NTNU

Kunnskap for en bedre verden

Bacheloroppgave

IE303612 Bacheloroppgave Data

Device Control

Kandidatnumre:

10011

10035

10048

Totalt antall sider inkludert forsiden: 84

Innlevert Ålesund, 31.05.2018

NTNU

Kunnskap for en bedre verden

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. **Manglende erklæring fritar ikke studentene fra sitt ansvar**.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	**Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7.**	☒
4.	**Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert**	☒
5.	**Jeg/vi er kjent med at NTNU vil behandle alle saker hvor det foreligger mistanke om fusk.**	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider**	☒

Publiseringsavtale

Studiepoeng: 20

Veileder: Arne Styve

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven §2).
Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage med forfatter(ne)s godkjennelse.
Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	☒**ja**	☐**nei**
Er oppgaven båndlagt (konfidensiell)? (Båndleggingsavtale må fylles ut)	☐**ja**	☒**nei**
- Hvis ja: **Kan oppgaven publiseres når båndleggingsperioden er over?**	☐**ja**	☐**nei**
Er oppgaven unntatt offentlighet? (inneholder taushetsbelagt informasjon. Jfr. Offl. §13/Fvl. §13	☐**ja**	☒**nei**

Dato: 31.05.2018

Forord

Denne oppgaven er skrevet av Kristoffer Rogne, Erik Haram Nygård og Thomas Skarshaug Todal, Dataingeniør studenter ved NTNU i Ålesund.

Vi valgte denne oppgaven blant annet fordi den bygger på mange av fagene vi har hatt i løpet av studiet; Objektorientert programmering, Webteknologi, Mobile og distribuerte applikasjoner, Datakommunikasjon med nettverksprogrammering, Datamodellering og databaseapplikasjoner, Operativsystemer, Sanntids datateknikk, Systemadministrasjon. Oppgaven vil utfordre oss i de nevnte fagene ved følgende tema: Java, Spring, kommunikasjon over Ethernet, database, trådstyrt programmering og serveradministrasjon. Oppgaven var også en av de som tilbød en god del praktisk arbeid, noe alle i gruppen var enig om var ønskelig både på grunn av erfaring og personlige interesser.

Oppdragsgiver er NTNU Ålesund, Institutt for IKT og realfag.
Veileder er Arne Styve.

Vi vil gjerne takke:
Arne Styve for god veiledning gjennom prosjektet.
Kay Sindre Lorgen for assistering med VM.

Innhold

Figuroversikt

Tabelloversikt

Sammendrag

Digitalisering av hverdagen er i full sving, og mange av produktene du kan få kjøpt i dag kan kobles til internettet på en eller annen måte. Utfordringen er å få en lett tilgjengelig sentral der du kan styre det som er ditt. Disse enhetene kan bli byttet ut når det trengs så programvare må dermed bli byttet ut. Det skal være enkelt for brukere av systemet å styre disse enhetene.

Målet med dette prosjektet var å lage en nettside der en bruker kan styre forskjellige enheter på en Visualiserings lab, prosjektorer og lydanlegg. Lett å starte og å bruke programmet selv.
Nettsiden skulle være funksjonell og selvstendig, ikke være avhengig av å vite hva det er den styrer. Siden må kunne være brukbar på alle nettlesere. Hver enhet skulle være selvstyrte slik at flere kan styres samtidig.

Denne rapporten forklarer teori som er nødvendig, hvilke metoder som ble brukt og hva enheter og materialer vi brukte for å nå målet. Spring REST for abstrahering av persistering av data, Java for modulisering og separering av prosjektet, HTML, CSS og JavaScript for å gjøre nettsiden tilgjengelig for alle. Gjennom Spring Boot og Tomcat blir det mulig å motta informasjon fra bruker, slik som hvilke enheter som skal slås på, nye enheter å legge inn i databasen eller henting av mer spesialisert informasjon fra en bestemt enhet.

Resultatet av dette prosjektet er en løsning som er enkel å utvide, samtidighet i programmet slik at enheter ikke blokkerer hverandre og database modifisering fra nettsiden. Disse enhetene har egne små program som er enkle og utvikle og kan bli lagt til som en utvidelse til nettsiden, uten at det trengs å rekompilere hele prosjektet.

Terminologi

Begreper

Brukergrensesnitt / Graphical User Interface	En interaktiv grafisk fremvisning av programvare.
Bug	Et problem eller feil i kode.
Brukervennlig	Noe som tar lite tid å lære og er intuitivt å bruke.
Cohesion	Det at en klasse skal representere én entitet, eller at en metode skal ha én oppgave.
Classpath	Området der klasser og filer blir lagt i Java Virtual Machine(JVM) for bruk.
Container	Et begrep fra Docker. En container kan sammenlignes med en virtuell maskin, men uten et eget operativsystem.
Coupling	Hvor mye en klasse trenger å vite om en annen klasse for å fungere.
Debugge	Identifisere grunn til feil og fikse/dokumentere dem.
Open Source	Åpen kildekode, programvare der opphavsrettsholderen har gitt løyve til å lese, distribuere, modifisere og publisere kildekoden til programvaren.
Persistent Data	Informasjon som aksesseres sjelden, og som har lav sannsynlighet for å endres.
Plugin	Utvidelse av program, ikke del av hoved programmet orginalt.
Representational state transfer	En metode å utvikle nettjenester som tillater interoperabilitet mellom datamaskin og nett.
Responsiv Design	Brukergrensesnittet skal være mulig å bruke på alle slags skjermstørrelser uten at det går på bekostning av utseende/stil.
Scrum	En arbeidsmetode der teamet er selvdrevet og kan definere egne mål for prosjektet. Målene blir nådd gjennom en kort iterativ prosess (sprint).
State	En tilstand programmet kan være i.
Universell Utforming	Programvare som følger en viss standard for brukervennlighet.

User Story	Brukerhistorie, kort beskrivelse på en enkelt funksjon/handling en bruker har lyst å kunne ta nytte av/gjøre i programmet.

Biblioteker og rammeverk

Spring	Et rammeverk som forenkler oppgaven å sette opp REST.
Bootstrap	Et rammeverk som forenkler oppgaven med å implementere responsiv design.

Forkortelser

CSS	Cascading style sheet
CSRF	Cross-Site Reference Forgery
DB	Database
DOM	Document Object Model
GUI	Graphical user interface / Brukergrensesnitt
HTML	Hypertext markup language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure as a Service / Infrastruktur som en tjeneste
IDE	Integrated Development Environment
JS	JavaScript
JSF	JavaServer Faces
JVM	Java Virtual Machine
MCR	Maven Central Repository
OS	Operativsystem
REST	Representational state transfer
RD	Responsiv design
SDK	Software Development Kit

SPI	Service Provider Interface
SQL	Structured Query Language
TCP/IP	Transport Control Protocol / Internet Protocol
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UU	Universell utforming
WAF	Web Application Framework

Verktøy

Java	Et objektorientert programmeringsspråk.
HTML	Verktøy som blir brukt for å sette opp grunnelementene i et nettverksgrensesnitt.
CSS	Verktøy som blir brukt for å bestemme utseende til et eksisterende HTML grensesnitt.
JavaScript	Verktøy som blir brukt for å implementere klient-side logikk i et nettverksgrensesnitt.
Maven	Et prosjekt håndterings verktøy som bygger prosjektet på samme måte uansett plattform.
MySQL	Open-Source relasjonsdatabase, brukt av mange store selskaper.
GitHub	Git Repository tjeneste. Holder på prosjekt filer.

1 Innledning

1.1 Bakgrunn

Visualiseringslabben på NTNU Ålesund har et enkelt kontrollsystem for de forskjellige enhetene som finnes på labben, men systemet mangler i responsiv design og brukervennlighet. NTNU Ålesund vil ha et kontrollsystem som tillater brukeren å skru enhetene av, på, og forandre ulike innstillinger unik til hver type enhet. Oppgaven vil innebære å utvikle et program som kan koble seg på de ulike enhetene og sende ut kommandoer som tillater brukeren å oppfylle de ulike handlingene som enhetene støtter. Programmet skal ha et oversiktlig brukergrensesnitt (Graphical User Interface, GUI) som er funksjonell og samtidig enkel for en bruker å skjønne seg på og utnytte. Enhetene som skal bli brukt av programmet er de som kan koble seg på internett. Oppgaven fokuserer sterkt på projektorer og vi vil derfor ha hovedfokus på dem, og nedprioritere de andre enhetene til implementasjonen av projektorer tilfredsstiller kravspesifikasjonen.

