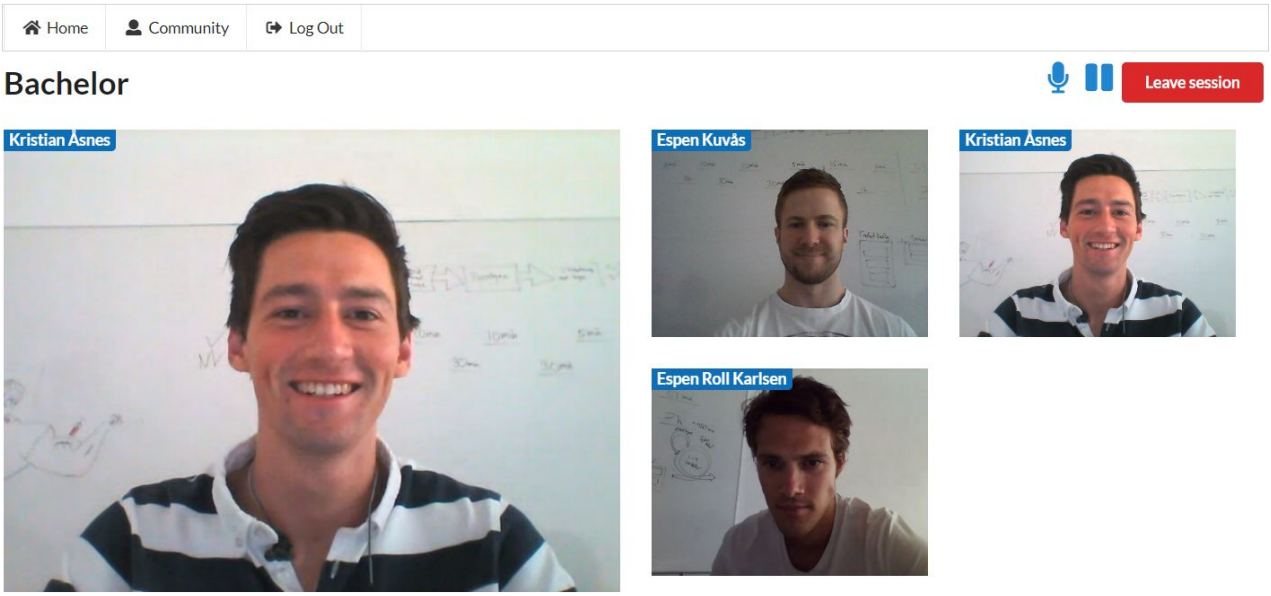
Begivenhet	**Beskrivelse**
streamCreated	For hver ny "stream" mottatt fra sesjonsobjektet, abonneres det på streamen og legger det til i tilstanden for å vise video fra brukere
streamDestroyed	Fjerner bruker fra tilstanden når en bruker kobler fra sesjonen

Tabell 4.5: Begivenheter i en videosesjon

Hver enkel videovisning skjer ved hjelp av "UserVideoComponent.js" og "OpenViduVideoComponent.js". UserVideoComponent tar seg av oppsettet for visning av navnet på brukeren og håndtere hvis en bruker klikker på videovinduet som endrer på mainStreamManager til denne brukeren slik at hver bruker selv kan velge hvilken bruker de vil ha som stor visning. Selve videovisningen blir håndtert av OpenViduVideoComponent som mottar objektet fra forelder-objektene (OpenVidu, u.å.).

Figur 4.14: Forklaring av et videoavspillingsvindu

Når en bruker forlater rommet kalles en metode som kjører en disconnect på sesjonen og setter tilstandene til komponenten tilbake til utgangspunktet.



Figur 4.15: Visning av en videosesjon.

5. Evaluering

I dette kapittelet beskrives evalueringen av prosjektet, hvilke evalueringsmetoder som er benyttet, kvaliteten og resultatet av evalueringen.