1.2 Mål

Effektmål: Løsningen skal gjøre bruk og vedlikehold av utstyr enklere og raskere.
Resultatmål: Sluttleveransen av prosjektet skal føre til større og mer effektiv bruk av verktøyene på vislabben.
Målsetting:

- Lage en brukervennlig GUI
- Gjøre løsningen så generell som oppgaven tillater
- Responsiv design

1.3 Problemstilling

Det har vært gjennomført en oppgave i 2016 med lignende mål som denne [1]. Den oppgaven hadde som mål å utvikle en styringsapplikasjon for enhetene på vislabben, men med hovedfokus på å utforske mulighetene tilbudt av forskjellige Web Application Framework (WAF). Som et resultat av dette er applikasjonen som ble utviklet nokså simpel i design.
Vår utfordring er dermed å utvikle et brukergrensesnitt som forholder seg til standarder som responsiv design og universell utforming. Dette betyr at brukergrensesnittet vårt skal fungere like godt på alt fra store til små skjermer, både med liten og stor oppløsning. Vi skal også utvikle selve enhetskontrolleren som håndterer kommunikasjonen mellom brukergrensesnittet, applikasjonen og enhetene.

1.4 Kravspesifikasjon

Kravspesifikasjonen består av en rekke ferdig formulerte brukerhistorier (user stories) i tillegg til noen tekniske krav (Vedlegg A). I de første møtene med oppdragsgiver fikk vi definert noen flere krav:
Administratorpanel
En administrator av programmet skal ha mulighet til å legge til, fjerne og organisere enheter.
Lydanlegg
Det er montert et lydanlegg med en forsterker som er koblet til nettverket, denne skal vi også kunne kontrollere gjennom applikasjonen vår.

Ytterligere tekniske krav

- **SonarQube:** Vi skal ta i bruk og teste ut funksjonaliteten i SonarQube.
- **Docker:** Vi skal teste ut og forsøke å ta i bruk Docker til å levere den ferdige applikasjonen.
- **Utvidbar applikasjon:** Applikasjonen skal kunne utvides med flere enhetstyper ved behov senere.
- **Trådbasert kommunikasjon:** Kommunikasjonen med enhetene skal være trådbasert, slik at om man skal utføre en handling på mange enheter på én gang, så skal ikke handlingen gå i sekvens, men tilnærmet på likt for alle enhetene.

Dersom vi ser på kravspesifikasjonen slik den var gitt, per det bare definert én rolle. Men om vi tar med de ytterligere kravene som ble definert i møtene, så ser vi at det er faktisk to roller i systemet: administrator og bruker. Brukerhistoriene vi har brutt dette ned til ser dermed slik ut:

Som administrator

- må jeg kunne sette opp grupperinger for enhetene
- må jeg kunne behandle brukere og enheter
- må jeg kunne sette opp den grafiske visningen av enhetene
- må jeg kunne logge inn i applikasjonen

Som bruker

- må jeg kunne slå en projektor på og av
- må jeg kunne slå testbilde til en projektor på og av
- må jeg kunne slå bildesignalet til en projektor på og av
- må jeg kunne slå en LCD-skjerm på og av
- må jeg kunne velge inngangskilde til en LCD-skjerm
- må jeg kunne markere én eller flere enheter ved å klikke på dem
- må jeg kunne lagre grupperinger av enheter til senere bruk
- må jeg kunne utføre en handling på valgte enheter
- må jeg kunne kontrollere et lydanlegg
- må jeg kunne logge inn i applikasjonen
- må jeg kunne interagere med applikasjonen gjennom et web-brukergrensesnitt
- må jeg kunne se hvilke innstillinger som er satt på en enhet
- må jeg kunne se statusen til en projektor
- må jeg kunne se statusen til et lydanlegg

1.5 Rapportens videre innhold

- **Kapittel 2 - Teoretisk grunnlag**
  I denne delen legges det frem teori som ligger til grunn for resten av rapporten.
- **Kapittel 3 - Materialer og metode**
  Dette kapitlet beskriver hvordan vi har brukt ressursene tilgjengelige for oss til å komme frem til resultatet vi har fått.
- **Kapittel 4 – Resultater**
  Her legger vi frem resultatet vårt.
- **Kapittel 5 – Drøfting**
  I dette kapitlet diskuterer vi hvordan resultatet henger sammen med stoffet som er presentert i Kapitlene 2 og 3.

- **Kapittel 6 – Konklusjon**
  I kapittel 6 trekker vi sammen alle røde tråder og vurderer resultatet opp mot målsetningen og kravspesifikasjonen.
- **Kapittel 7 – Referanser**
  I denne delen legger vi frem kildene vi har brukt i stilen Vancouver.

2 Teoretisk grunnlag

2.1 State Pattern

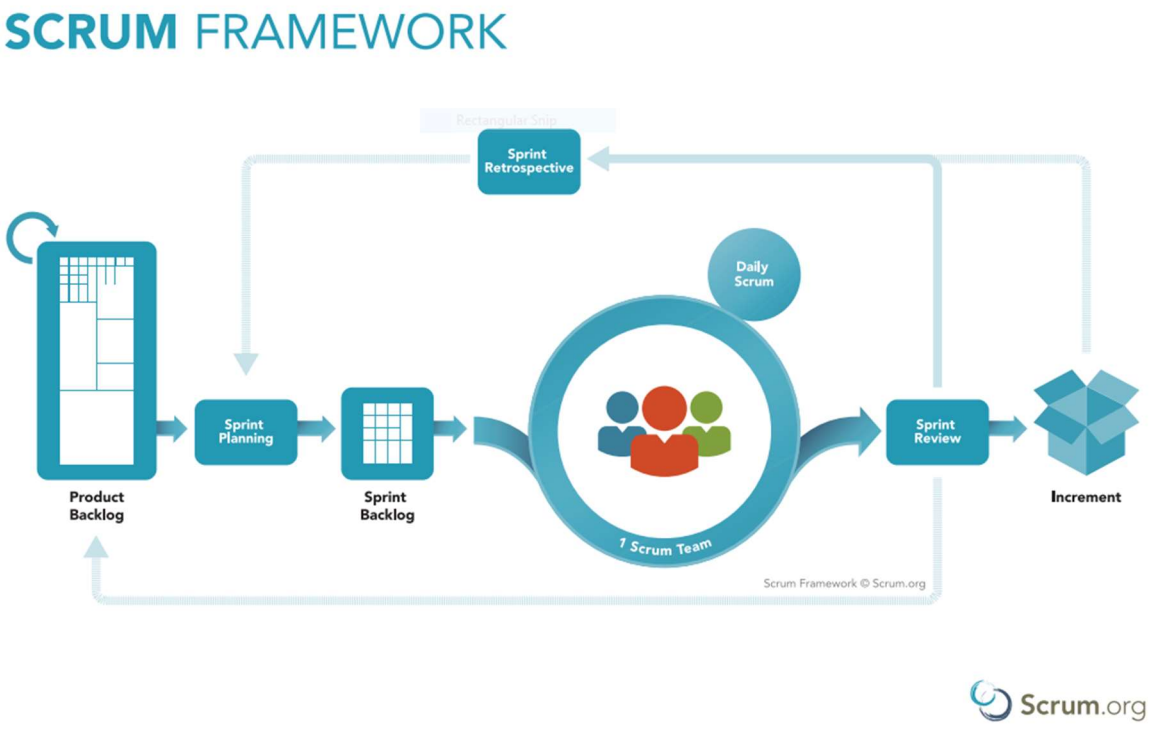
State Pattern [2] er en struktur metode for å minske «coupling» i programmet i tillegg til å øke «cohesion». I Java går dette ut på å representere forskjellige tilstander i ei tilstandsmaskin som individuelle klasser. Hver klasse har ei oppgave, og når den er gjort, bytter den til neste passende tilstand. Det blir tatt i bruk en kontekst klasse som held på hva tilstand den er i. Når «state patternet» er i bruk, blir det ikke fortalt til resten av programmet hva tilstand den er i.

2.2 Agile metoder

Agile metoder [3] er et fellesbegrep på en samling arbeidsmetoder som er utviklet for å forenkle prosessen med å håndtere endringer i kravspesifikasjonen . Hoved-konseptet bak agile metoder er å arbeide iterativt og inkrementelt. Det vil si at teamet som arbeider på prosjektet planlegger et inkrement om gangen, og neste inkrement tar utgangspunkt i forrige, slik at hele prosessen blir en iterativ prosess som tilpasser seg prosjektet over tid i stedet for at alt må planlegges fra starten. En stor fordel med agile metoder er tilbakemelding fra produkteieren på slutten av hvert inkrement, slik at produktet etter hvert inkrement nærmer seg mer og mer hva kunderepresentant ønsker.

2.3 Scrum

Scrum er et arbeids-rammeverk [3] basert på agile metoder. Scrum betraktes som en av de mest utbredte rammeverkene innenfor agile metoder. Det blir beskrevet som et rammeverk fordi innenfor Scrum kan en ta i bruk andre arbeidsmetoder og -teknikker. Scrum er i hovedsak utviklet for å lage informasjonssystemer, men kan brukes til andre typer prosjekter.
Scrum sitt fokus er også å forbedre kommunikasjon mellom ansatte og arbeidsgiver gjennom arbeidsprosessen for å få ut et resultat som ligner mest mulig på det arbeidsgiveren ønsket seg. Scrum er basert på empiriske prosesser, det vil si at en jobber iterativt. Teamet utvikler først en periode, ser hvor de kommer, så planlegger de neste arbeidsperiode.



Figur 2-1 Scrum rammeverket [4]

Som vist på Figur 2-1, så består Scrum av flere faser:

- Sprintplanlegging
- Sprint
- Sprint evaluering
- Spring retrospektiv

Disse fasene vil bli nærmere forklart senere.
For at Scrum skal fungere ordentlig har det også noen kritiske roller for hver involvert person, disse rollene er:

- Produkteier
- Scrum Master
- Utviklingsteam

2.3.1 Sprintplanlegging

Arbeidet som skal utføres i en sprint vil bli planlagt i sprintplanleggingen. Her vil hele teamet samarbeide om å fylle en sprint «backlog» med arbeidsoppgavene som skal utføres. En «backlog» er en liste av oppgaver som må utføres. I Scrum er det «Product Backlog», som er alle oppgavene som må bli utført for å få et ferdig produkt, og «Sprint Backlog» som er oppgavene som må bli utført for å fullføre en sprint. Sprintplanlegging skal svare på to spørsmål:

- Hva skal resultere fra denne sprinten?
- Hvordan vil arbeidet i denne sprinten bli oppnådd?

Det store fokuset med planleggingen vil være å fylle en «Sprint Backlog». Arbeidet som blir planlagt i Sprint Backloggen er ikke satt i stein at det må utføres, men heller en omtrentlig beregning over hvor mye arbeid som skal gjøres den sprinten. For å forenkle beregningen av arbeidet er det ikke uvanlig å tildele poeng, eller «story points», til hver «user story» som skal utføres.

2.3.2 Sprint

En sprint er en iterasjon eller et inkrement av hele utviklingsprosessen [5]. En iterasjon varer som oftest mellom en uke til fire uker. I en sprint vil utviklingsteamet bruke Sprint Backloggen de opprettet i sprintplanleggingen for å bestemme hvilken funksjonalitet som blir utviklet den sprinten. Når en sprint starter så vil ikke Sprint Backloggen bli forandret noe mer med mindre det blir helt nødvendig. Medlemmene av utviklingsteamet står fri til å selv velge arbeidsoppgavene sine fra backloggen etter at sprinten har startet.
På starten av hver arbeidsdag i sprinten vil utviklingsteamet ha et kort stand-up møte hvor de identifiserer mulige utfordringer eller problemer og legger planer for å løse de senere. Møtet er bare for å kartlegge utfordringer/problemer og finne ut av hvem som tar ansvar for å løse de. Møtet er ofte begrenset til 15 minutter. Fokuset i møtet er å svare på tre spørsmål:

- Hva ble oppnådd i går for å komme i mål?
- Hva kan oppnås i dag for å komme i mål?
- Hvilke hinder eller utfordringer kan møtes for å ikke komme i mål?

2.3.3 Sprint Evaluering

På slutten av en sprint vil teamet evaluere sprinten de nettopp utførte. Under evalueringen vil de evaluere arbeidet som ble utført, og arbeidet som ikke ble utført, om de var for optimistisk, eller ikke optimistisk nok. De vil også presentere det utførte arbeidet for produkteieren, ofte i form av en forberedt demo og planlegge med produkteieren hvilken retning de skal ta derfra. Målet med en sprint evaluering er å presentere utført arbeid, og ikke ikke-utført arbeid. Fokuset til en sprint evaluering er selve arbeidet som ble utført under sprinten, og ikke hvordan det ble utført.

2.3.4 Sprint retrospektiv

I en sprint retrospektiv vil teamet reflektere på hvordan arbeidet gikk den forrige sprinten. Teamet vil se på sprinten og diskutere mulige forbedringer slik som arbeidsflyt, effektivitet, samarbeid og andre faktorer som påvirker arbeidsmiljøet. Teamet i hovedsak fokuserer på tre ting:

- Hva burde de slutte å gjøre
- Hva burde de begynne å gjøre
- Hva burde de fortsette å gjøre

2.3.5 Roller

Scrum-rammeverket inneholder tre roller med forskjellige ansvarsområder og oppgaver, rollene er:

- Produkteier
- Scrum master
- Utviklingsteam

2.3.5.1 Produkteier

- Representerer interessentene.
- Ansvar for at produktet har verdi for kunden.
- Spesifiserer kravene for produktet med ikke-tekniske begreper («user stories»).
- Prioritering av funksjoner.
- Kommunikasjon mellom interessenter og utviklerne.

2.3.5.2 Scrum Master

- Ansvar for at Scrum-rammeverket blir brukt og riktig utnyttet.
- Ansvar for at utviklingsteamet organiserer seg selv.
- Hjelpe produkteier med prioriteringer.
- Sørge for at utviklingsteamet bruker sprint retrospektiv og -evaluering ordentlig slik at de hele tiden forbedrer arbeidsflyten og arbeider med de riktige oppgavene.
- En Scrum master har de samme oppgavene som en av utviklingsteamet i tillegg til Scrum master sine.

2.3.5.3 Utviklingsteam

- Ansvar for selve Sprintene med å sette opp Sprint Backloggen, og utføre oppgavene i backloggen.
- Gjennomføre de daglige Scrum-møtene.
- Består som regel av tre til ni medlemmer.
- Sørger for at de har noe å presentere på slutten av hver Sprint.
- Skal være selv-organiserende selv om de noen ganger kommuniserer med roller utenfor Scrum.
- Har de samme arbeidsoppgavene:
    - Design
    - Analyse
    - Testing
    - Dokumentering
    - Utvikling
    - Arbeider til samme tid på samme sted.

2.4 Design

For design har vi tatt utgangspunkt i *Designing with the Mind in Mind* [6], som er en bok som forklarer teorien og prinsippene bak design av GUI. Boken tar for seg temaer slik som fargevalg, utforming av brukergrensesnitt og side-arkitektur. Fargevalget for et grensesnitt skal bruke farger som passer bra sammen og fremkaller følelser fra brukeren som skaperen ønsker å fremkalle. Forskjellige farger har forskjellige assosiasjoner så de har ulike effekter på brukere. Farger skal også kombineres slik at de ikke krasjer med hverandre. Farger kan også bli brukt til å fremvise det gjennomgående temaet til brukergrensesnittet.
Utformingen av brukergrensesnittet bruker flere prinsipper fra psykologi som handler om hvordan mennesker observer objekter som et mønster eller en organisert gruppe. Prinsippene som er lagt vekt på i boken er "Gestaltprinsippene":

- Similarity
- Continuation
- Closure
- Proximity
- Common fate

Similarity er prinsippet om hvordan noe som ser likt ut hører sammen i samme gruppe eller objekt.

Continuation er når folk observer og organiserer skarpe linjer og kurvede linjer til å høre sammen selv om de går over hverandre slik at det stemmer overens med hva de tror de observerer.

Closure er hjernen sin evne til å se ikke-fullførte figurer i sin helhet. Closure blir brukt til å ta oppmerksomhet fra brukeren med å gjemme deler av figurer på en eller annen måte. Det kan være å ikke fullføre figuren helt eller gjemme den delvis bak en annen.

Proximity er når visse elementer blir plassert i nærheten av hverandre slik at de gir inntrykket at de tilhører samme gruppe. Dette blir ofte utnyttet med å plassere for eksempel tekstfelt med knapper i nærheten av hverandre slik at det er synlig at de hører sammen.

Common fate er når objekter tilsynelatende beveger seg i en felles retning med lik hastighet slik at det blir oppfattet at de tilhører samme gruppe. Et eksempel på dette er når en velger flere filer med dra-og-slipp, så vil filene bevege seg som en gruppe mot det valgte punktet.

2.4.1 Responsiv design

Responsiv design [7] er å utvikle nettsiden slik at den tilpasser seg skjermstørrelsen til enheten. Å utvikle en responsiv nettside vil si at utseendet til nettsiden har egendefinerte størrelser for et antall angitte standard skjermstørrelser.

2.4.2 Universell Utforming

Universell utforming er å utvikle et nettsted slik at det tar hensyn til alle innbyggere, inkludert de med varierende grad nedsatt funksjonsevne. Universell utforming har stilte krav fra likestillings- og diskrimineringsloven [8], hvor universell utforming blir definert slik i §17:

> "Med universell utforming menes utforming eller tilrettelegging av hovedløsningen i de fysiske forholdene, inkludert informasjons- og kommunikasjonsteknologi (IKT), slik at virksomhetens alminnelige funksjoner kan benyttes av flest mulig, uavhengig av funksjonsnedsettelse."

Universell utforming sin hensikt er å stille krav til utvikling av applikasjoner slik at de blir så brukervennlige som mulig at spesialkrav for enkelte tilfeller ikke oppstår. Det finnes en standard for web-innhold spesifikt, WCAG 2.0 [9]. WCAG (Web Content Accessibility Guidelines) deler det opp i fire prinsipper:

Første prinsipp: Mulig å oppfatte
Informasjonen som blir presentert skal være mulig for brukerne å oppfatte. Om informasjon blir presentert grafisk, det vil si, oppfattes med øynene så skal den informasjonen ha et alternativ, slik som tekst. Om informasjonen har en alternativ tekst kan den bli oversatt gjennom tekst-til-tale eller lignende hjelpemidler for folk med nedsatt synsevne.