5.1. Evalueringsmetoder

5.1.1. Scrum

Utviklingsmetoden Scrum har vært et verktøy for prosjektgruppen for å kunne gi fortløpende tilbakemeldinger. Tilbakemeldingene fra hverandre har bidratt til å evaluere applikasjonen mot problemstillingen underveis, enten via møte eller intern kommunikasjon. Kommunikasjonsflyten i gruppen har pågått nesten daglig for å sikre god kvalitet over arbeidet.

Ekstern veileder har bidratt med tilbakemeldinger underveis i utviklingen av applikasjonen. Dette har skjedd via tilbakemeldinger på prosjektrapporten og på presentasjonen av prosjektet for en gruppe personer på HVL.

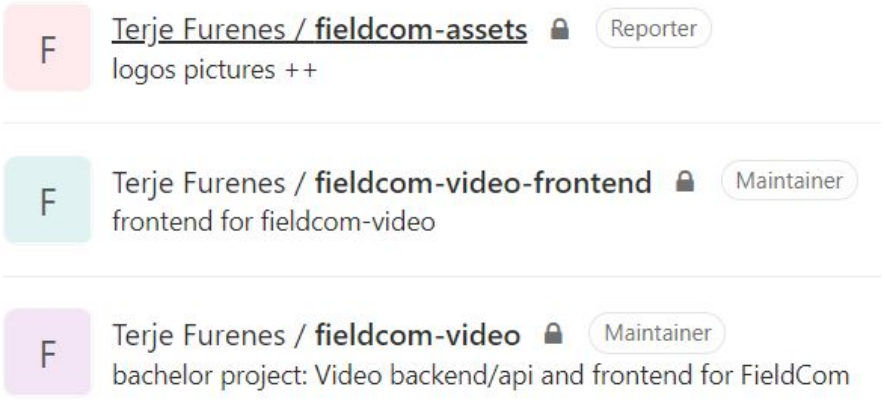
5.1.2. Evaluering ved testing

Funksjonell testing av applikasjonen har blitt utført underveis i hele prosessen, for å kontrollere og verifisere at funksjoner utfører og returnerer som forventet. Til testing av backend har det blitt brukt Postman for å verifisere at kontrollere, backend-infrastruktur og databasen oppfører seg som forventet. Postman er en gratis programvare som gjør API-utvikling raskere og lettere (Postman - Products, u.å.).

Etterhvert som applikasjonen vokste og frontend ble utviklet, ble Postman-testing supplementert med lokal kjøring av React mot Spring benyttet. På denne måten får man rask tilbakemelding og kan gjøre endringer deretter hvis nødvendig for å sikre at kvaliteten av applikasjonen tilsvarer kravene.

5.1.3. Evaluering fra oppdragsgiver

Prosjektgruppen har hatt god kommunikasjon med Terje Furenes fra FieldCom AS utover i utviklingsprosessen via Skype og appear.in. Etter det første møtet med oppdragsgiver opprettet FieldCom to GitLab repositories, en for backend og en for frontend, hvor kildekoden ble delt kontinuerlig. På denne måten hadde FieldCom kildekoden tilgjengelig til enhver tid, og kunne evaluere og gi tilbakemelding hvis nødvendig.



Figur 5.1: Visning av tilgjengelige repositories for prosjektet i GitLab

5.1.4. Evaluering ved prøving og feiling

Prosjektgruppen har brukt mye ressurser på utprøving av teknologi og eksisterende demoer fra OpenVidu. Det var avgjørende å teste de ulike teknologiene brukt i applikasjonen, da prosjektgruppen hadde lite forhåndskunnskap om disse. Måten dette ble testet var å installere nødvendige komponenter på AWS'en og rulle ut en demo applikasjon fra OpenVidu. Når dette etterhvert fungerte fikk prosjektgruppen en bekreftelse på at prosjektgruppen kunne gå videre med den denne teknologien. Dette ble gjort på et tidlig stadium og det var mye prøving og feiling i denne prosessen.

I utviklingsprosessen har det også vært mye prøving og feiling. Da spesielt med tanke på implementering av JWT og WebSocket i applikasjonen. Dette er konsepter prosjektgruppen ikke hadde noen forkunnskaper om, og implementeringen skjedde ved hjelp av artikler og eksempler nettet.

5.2. Evalueringsresultater

Evalueringsmetodene nevnt over danner grunnlaget for evaluering av resultatet av prosjektet.

Gruppen har hele tiden hatt en kjørende applikasjon på AWS-serveren som har vært tilgjengelig for alle. Dette har gitt oppdragsgiver mulighet til å teste, evaluere og komme

med innspill til løsningen. Applikasjonen har dermed blitt utviklet i tråd med ønsker fra oppdragsgiver. På denne måten har potensielle problemer blitt fanget opp og diskutert.

Ettersom kommunikasjonen med oppdragsgiver har vært såpass frekvent, vil prosjektmedlemmene si at de har utviklet et riktig sluttprodukt for formålet, kontra et nytteløst men optimalt produkt. Det som blir forsøkt å få fram her er at gruppen har hatt som hovedprioritet å lage et produkt som blir brukt i fremtiden. Altså å lage et riktig produkt for virksomheten.

Prosjektgruppen sendte en liste med spørsmål til FieldCom for evaluering av prosjektperioden og resultatet. Under følger listen med spørsmål og svar fra Terje Furenes.

1. Før prosjektet startet, hvor stor var forespørselen om et videomøterom for lokale nettverk?
Litt vanskeleg å sei, men det var eit spørsmål som ofte dukka opp i samtaler med potensielle kundar. Enkelte likar å ha veldig lukka nettverk og er redde for "skyen". Det er nok noko som vil endre seg, men det er noko som absolutt vil vere kjekt å tilby. Denne videoløysinga vil også kunne plasserast i «skyen», som gjer produktet fleksibelt og komplimenterande til andre funksjonar vi tilbyr.

2. Hvordan synes dere det har gått med prosjektarbeidet underveis?
Det har vore bra framdrift og det ser ut til å vere god kvalitet på kode og produkt.

3. Er dere fornøyd med teknologien som er benyttet for å utvikle produktet?
Ja.

4. Hvordan har dialogen mellom dere og prosjektgruppen vært?
Det har vore ok. Vi ser at vi i FieldCom kunne sett av litt meir tid til oppfølging, men det tar vi sjølvsagt på vår kappe. Prosjektgruppa har hatt kontakt på epost, skype og videomøte - det har fungert bra for vår del.

5. Vil løsningen være enkel for dere å vedlikeholde og endre i ettertid?
Det er brukt kjent teknologi, språk og rammeverk, og koden er godt dokumentert, så det håpar vi skal går bra.

6. Hva er tankene om det nåværende produktet?
Ser veldig bra ut.

7. Har prosjektgruppen oppnådd deres mål og forventning for dette prosjektet?
Ja.

6. Diskusjon

I dette kapittelet diskuteres resultater i forhold til arbeidsmetode og om noe kunne vært gjort annerledes.

Resultatet av utviklingsprosessen ble påvirket av hvordan prosjektgruppen angrep problemstillingen. Det har hele tiden vært fokus på å utvikle en fungerende applikasjon og oppfylle de kravspesifikasjoner som ble gitt. Strategien var å utvikle de viktigste brukstilfellene først, slik at man tidlig kunne ha en kjørende applikasjon på AWS'en.

Fremgangsmåten medførte at løsningen var demonstrerbar for oppdragsgiver gjennom mesteparten av prosjektperioden. Oppdragsgiver kunne derfor få hyppige oppdateringer om produktets funksjonalitet, og kunne gi konkrete tilbakemeldinger på endringer som ble gjort i løsningen. Et eksempel på dette var valget om å etablere en egen MySQL-database på AWS'en for å lagre brukere og videorom. Dette valget var basert på demoprosjekter fra OpenVidu.