Andre prinsipp: Mulig å betjene
Grensesnittene som blir utviklet skal være mulig for brukerne å ta i bruk. Det skal være mulig å navigere seg rundt i interaktive web-applikasjoner ved å bruke utstyret som brukerne benytter seg av. For eksempel, det skal være mulig for brukerne å navigere seg rundt på et nettsted med hjelp av tastatur.

Tredje prinsipp: Forståelig
Det må være mulig å forstå informasjonen som blir presentert og brukergrensesnittet. Det er viktig at brukeren forstår hvordan nettsiden skal blir brukt og informasjonen som blir presentert. Det er derfor viktig å bruke enkelt språk, god hjelpefunksjonalitet og forutsigbarhet.

Fjerde prinsipp: Robust
En web-applikasjon må være kodet på en måte slik at den er robust nok til at brukeragenter (nettlesere, mediespillere, og slik) kan tolke innholdet på en pålitelig måte. Prinsippet følges som regel med å bruke standard elementer i HTML og lignende. Om en bruker egne elementer (custom widgets) så må en kompensere med spesiell koding.

2.5 Java SPI

Service Provider Interfaces (SPI) [10] er en metode i Java der du kan finne tilgjengelige implementasjoner av en «interface» uten å vite direkte om dem. For å kunne laste dem inn må «ServiceLoader», en standard klasse inkludert i Java, bli brukt. Den søker i "classpath" etter klasser som implementerer et «interface» den har blir gitt, tar i bruk en tom konstruktør i alle implementeringene for å lage objektet. Alle moduler som skal bruke denne tjenesten må legge ved en fil i en spesifikk mappe med linjer om hvilke klasser som implementerer et «interface».

2.6 Hypertext Markup Language (HTML5)

HTML er et markeringsspråk, som blir brukt for å markere forskjellige elementer som blir brukt for å konstruere en nettside [11]. Elementene er byggeklosser som representerer ulike tilstander og funksjoner. HTML har mange ulike markeringer med forskjellige bruksområder og attributter. Den nåværende versjonen av HTML er HTML5.

2.7 Cascading Style Sheet (CSS)

CSS er et stilspråk som blir brukt for å beskrive utseendet til et markeringsspråk slik som HTML. CSS fungerer med å direkte beskrive utseende på et element, eller definere referansepunkter i markeringsspråket som kan være en unik id for et spesifikt element, eller en generell identifikator for flere elementer med samme referansepunkt [11].

2.7.1 Bootstrap

Bootstrap [12] er et rammeverk for HTML, CSS og JavaScript. Bootstrap fokuserer på å bygge responsive og mobilvennlige nettsider. Det er en samling av CSS-klasser og JS-funksjoner som manipulerer utseende på en nettside.

2.8 JavaScript

JavaScript [13] er et tolket programmeringsspråk som støtter flere programmeringsparadigmer, slik som objektorientert, funksjonell og hendelsesdrevet programmering. JavaScript blir ofte brukt for å skripte funksjonalitet på klienten sin side på en nettside. JavaScript standarden er kalt ECMAScript (ES). JavaScript kan bli brukt for server-funksjonalitet, men som oftest blir det brukt for å manipulere Document Object Model (DOM) på en nettside.

2.9 JavaScript Object Notation (JSON)

JSON [14] er et skrive-format for dataoverføring. JSON er lett å generere og lese for datamaskiner, samtidig som det er enkelt å lese og skrive for mennesker. JSON er basert på JavaScript, men er fortsatt uavhengig av språk. Et JSON-objekt er bygget opp av en samling av attributt-verdi-par og en ordnet liste med verdier.

2.10 Unified Modeling Language (UML)

Unified Modeling Language er en gruppe med modeller som er beregnet for å være en standard som visualiserer hvordan et system er designet [15]. UML ble utgitt først i 1994-1995, men ble først tatt opp som en ISO standard i 2005. UML tilbyr en stor variasjon av modeller og diagrammer, hvor modeller er en samling av diagrammer som representerer de ulike komponenter i en modell.
Strukturerte UML diagram:

- Class diagram
- Component diagram
- Composite structure diagram
- Deployment diagram
- Object diagram
- Package diagram
- Profile diagram

Oppførsel diagram:

- Activity diagram
- Communication diagram
- Interaction overview diagram
- Sequence diagram
- State diagram
- Timing diagram
- Use case diagram

2.11 Wireframe

En wireframe [16,17] er som en plantegning for en nettside. Plantegningen representerer et skjelett for selve nettsiden slik at man har noe å jobbe ut i fra når man skal bygge opp selve nettsiden. En wireframe fokuserer på å vise posisjonen til elementene på nettsiden, og de funksjonene tilknyttet hvert element. En wireframe viser bare plasseringer av elementer og funksjoner, og inneholder ingen grafikk, bilder eller lignende.

2.12 Hypertext Transfer Protocol (HTTP)

HTTP [18] er den mest brukte protokollen for overføring av informasjon mellom tjener og klient gjennom Internett. Protokollen er en forespørsel-respons protokoll, dvs. klienten sender en forespørsel [19] til tjeneren som sender tilbake en respons. Responsen [20] er som regel i form av en html-fil og dens tilhørende filer. Når en klient sender en forespørsel er det flere typer forespørsler den kan sende, som:

- GET - klienten spør etter data.
- POST - klienten ønsker å forandre noe data på nettstedet, fører ofte til noe forandring.
- PUT - erstatt data på nettstedet med forespørsel data.
- DELETE - slett spesifisert data.

Når en tjener mottar en forespørsel fra en klient vil den sende en respons. De forskjellige responsene er delt opp i flere hovedgrupper med noen eksempler fra de gruppene:

- 1xx - informerende respons
  - 100 - Continue, tjeneren har mottatt forespørselen og klienten må nå sende forespørsel data om nødvendig.
- 2xx - Suksess
  - 200 - OK, Standard respons for suksessfull forespørsel.
- 3xx - Redireksjon / Videresending
  - 301 - Moved Permanently, denne og alle framtidige forespørsler blir videresendt til ny URL.
- 4xx - Klient feil
  - 400 - Bad Request, tjeneren vil ikke prosessere forespørselen pga. en klient feil.
- 5xx - Tjener feil
  - 500 - Internal Server Error, når noe uventet skjer og det ikke finnes noe mer passende feilmelding.

2.13 Hendelsesdrevet Programmering

Hendelsesdrevet [21] programmering er en programmeringsparadigme hvor målet er å svare til ulike hendelser som kan oppstå. Det blir hovedsakelig brukt i programmering av GUI hvor målet er å utføre oppgaver ut i fra hvilke hendelser brukeren utfører, slik som et museklikk eller et tasteklikk på spesifikke områder. Hendelsesdrevet programmering kan også motta beskjeder, eller hendelser fra andre programmer, og reagere til dem.

2.14 Objektorientert Programmering (OOP)

Objektorientert Programmering (OOP) [22] er et programmeringsparadigme som tar i bruk konseptene objekter og klasser. Objekter er instanser av klasser som inneholder generelle felt for de spesifikke objektene den skal representere. Et objekt er en instansiert klasse med felt som inneholder gyldige verdier. Objekter inneholder metoder som blir brukt for å gjøre forandringer på objektet eller hente ut informasjon fra det. Objektorientert programmering blir brukt der det er viktig at et objekt inneholder data og tilstand for applikasjonen.

2.14.1 Abstraksjon (abstraction)

Abstraksjon [22] handler om å ta komplekse problemer og dele de opp i mindre og mindre problemer til de er enkelt løsbare i mindre form. Etter de mindre problemene er løst kan de bli brukt som byggeblokker for de større problemene.

2.14.2 Innkapsling (Encapsulation)

Innkapsling [22] handler om å skjule all data i en klasse fra andre klasser, slik at andre klasser vet hva metoder klassen har og hva den kan utføre, men ikke hvordan. Om en skriver klassene sine slik, gjør det svært mye enklere å gjøre forandringer på individuelle klasser uten å påvirke andre klasser.

2.14.3 Polymorfi (Polymorphism)

Polymorfi [22] er når en metode utfører forskjellige handlinger avhengig av objektet. Om man lager en generisk klasse så kan en bruke den klassen til å forlenge andre klasser slik at de kan overskrive metodene til den klassen. Når en da kaller en overskrevet metode vil resultatet være avhengig av objektet en bruker den på.

2.14.4 Arv (Inheritance)

Arv [22] er når en klasse får egenskapene og metodene til en annen klasse. Målet til arv er å redusere mengden redundant kode, slik at det blir mer fokus på å skrive unik kode. Arv fungerer på den måten at de blir implementert som en forlengelse av en allerede eksisterende klasse. Den forlengede klassen kan da ta i bruk dens metoder og egenskaper.