Prosjektet omfatter veldig mange teknologier, noe som betyr mye informasjon på kort tid. Det var derfor avgjørende med en god struktur på arbeidet og fordele oppgaver tidlig. Dette ble gjort allerede i innledningsfasen der de ulike teknologiene ble fordelt mellom prosjektdeltakerne. Dette førte til at prosjektdeltakerne fikk hver sin ekspertise og ulike ansvarsområder. Applikasjonen ble delt opp og de ulike delene ble utviklet separat og parallelt, med fokus på sprintene i prosjektplanen, brukstillfeller og Scrum som utviklingsmetode. Sammenslåingen, utrulling og testing har også blitt gjort gradvis, etterhvert som de individuelle delene av applikasjonen ble ferdigstilt.

Prosjektgruppen føler den har oppnådd et godt resultat med tanke på prosjektets kompleksitet, tidsbegrensing og forhåndskunnskaper. Hvis prosjektet skulle blitt utført på nytt, ville det ikke vært noen endringer på valg av teknologi eller utviklingsverktøy, da prosjektgruppen har for lite kunnskap om andre alternativer. Prosjektgruppen har brukt Spring Boot til utvikling av backend og React til utvikling av frontend. På AWS'en har det blitt installert følgende komponenter: OpenVidu-server, Kurento Media Server, Coturn, Redis, MySQL database og Tomcat.

Det foreligger mye god dokumentasjon for disse komponentene som prosjektdeltakerne har støttet seg på underveis i utviklingen. Under utviklingsprosessen ville imidlertid prosjektgruppen ønsket å bruke mer tid på å designe databasen og forbedrere databasestrukturen. Slik løsningen er utformet nå, brukes Alfresco sitt API til å hente brukeren fra en database ved førstegangs innlogging, og legger den inn i databasen på AWS'en. Man må dermed gjøre en sjekk mot denne Alfresco-databasen ved alle innlogginger for å se om noen av brukerens datafelter har blitt endret.

7. Konklusjon

7.1. Måloppnåelse

Hovedmålet for prosjektet var å opprette et videomøterom for lokale nettverk, etter ønsker fra kundegruppene til oppdragsgiver. FieldCom ønsker med dette prosjektet å erstatte den eksisterende løsningen Twilio, som er avhengig av internettforbindelse til alle brukere. Prosjektgruppen har implementert hovedmålet, en grunnleggende og fungerende videomøterom-løsning med rom for utvidelse etter eget ønske. Det ble diskutert mange brukstilfeller i møte med FieldCom, hvorav de ønsket så mange som mulig. Prosjektgruppen har arbeidet ut i fra disse ønskene og har kommet i mål med mange, mens de gjenstående viser seg å være tidkrevende implementeringer, f.eks. implementasjon mot Skype og intern ringe-funksjon.

Følgende brukstilfeller eksisterer i applikasjonen:

- Logg inn via Alfresco
- Vis andre brukere
- Vis andre brukeres status (Online/Offline)
- Logg ut
- Opprette et video-rom
- Slette et video-rom
- Bli med i video-rom
- Velge “programleder” for en video-sesjon
- Slå av egen mikrofon
- Slå av egen videostrøm
- URL til videorom

FieldCom ønsket at prosjektet skulle utvikles som en egen applikasjon, slik at de selv kan implementere funksjonene inn i sin egen applikasjon senere. Ut i fra dette og evalueringen betrakter prosjektgruppen at målene for prosjektet er oppnådd, med rom for utbedringer og potensiale som diskutert under.