2.15 Java

Java [22,23] er et programmeringsspråk som er objekt- og klasse-basert og samkjørende. En av Java sine mest ettertraktede trekk er "write once, run anywhere" (WORA) som lar Java kjøre på maskiner som støtter Java uten re-kompilering. Java kjører som regel på "Java Virtual Machine" (JVM). JVM er en virtuell maskin som kjører applikasjoner skrevet i Java eller er kompilert til Java bytecode. Språket bruker også JIT (Just-in-time compilation) som kompilerer koden når programmet blir kjørt i stedet for å kompilere alt før kjøringen.

2.15.1 Java Virtual Machine (JVM)

Java Virtual Machine [22,24] er en virtuell maskin som lar en maskin kjøre programmer skrevet i Java eller programmer kompilert til Java bytecode. En kan kjøre Java programmer på alle maskiner som har en JVM. Eneste forskjellen på JVM'er er hvordan de er implementert på forskjellige plattformer, ikke hvordan de leser Java bytecode.

2.15.2 Java Runtime Environment (JRE)

Java Runtime Environment [22,24] er en samling med programvare verktøy for utvikling og kjøring av Java-programmer. Det kombinerer Java Virtual Machine (JVM) sammen med en implementasjon av Java Class Library (JCL). JCL består av et sett med dynamiske biblioteker som kan importeres og kalles på under kjøring.

2.15.3 Java Development Kit (JDK)

JDK [25] er en samling av Java verktøy som er nødvendig for å utvikle Java-programvare. Eksempler på disse verktøyene er:

- *javac* - kompilatoren som konverterer Java-kildekode til Java-bytecode
- *javadoc* - genererer dokumentasjon for Java-kode.
- *jar* - arkiveren, den som pakker relaterte Java-biblioteker sammen i en JAR-fil.

2.16 Representational State Transfer (REST)

Representational State Transfer [26] er en arkitekturstil som sørger for interoperabilitet mellom maskiner over Internett. REST ser på data og funksjoner som ressurser, en får tilgang til disse gjennom Unified Resource Identifier IURI. REST tar i bruk noen operasjoner definert av HTTP. Disse operasjonene er: GET, PUT, UPDATE, DELETE, POST og flere. Kommunikasjon med REST er tilstandsløs.

2.17 Java Enterprise Edition (JavaEE)

Java Enterprise Edition [27] er en forlengelse av Java Standard Edition som fokuserer på distribuerte systemer og web applikasjoner. JavaEE blir brukt for å utvikle større systemer som håndterer transaksjoner, sikkerhet, samkjøring, skalerbarhet og styring av komponentene de distribuerer.

2.18 Java Persistence API (JPA)

Java Persistence API [28] er en samling med klasser og metoder for å lagre store mengder data i en database. JPA hjelper til i Java-applikasjoner hvor applikasjonen gjør operasjoner mot en database.

2.19 Sanntidsprogrammering

Sanntidsprogrammering er konseptet å kjøre en blokk med kode parallelt med den allerede kjørende hovedkoden i samme prosess [29]. Alle prosesser eller instanser av applikasjoner har minst en tråd, hovedtråden. Hovedtråden kan starte flere mindre tråder i den samme prosessen som kjører parallelt med hovedtråden. De mindre trådene kan utføre oppgaver som er tidkrevende slik at hele applikasjonen ikke stopper opp når slike oppgaver dukker opp.

2.20 Relasjon Database

En relasjonsdatabase er en samling med datatabeller som oppretter relasjoner og forhold mellom rader i tabellene. En rad representerer et objekt i tabellen, en kolonne representerer en attributt i tabellen og hele tabellen representerer en entitet [30]. Hver tabell har en primærnøkkel som er en identifikator for tabellen. Når en tabell blir bundet opp mot en annen vil tabellens primærnøkkel være synlig i den andre tabellen som en fremmednøkkel. i den andre tabellen som en fremmednøkkel.

2.20.1 Relational Database Management System (RDBMS)

Relational Database Management System (RDBMS) er et Database Management System (DBMS) opp mot en relasjonsdatabase [31,32]. En DBMS er et program som ligger mellom bruker og database. Programmet kan opprette en database, og utføre operasjoner mot den med hjelp av en GUI.

2.20.2 Structured Query Language (SQL)

Structured Query Language, også kjent som SQL er et spørrespråk som er designet for å håndtere data i en relasjonsdatabase. SQL uttrykk er oppbygd av svært lesbar syntaks som enten henter, skaper, oppdaterer eller sletter data fra en tabell [30].

2.20.2.1 MySQL

MySQL er en RDBMS (Relational Database Management System) med åpen kilde-kode. MySQL er tilgjengelig på flere operativsystemer (Windows, Linux, macOS, FreeBSD) [33].

2.20.3 Datamodellering

Datamodellering er å beskrive strukturen til dataobjekter på en systematisk måte. I sammenheng med relasjonsdatabaser vil det være å beskrive relasjonene mellom entitetene [30]. En entitet kan ha forskjellige relasjoner til forskjellige andre entiteter. De er:

- En-til-en
- En-til-mange
- Mange-til-en
- Mange-til-mange

2.20.4 Entitetsrelasjonsdiagram

Et entitetsrelasjonsdiagram er et diagram som viser databasens entiteter og de tilhørende attributtene med entitetene sine relasjoner med andre entiteter.

2.21 Versjonskontrollsystem (VCS)

Versjonskontrollsystem (VCS), eller «version control» system på engelsk, er et program eller en tjeneste som tar vare på tidligere versjoner av prosjektet ditt [34]. Det kan ha annen funksjonalitet i tillegg til dette, som f.eks. å hjelpe til å slå sammen flere versjoner av én fil dersom to eller flere personer har gjort forandringer på den, eller å tilby et felles lagringssted til prosjektfilene.

2.22 Git

Git er et VCS utviklet av Linus Torvalds i 2005 [35]. Git opererer med grener, separat lagring av arbeid og kode, slik at du kan lage eksperimentell kode uten at andre blir påvirket av dette.

2.23 Git Flow

Git Flow er en måte å jobbe med git. Det består av et sett med regler som beskriver hvordan man skal forholde seg til og navngi forskjellige grener i git-repoet [36]. Ved å følge disse reglene blir store prosjekter mer oversiktlige, og det blir lettere å holde øye med fremgangen i prosjektet. Git Flow separer ut utgivelse og utviklings kode i egne greiner, slik at en kunde alltid har tilgang til den nyeste fungerende versjonen av programmet.

2.24 SourceTree

SourceTree [37] er et program som legger frem funksjonaliteten til VCS i et GUI som er lettere å lære, og gir bedre oversikt over prosjektene dine sammenliknet med å bruke et kommandolinjeverktøy.

2.25 Azure/Infrastructure as a Service (IaaS)

Azure [38] er en tjeneste levert av Microsoft. Tjenesten baserer seg på IaaS-konseptet, dette innebærer at du som kunde kan leie lagringsplass eller datakraft i form av en VM som enten kan konfigureres etter en mal, eller som du kan konfigurere etter eget behov. Styrken til IaaS er at det er veldig lett å skalere ytelsen etter behov, det er også lett å starte opp en ny VM takket være maler. Azure har maler som inkluderer oppsett av Docker, MySQL, webserver mm. Bruk av tjenesten opp til en viss grense er kostnadsfritt [39].

2.26 Spring

Spring [40] er et Java-rammeverk som fokuserer på programmering av enterprise-applikasjoner. Spring er fleksibelt nok til å kunne støtte flere typer arkitekturer avhengig av hva brukeren ønsker. Spring er ment å være et tillegg til Java EE og integreres med ulike EE spesifikasjoner. Rammeverket har utviklet en del egne filosofier når det kommer til design:

- Ha et valg i hvert nivå.
- Være imøtekommende for forskjellige perspektiv. Det vil si at Spring støtter flere måter å utvikle en applikasjon på.
- Bakoverkompatibilitet.
- Legge vekt på API Design.
- Høye standarder for kodekvalitet.

2.27 Jenkins

Jenkins [41] er en automatiseringsserver som kan utføre arbeid enten når en hendelse oppstår eller på et gitt tidspunkt. Jenkins har en mengde plugins som gjør at du kan tilpasse serveren til ditt bruk. Jenkins kan utføre oppgaver periodisk eller etter at en hendelse skjer.

2.28 SonarQube

SonarQube [42] er et analyseverktøy som analyserer kode og vurderer om koden møter oppsatt standard. Den poengterer feil, dårlig kode, dårlig kodestil, og mulige svakheter i et prosjekt. Ved å følge med på analysen til SonarQube får man hjelp til å holde kodekvaliteten på et visst nivå slik at koden blir mer oversiktlig og forståelig for andre.