Applikasjonen skal kunne brukes på lokale nettverk uten internett, men ettersom Alfresco-serveren som er mottatt fra FieldCom kjører via internett, kan ikke applikasjonen testes fullverdig uten internett uten at det kan installeres med en egen Alfresco-server lokalt.

7.2. Kjente feil og mangler

- I nettleserne Firefox og Opera fungerer kun "secure websockets (wss)" med godkjente sertifikater fra en tilbyder. Ettersom applikasjonen er implementert med selvsignerte sertifikater må man derfor navigere seg inn på OpenVidu-serveren sin egen testside for å godkjenne sertifikatet før man får en fungerende video-sesjon i applikasjonen. Selv om sertifikatet som benyttes til OpenVidu og applikasjonen er den samme. I nettleseren Google Chrome er det kun nødvendig å godkjenne sertifikatet som blir presentert når man benytter seg av applikasjonen.
- JWT har foreløpig ingen ekspirasjonstid, dette kan betraktes som en sikkerhetsrisiko ettersom nøkkelen vil være godkjent i all evig tid inntil Spring Boot blir restartet.
- WebSocket-forbindelsen for oppdatering av brukerens online/offline status benytter foreløpig ikke JWT. Dette kan betraktes som en sikkerhetsrisiko ettersom man kan lage sin egen applikasjon og koble til samme server hvis man kjenner til oppbyggingen.
- Etter at det er blitt opprettet en helt ny bruker i Alfresco, eksisterer denne ikke i den lokale databasen før denne brukeren logger inn. Hvis så denne brukeren logger inn mens en annen bruker befinner seg innlogget på forsiden, oppsto det en feilmelding på listing av påloggede brukere at denne brukeren ikke eksisterer. Dette er løst ved å hente ned alle brukere på nytt ved hver beskjed om at et ny bruker er logget på. Dette er ikke optimalt etterhvert som applikasjonen vokser i størrelse og bør utbedres.

7.3. Forbedringspotensial

Det er stort sett alltid rom for forbedring i større prosjekter, og det er det også i denne applikasjonen. Kodestrukturen kan implementres med mer fokus på feilhåndtering og unntakshåndtering for å hindre uønskede hendelser og feilmeldinger. Testmetoder med enhetstesting av kontrollere for å verifisere at forventet utfall oppnås, inkludert mer kommentering generelt i kildekoden.

Databasetabellen "hibernate_sequence" tildeler neste verdi til både pålogget bruker og nytt videorom, en utbedring ville vært en teller for hver tabell, slik at videorom-id vokser i løpende rekkefølge. Både brukertabellen og videoromtabellen burde hatt en egen auto_increment.

Det er ikke spesifisert fra oppdragsgiver hva som skal skje ved duplikat innlogging, dette er foreløpig tillatt hvorav begge brukere kan bli med i en video-sesjon, disse vil da også vise samme navn i sitt videovindu.

For å hindre advarselsiden ved selvsignerte sertifikater kan man opprette et CA-signert sertifikat, eller opprette sertifikat hos "Let's Encrypt", men da må sertifikatet fornyes hver tredje måned.

7.4. Videre arbeid

Anbefaling for videre arbeid ville startet med utbedring av eksisterende feil og mangler for å ha en sikker grunnstruktur for applikasjonen før den utvides. Under følger en liste over brukstilfeller som kan være aktuelle å implementere, i vilkårlig rekkefølge.

- Funksjon for å ringe en annen bruker
- Invitere andre brukere til et videorom
- Passordbeskytte videorom
- Integrere mot Skype
- Dele rom med eksterne brukere
- Opptak og visning av opptak
- Skjermdeling
- Bedre utforming på video-sesjon med flere brukere

8. Litteratur og Referanser

Amazon Elastic Compute Cloud (u.å) [Internett] Tilgjengelig fra: https://en.wikipedia.org/wiki/Amazon_Elastic_Compute_Cloud [Lest 07.05.19]

Amazon EC2 (u.å) [Internett] Tilgjengelig fra: https://aws.amazon.com/ec2/

SimpleWebRTC (u.å.) [Internett] Tilgjenglig fra: https://www.simplewebrtc.com/ [Lest 26.03.19]

NextRTC (u.å.) [Internett] Tilgjengelig fra: https://nextrtc.org/ [Lest 26.03.19]

Jitsi (u.å.) [Internett] Tilgjengelig fra: https://jitsi.org/ [Lest 26.03.19]

OpenVidu (u.å.) [Internett] Tilgjenglig fra: https://openvidu.io/ [Lest 25.03.19]

OpenVidu (u.å.) Tutorials: openvidu-mvc-java [Internett] Tilgjengelig fra: https://openvidu.io/docs/tutorials/openvidu-mvc-java/ [Lest 07.05.19]

OpenVidu (u.å.) Tutorials: openvidu-insecure-react [Internett] Tilgjengelig fra: https://openvidu.io/docs/tutorials/openvidu-insecure-react/ [Lest 07.05.19]

Frozen Mountain (u.å.) Ultimate Guide to WebRTC [Internett]. Tilgjengelig fra: https://www.frozenmountain.com/ultimate-guide-to-webrtc [Lest 25.03.19]

PuTTY (u.å.) [Internett] Tilgjengelig fra: https://www.putty.org/ [Lest 36.03.19]

WebRTC (u.å.) Home [Internett] Tilgjengelig fra: https://webrtc.org/ [Lest 25.03.19]

Scrum (u.å.) What is Scrum? [Internett] Tilgjengelig fra: https://www.scrum.org/ [Lest 27.03.19]

What Is React (2019) [Internett] Tilgjengelig fra: https://www.edureka.co/blog/what-is-react/ [Lest 27.04.19]

Understanding Redux (2018) [Internett] Tilgjengelig fra: https://medium.com/@tkssharma/understanding-redux-react-in-easiest-way-part-1-81f3209fc0e5 [Lest 27.04.19]

Advanced React and Redux Udemy Course (2018) 2018 Edition [Internett] Tilgjengelig fra: https://www.udemy.com/react-redux-tutorial/ [Lest 28.04.19]

Spring Boot – mindre kode, mere applikation (2017) [Internett] Tilgjengelig fra: https://home.lundogbendsen.dk/spring-boot/ [Lest 1.05.19]

API Endpoints - What Are They? Why Do They Matter? (u.å) Tilgjengelig fra: https://smartbear.com/learn/performance-monitoring/api-endpoints/

Tutorialspoint - Spring Boot - Building RESTful Web Services (u.å) Tilgjengelig fra: https://www.tutorialspoint.com/spring_boot/spring_boot_building_restful_web_services.htm [Lest 28.2.19]

Stack Overflow - HashRouter vs BrowserRouter (2018) [Internett] Tilgjengelig fra: https://stackoverflow.com/a/51976069 [Lest 08.05.19]

HTTP requests using Axios (2018) [Internett] Tilgjengelig fra: https://flaviocopes.com/axios/ [Lest 08.05.19]

Postman - Products (u.å.) [Internett] Tilgjengelig fra: https://www.getpostman.com/products [Lest 08.05.19]

Dejan Milosevic - REST Security with JWT using Java and Spring Security (u.å) [Internett] Tilgjengelig fra: https://www.toptal.com/java/rest-security-with-jwt-spring-security-and-java [Lest 08.05.19]

What's Kurento (u.å.) [Internett] Tilgjengelig fra: https://www.kurento.org/whats-kurento [Lest 09.05.19]

Redis (u.å.) Introduction to Redis [Internett] Tilgjengelig fra: https://redis.io/topics/introduction [Lest 09.05.19]

Apache Tomcat (u.å.) [Internett] Tilgjengelig fra: http://tomcat.apache.org/ [Lest 09.05.19]

openvidu-java-client API (u.å) [Internett] Tilgjengelig fra: https://openvidu.io/docs/reference-docs/openvidu-java-client/