2.29 Docker

Docker [43] er en virtualiseringsprogramvare som virker på operativsystem(OS)-nivå, en applikasjon kan bygges om til en avbildning, som i tur kan kjøres i en container. Docker kan på mange måter sammenlignes med tradisjonelle virtuelle maskiner, men har noen nøkkelforskjeller. I et miljø som kjører Docker har bare vertsmaskinen et OS installert, Docker-programvaren sørger for at hver container får tilgang til nødvendige OS-funksjoner og ikke får tilgang til andre ressurser den ikke skal ha tilgang til.

2.29.1 Docker Compose

Docker Compose [44] er et verktøy som lar deg konfigurere flere Docker images, eller konteinere til å starte opp, konfigureres, og kobles sammen under én kommando. Disse konteinerne blir lagt inn i et eget privat nettverk som ikke kan aksesseres fra utsiden.

2.30 Maven

Maven [45] gir enkel tilgang til en rekke biblioteker og plugins som gjør at arbeidet med å få tak i disse er veldig fort gjort. Dette skjer gjennom et sentralt repository(repo). Maven sentraliserer bygging av et prosjekt i ei "Project Object Model" (POM) fil, noe som gjør at prosjektet kan bygges på samme måte, uansett plattform.

2.31 DevOps

DevOps er en filosofi som binder utvikling og forvalting av programvare tettere sammen [46]. Dette gjør det lettere for utviklere å jobbe mer effektivt. Automatisering av bygging, pakking, testing og utlevering av programvare gjennom en rekke verktøy.

2.32 JUnit

JUnit [47] er et rammeverk for testing av metoder i Java. Å skrive tester lar programmerer sikre at metode svaret forblir det same selv om hvordan ting blir gjort endrer seg.

2.33 JIRA

Atlassian JIRA er et verktøy for å organisere arbeidsflyten i et programvareprosjekt [48]. JIRA er godt egnet til å bruke i forbindelse med agile arbeidsmetoder slik som Scrum, kanban, etc.

2.34 Confluence

Atlassian Confluence er et verktøy for å lage blogg, wiki, og å samle dokumentasjon [49]. Confluence er knyttet sammen med JIRA. Slik at om man jobber med et prosjekt i JIRA, er det mulig å lage rapporter, skjemaer, osv. på Confluence som henter data fra JIRA.

3 Materialer og metode

3.1 Arbeidsmiljø

Ettersom programvaren vår skal styre enheter på Visualisering og Simuleringslab. (Vislab.), var det naturlig å sette seg der. Labben har tre pulter, hver med to skjermer som enkelt kan kobles til egen PC. Det er og satt opp datamaskiner til bruk, men pga. restriksjoner på installasjon av programvare på disse, har vi brukt egne datamaskiner. Vislabben blir vist frem til besøkende, noe som fører til forstyrrelser under arbeid.
Rommet i seg selv har litt bakgrunnsstøy ettersom det er elektronisk utsyr som går døgnet rundt i og at med labben. Det er snakk om projektorer og datamaskiner i labben, servere og elektrisk anlegg i naborom.
Som gruppe, var vi enige om at forstyrrelsene og støyen i labben ikke var store nok til å sitte ved tilrettelagt bachelor rom. Bachelor rommet er ikke privat og har andre grupper som snakker og arbeider hele tiden. Direkte tilgang til enhetene uten å måtte flytte seg var den største grunnen til valget å sitte i vislabben.

3.2 Prosjektorganisasjon

3.2.1 Prosjektgruppen

Prosjektgruppen for dette prosjektet består av følgende studenter med kandidatnummer

- 10035
- 10048
- 10011

Alle medlemmer er avgangsstudenter ved studiet Bachelor i Ingeniørfag - data, ved Institutt for IKT og realfag hos NTNU i Ålesund.

3.2.2 Oppdragsgiver

Arne Styve er oppdragsgiver for dette prosjektet. Det var ønske om å lage en webapplikasjon som lar brukere styre enheter på Visualisering og Simuleringslabben ved NTNU i Ålesund. Ønsket kom av at eneste måte å slå hele labben på var fra ene maskinen på labben i en kommandolinje. Det var utviklet en kravspesifikasjon før januar 2018. Denne så liten til ingen endring i løpet av prosjektet.

3.2.3 Veileder

Veileder for prosjektet var oppdragsgiveren, Arne Styve. Han er universitetslektor ved Institutt for IKT og realfag host NTNU i Ålesund (2018). Ved oppstart av prosjektet ble det fastsatt retningslinjer for Sprint lengde, møte intervall og bruk av agile metoder. Arne Styve var tilgjengelig for veiledning og råd utenom oppsatt møtetid.

3.2.4 Styringsgruppe

Prosjektets styringsgruppe var veileder og oppdragsgiver Arne Styve.

3.2.5 Rådgivning

Det var kun tatt råd fra veileder, etter som teknologier vi brukte og enheter som skulle bli styrt, var ukjent territorium for de fleste. Vi var avhengig av dokumentasjon på enheter [50,51] og arbeid fra et tidligere prosjekt [1].

3.2.6 Prosjektorganisering

Det vart satt krav til bruk av agile metoder, scrum spesifikt (beskrevet i kapittel 2.3), ved oppstart av veileder. I scrum er rollefordelingen en av de viktigste konseptene. Det er tre typer roller i scrum: Produkteier, Utviklingsteamet og Scrum Master. Hele gruppen er med som utviklere, så det må bli flere roller på noen av medlemmene. Disse medlemmene har ansvaret fra begge roller som må opprettholdes.
Organisering ble dermed som vist i Tabell 1

Rolle	Kandidatnummer / Navn
Produkteier	Arne Styve
Utvikler	10011
Utvikler	10035
Utvikler	10048
Scrum Master	10011
Sekretær *	10035

Tabell 1: Prosjektorganisasjon

* Selv om Scrum [52] ikke spesifiserer en sekretær, ble vi som gruppe enig med veileder at noen må ha ansvar for at skriving i rapport, dokumentering og møtereferat blir skrevet.

3.2.6.1 Oppgaver for Utviklingsteam

Utviklingsteamet er selvstyrt og har dermed ingen fastsatt arbeidsleder. Arbeid er tilgjengelig gjennom en backlog med prioritering.
Alle medlemmer i utviklingsteamet har ansvar for:

- Utvikling av produktet
- Levering av en produktdemonstrasjon
- Planlegging av sprint
- Legger issues inn i JIRA backlog
- Dokumentere på confluence
- Skrive JavaDoc
- At arbeid blir utført på en slik måte at målet for framgang nås.

3.2.6.2 Oppgaver for produkteier

Produkteier har ansvar for følgende:

- Å prioritere backlog slik at nødvendig funksjonalitet blir jobbet med først.
- Fokusere på hva som skal gjøres, ikke hvordan.
- Backlog har nødvendige user stories.

En prosjekteier skal bare peke utvikler teamet i den retningen som er ønsket/nødvendig for å nå mål som er satt. Produkteier skal aldri være Scrum Master, men kan være del av utvikler teamet.

3.2.6.3 Oppgaver for sekretær

Sekretæren skal sikre at dokumentering og skriving i rapport blir gjort, enten ved å gjøre det selv, eller delegere arbeidet blant gruppemedlemmer.
Ansvar for:

- Sikre grundig dokumentering av arbeid.
- At møtereferat blir skrevet.
- Interne og eksterne dokumenter og dokumentasjon.
- At all dokumentasjon opprettholder retningslinjer avtalt i gruppen.

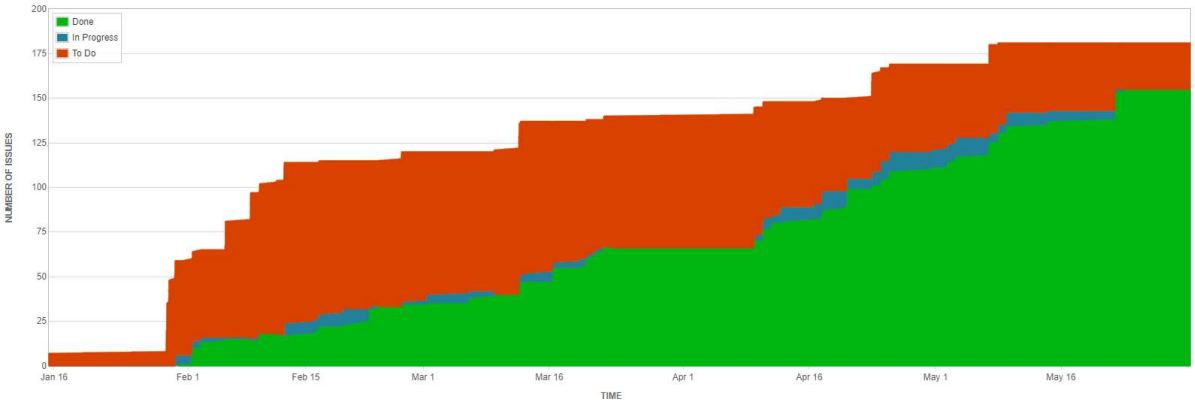
3.2.6.4 Oppgaver for Scrum Master

Scrum master har ansvar for følgende:

- At retningslinjene til Scrum blir fulgt.
- At utviklingsteamet blir selvorganisert.
- Forbedre arbeidsformen gjennom daglige møter.
- Sikre at teamet fungerer godt nok til å nå arbeidsmengden som var planlagt.

3.3 Utviklingsmetodikk

Som beskrevet i kapittel 3.1.6, ble Scrum benyttet som utviklingsmetodikk. Scrum i seg selv er ikke en utviklingsmetodikk men har rammeverk slik at alle som deltar i en scrum prosess finner de metodikker, teknikker og metoder som fungerer best for hele gruppen. Det var fastsatt ved oppstart at vi skulle bruke JIRA(se kapittel 2.34) og Confluence(se kapittel 2.35) som verktøy og at sprintene skulle være 14 dager lange, så lenge det ikke var grunnlag for noe annet.
Gjennom studiet har vi sett og lært at scrum og agile metoder fungere mye bedre enn noe annet når det gjelder programvare utvikling. Det har mye med at utviklingsprosjekter kan mye enklere bytte ut grunnstruktur enn det et tradisjonelt byggeprosjekt kan. Dette kan medføre uforutsette problemer (bugs). Ved å ta hyppige sjekker på kode funksjonalitet, minsker faren for at kritiske eller fundamentale feil forblir uoppdaget i lengre tid. Det er denne tankegangen Scrum er bygd på, korte sprinter og når en sprint avsluttes skal all fungerende kode leveres og vises frem til produkteier.
Siden prosjektgruppen er på kun tre personer og en fjerde som produkteier, blir nøyaktig følging av Scrum vanskelig. Vi har implementert det meste av Scrum sine retningslinjer for arbeidsflyt inn i vår arbeidsflyt.



Figur 3-1: Kumulativ arbeidsmengde

3.3.1 Backlog

Backloggen er fylt med «user stories», beskrivelse av hva en bruker ønsker å oppnå ved visse handlinger. Disse beskriver ikke hvordan det skal oppnås men heller hva ønsket resultat er. Disse er lagt inn i samarbeid med produkteier, mens faktiske oppgaver blir lagt inn som element under hver enkelt «user story». Eksempel:

- Som en bruker vil jeg kunne slå på og av enheter.

Dette er en «user story». Under oppgaver til denne vil kunne være:
Lag interface til kommunikasjon

- Implementer kommunikasjon til enhet
- Design GUI
- Knytt funksjon til GUI gjennom knapper og bruker handlinger.

Det er en daglig oppgave å tenke ut hva som skal gjøres og hvordan det skal gjøres. Alle utviklere i teamet skriver oppgaver, enten som under oppgaver av andre oppgaver/user stories eller enkelt oppgaver (fiksing av bugs). Desse blir lagt inn i backloggen klar til å legges inn i neste sprint. Kritiske oppgaver eller bugs som oppdages midt i en sprint blir lagt til den pågående sprinten.
Oppgaver og user stories har statuser for å enkelt kunne se hva som skal gjøres, er i gang eller har blitt gjort. Det er statusene “To Do”, “In Progress” og “Done” i JIRA.

3.3.2 Sprint

Våre sprinter varer i 2 uker etter krav fra veileder. En sprint starter på mandag, og blir avsluttet på fredag uken etterpå.

3.3.3 Sprint planlegging:

Dette ble gjort etter helgen for å la utviklerne tenke på hvilke oppgaver som kan gjøres neste sprint. Dette valget gir også utviklerne tid til å slappe av og roe seg ned til neste sprint.

3.3.4 Daily Standup/Scrum:

På grunn av vårt arbeidsmiljø, har dette blitt integrert som en del av arbeidet. Det er klar kommunikasjon internt i utviklingsteamet mens arbeid pågår, slik at møtet i seg selv ble overflødig.

3.3.5 Sprint Review:

Dette ble gjort klokken 13:00 hver annen fredag, sammen med produkteier.

3.3.6 Sprint Retrospective:

Dette ble tatt muntlig med teamet ved sprint avslutting.

3.4 Valg av rammeverk

3.4.1 Valg av Web-Application framework (WAF)

Ved oppstart av prosjektet ble vi som gruppe enig om at vi skulle holde oss mest mulig til rammeverk og metoder vi hadde kjennskap til. Gjennom en tidligere rapport [1], der undersøkelse av WAF ble gjort, så vi på populære rammeverk og trendene som viste seg i tallene. Spring var noe vi hadde kjennskap til så Spring Boot var vårt førstevalg. Spring Boot har integrert webserver, Tomcat i biblioteket. Biblioteket tar seg av mye av konfigureringen som ellers ville vært nødvendig, og bygger på kjent syntaks fra Java EE [53].

3.4.2 Valg av Web-rammeverk

Gjennom tidligere erfaringer var vi klar over at utvikling av en bra og funksjonell GUI vil ta en god del mer tid enn nødvendig om en vil utvikle den fra grunnen av. Derfor bestemte vi oss for å bruke et rammeverk for å assistere oss i å utvikle en GUI. Allerede før vi begynte å lete etter et passende rammeverk var vi klar over allerede eksisterende rammeverk som oppfylte det vi trengte dem for. Vi endte opp med å bruke Bootstrap simpelthen fordi det er et godt etablert rammeverk med en stor mengde dokumentasjon både fra sider som https://www.w3schools.com/ og https://getbootstrap.com/.

3.5 Metode

I denne seksjonen vil det bli tatt opp arbeidsmetode, valg som ble tatt, hvordan problemer ble håndtert og hvordan prosjektet ble prioritert.

3.5.1 DevOps

DevOps ble implementer gjennom Jenkins, SonarQube, Docker, Git og Maven. Git gir beskjed til Jenkins når en ny push har skjedd, så bygger Jenkins programmet. Etter bygging blir det kjørt tester og levert til SonarQube for analyse. Etter suksessfull bygging og testing blir et Docker Image bygd og levert til Docker serveren.
DevOps ble implementert i rundt månedsskifte fra februar til mars. Vi tok nytte av oppsettet fram til midten av april når vi moduliserte prosjektet. Med prosjektet delt inn i forskjellige moduler ville det blitt brukt for mye tid på å implementere hele prosessen igjen.

3.5.2 IaaS/Azure

Vi leste litt om Azure og prøvde ut tjenesten litt, men bestemte oss til sist for å ikke benytte tjenesten. Denne avgjørelsen var basert på at det ville være mer praktisk å benytte oss av skolens eget VM-tilbud, og at vi mest trolig ville kommet til å bruke mer ressurser fra Azure enn hva som er tilgjengelig kostnadsfritt. I tillegg til det så er kommunikasjon med projektorene stengt fra utsiden av skolens nettverk, dette medfører at vi hadde vært nødt til å han en server lokalt uansett, dermed er det lite å vinne på å bruke Azure.

3.5.3 MySQL

For å lagre og hente all informasjonen som blir brukt i applikasjonen trengte vi en database for å holde på den. Ingen i gruppen hadde noe ekstra erfaring når det kom til SQL eller noSQL databaser. Vi valgte derfor utifra det vi kunne allerede som var SQL og relasjonsdatabaser. Vi bestemte oss derfor for å velge en allerede godt kjent og dokumentert RDBMS (Relational Database Management System). Vi la ikke mye tanke inn i valget av RDBMS, annet enn de allerede nevnte kriteriene: godt kjent og dokumentert.

3.5.4 Database

Databasen er bygd ved hjelp av Spring. I starten lagde vi en enkel modell som viste hvordan vi tenkte at databasen skulle se ut. Deretter brukte vi modellen av databasen til å gjøre om databasen til klasser i Java med hjelp av Spring-biblioteker. Spring håndterer videre resten av jobben med å lage tabeller med riktige kolonner, primær- og fremmednøkler og riktige forhold mellom hver tabell.

3.5.5 Service Provider Interfaces

I prosjektet bruker vi Service Provider Interfaces (SPI) til å ta i bruk nye moduler. Disse modulene blir lastet inn av ServiceLoader [54] automatisk ved hjelp av identifikasjonsmetoder grensesnittet. Det er implementert annotasjoner som skal generere konfigurasjons filene automatisk, @ProjectorSPI og @DeviceSPI. Disse skal plasseres i alle klasser som implementerer «*Projector*» og «*Device*» grensesnittene våre. Annoteringene er en spesialisert variant av META-INF/services generator [55], et generelt bibliotek for auto generering av service konfigurasjons filer.
SPI er hjørnesteinen for å gjøre programmet lett utvidbart, siden det blir ikke nødvendig å rekompilere hele prosjektet for å utvide programmet med nye moduler. Nye moduler kan bli bygd uavhengig av programmet og lagt til i en «plugin» mappe. Denne mappen blir søkt av programmet for å finne .jar filer for å legge til for å gjøre tilgjengelig. Hver modul melder fra til ServiceLoader hvilken klasse som implementer hvilket grensesnitt slik at de kan bli funnet. ServiceLoader går dermed igjennom en liste over klasser som har meldt seg som en implementasjon av grensesnittene for å opprette objektet.