Google Codelab (u.å.) Real time communication with WebRTC [Internett] Tilgjengelig fra: https://codelabs.developers.google.com/codelabs/webrtc-web/ [Lest 09.05.19]

Medium (2018) OpenVidu 2.2.0: TURN made easy [Internett] Tilgjengelig fra: https://medium.com/@openvidu/openvidu-2-2-0-turn-made-easy-9d7e145f8905 [Lest 09.05.19]

Designrevision (2019) How to use Axios with React [Internett] Tilgjengelig fra: https://designrevision.com/react-axios/ [Lest 28.05.19]

Medium (2017) Fetch vs. Axios.js for making http requests [Internett] Tilgjengelig fra: https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5 [Lest 28.05.19]

treehouse (2013) An Introduction to WebSockets [Internett] Tilgjengelig fra: https://blog.teamtreehouse.com/an-introduction-to-websockets [Lestd 28.05.19]

Redux (u.å.) Reducers [Internett] Tilgjengelig fra: https://redux.js.org/basics/reducers [Lest 28.05.19]

Avaya (2014) Understanding WebRTC Media Connections: ICE, STUN and TURN [Internett] Tilgjengelig fra: https://www.avaya.com/blogs/archives/2014/08/understanding-webrtc-media-connections-ice-stun-and-turn.html [Lest 02.05.2019]

9. Appendix

9.1. Akronymer

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
AWS	Amazon Web Services
CA	Certificate Authority
DOM	Document Object Model
EC2	Amazon Elastic Compute Cloud
ICE	Interactive Connectivity Establishment
IDE	Integrated Development Environment
JAR	Java Archive
JWT	JSON Web Token
RTC	Real-Time Communications
SSH	Secure Shell
STUN	Session Traversal Utilities for NAT
TURN	Traversal Using Relays around NAT
U.Å.	Uten år
WebRTC	Web Real-Time Communication

9.2. Ordliste

Alfresco	Samling av programvare for informasjonsbehandling av programvare
appear.in	Nettside for enkle videokonferanser
Async	En asynkron funksjon som opererer asynkront via hendelsesløkken, ved hjelp av et implisitt løfte(Promise) om å returnere resultatet
Await	Await venter på et løfte(Promise). Den kan bare brukes i en async-funksjon.
Coturn	Free open source implementation of TURN and STUN Server
Deploye	Utplassere ressurser f.eks til en server
Dupleks	Punkt-til-punkt-system mellom to enheter som kan kommunisere med hverandre i begge retninger samtidig.
GitHub	Web-basert versjonskontrollsystem
GitLab	Web-basert versjonskontrollsystem
Modalvindu	Grafisk kontrollelement underordnet programmets hovedvindu
OpenVidu	Plattform for tilrettelegging av videosamtaler
Plugin	Programvarekomponents som legger til en ekstra egenskap i et eksisterende program
Promise	Et objekt som brukes til å håndtere asynkrone beregninger
Render	Prosess for å transformere React komponenter til DOM noder som nettleseren kan forstå og vise på skjermen.

Repository	Lagringslokasjon hvor programvare kan hentes og installeres på en datamaskin
Signaling	Sette opp, kontrollere og avslutte en kommunikasjonssesjon.
Telnet	Protokoll til utveksling av informasjon mellom en tjener og klient
Tomcat	Service for kjøring av Java-baserte webapplikasjoner
Twilio-Video	Twilio er en utviklingsplattform for kommunikasjon. Twilio-APIer brukes for å implementere stemme, video og meldingstjenester til applikasjoner.
WebSocket	Kommunikasjonsprotokoll som gir full-dupleks kommunikasjonskanaler over en enkelt TCP-tilkobling.
XHR (XMLHttpRequest)	JavaScript API for å lage AJAX-forespørsler