3.5.6 JUnit

JUnit ble brukt i starten, men ble avviklet fordi testing av tråder og enheter ble for tidskrevende for gruppen.

3.5.7 Sikkerhet

Der har vært noen tanker om det med sikkerhet. Vi vil ikke at hvem som helst skal kunne styre enhetene og lage endringer i databasen. Nettsiden ligger bak et eget innloggingssystem bygd på Spring Security [56], en komponent i Spring. Når en bruker prøver å gå inn på nettsiden vår blir de bedt om å logge inn. Bruker data er lagret med kryptert passord i databasen, gjennom BCrypt.
Der har vært et forsøk på å implementere SSL. Gjennom konfigurasjon med Spring Security er det et eget signert sertifikat på nettsiden som beskytter mot at informasjonen blir fanget opp i brukbar form i mellom kommunikasjonspunkt.

3.5.8 Sanntidsprogrammering

Et av kravene som er stilt til oppgaven er at kommunikasjonen med enhetene skal være trådstyrt, noe vi har løst på følgende måte:
Vi har en hovedtråd som applikasjonen kjører på, og for hver aktiv enhet er det en kommunikasjonsdriver som kjører i en egen tråd. Driveren har ansvar for å holde kommunikasjonen gående med sin tildelte enhet og lytte etter input fra brukeren og håndtere den. Tråden opprettholder regler og krav til tid for kommunikasjon til enhetene.
Barco F22 har veldig lange ventetider mellom én kommando sendes til neste, spesielt etter oppstart, og etter 20 kommandoer. Denon DN500-AV er derimot en del raskere, og krever bare noe lengre enn normal ventetid etter den slåes på. Dette er kun de to enhetstypene vi endte opp med å utvikle støtte for i prosjektet. Og om man tar høyde for at det er 12 Barco F22 enheter og 1 Denon DN500-AV i tillegg til utvidingsmulighetene for applikasjonen, så begynner man å se behovet for å ta i bruk trådprogrammering.

3.6 Materialer

3.6.1 Hardware

PC

- Windows 10 PC

Projektor

- Barco F22-series

Lyd

- Denon DN500-AV

Virtuelle Maskiner hos NTNU i Ålesund

- Ubuntu 16.04 - Docker + PostgreSQL + JDK8
- Windows Server 2012 R2

Projektorene som vi skal styre er montert nede i visualiseringslabben, selve monteringen av projektorene har liten relevans til denne oppgaven. Projektorene er koblet opp til nettverket på skolen, og vi kan bruke dette til å kommunisere med projektorene via en TCP/IP-protokoll.

3.7 Programmeringsspråk

3.7.1 Java

Som et objektorientert programmeringsspråk, vil struktur og koding i Java automatisk følge en gitt form. Arv og polymorfisme er hjørnesteiner i vår design av Java programmet. Oppdaging av moduler gjennom et felles grensesnitt, kjøre metoder på objekt som ikke vet hva er helt spesifikt. «Device» og «Projector» er grensesnittene som gjør at hele programmet fungerer. Hver enkelt utvidelse implementerer disse grensesnittene i tillegg til sine egne for å kunne kjøre. Det er i gjennom Java sin ServiceLoader(beskrevet i kapittel 3.5.4) vi finner de rette klassene.

3.7.2 SQL

For å opprette og operere mot en database blir spørrespråket SQL benyttet. For dette prosjektet benyttet vi SQL for to operasjoner. Scriptet vil opprette en standard bruker med admin-rettigheter og legge til støttede enhetstyper (Projector, Device, Sound System").

3.7.3 JavaScript

For å bestemme hvordan nettsiden skal oppføre seg mot bruker interaksjon og når forskjellige hendelser oppstår blir JavaScript benyttet. JavaScript brukes som oftest i samspill med HTML for å gi unik oppførsel til HTML elementer. Det blir også som kommunikasjonslaget mellom Java og HTML. I dette prosjektet benyttes JavaScript kun for å manipulere HTML og vise hentet informasjon.

3.7.4 HTML

Markeringsspråket HTML benyttes for å bygge alle grunnelementene på nettsiden. HTML brukes til å strukturere informasjonen, det vil si HTML forteller hva som er overskrifter, lister, input-felt og videre. HTML er for øyeblikket versjon 5, men kommer med fler og fler utvidelser og oppdateringer til nyere krav, slik som responsiv design. Et av målene til HTML er å redusere antall avhengigheter nettsider har, de oppnår dette med å implementere knapper med pre-definert logikk slik som submit-knapper og metoder for å spille av video direkte i HTML i stedet for å bruke applikasjoner slik som Silverlight, Adobe Flash eller QuickTime.

3.7.5 CSS

For å bestemme styling på de ulike HTML-elementene som ble tagget har vi brukt Cascading Style Sheet. Hver .html-fil får CSS fra tre kilder. En unik .css-fil for hver .html-fil, en overordnet .css-fil som inneholder fargevalg og generell styling, og importen til Bootstrap. Grunnen til at vi valgte å ha tre kilder for CSS for hver fil var på grunn hvor enkelt det er å gjøre forandringer uten å ødelegge noe annet, og om vi vil forandre f.eks. farger trenger vi kun å forandre i èn .css-fil. Fordelen med å inkludere Bootstrap i alle .html-filene er at det vil gi tilgang Bootstrap sine .css-klasser til alle .html-filene slik at vi kan utnytte Bootstrap sitt grid-system og klasser for posisjonering og simple stiler.

3.8 Utviklingsverktøy

3.8.1 NetBeans IDE

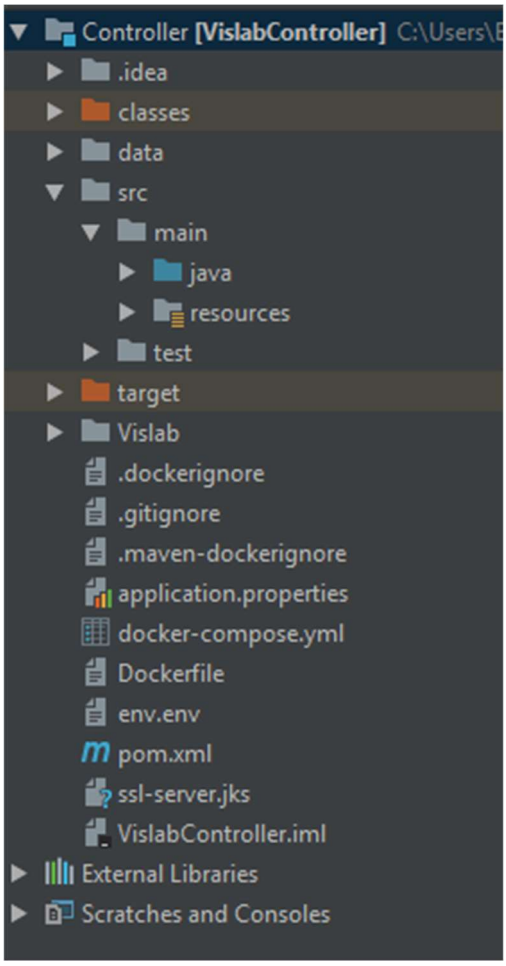
NetBeans er en open source og gratis Integrated Development Environment(IDE). Til å begynne med brukte vi NetBeans til å utvikle prosjektet, vi brukte NetBeans fordi det er et IDE vi har blitt kjent med gjennom skolen, og det har innebygget funksjonalitet for svært mange av rammeverkene og verktøyene vi bruker i prosjektet. NetBeans er Java 8 sin offisielle IDE. NetBeans støtter andre språk enn Java, slik som HTML, CSS, C/C++, JavaScript og flere. Dette tillater utviklere som bruker NetBeans for å lage større applikasjoner som krever andre moduler enn rene Java-moduler å samle alt på en plass.
NetBeans ble brukt eksklusivt i starten, men etterhvert som vi kom i gang byttet vi IDE over til IntelliJ. En av grunnene vi valgte å bytte over til IntelliJ var fordi vi får gratis tilgang med NTNU. De funksjonelle grunnene var:

- Code finish
- Auto-import
- Implementert Maven- og Gradle-støtte
- Gode plugins for rammeverk
- Støtte til valgfri VCS

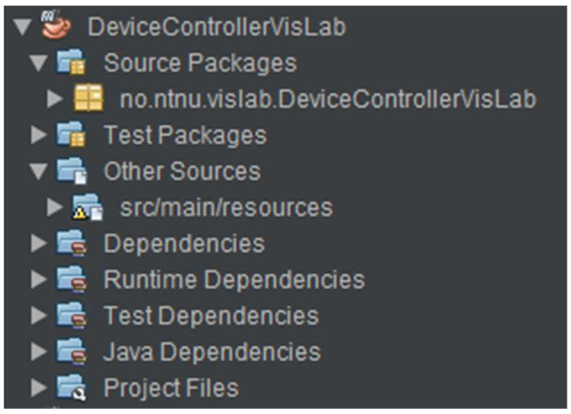
Vi brukte NetBeans til å utvikle selve Java-applikasjonen sammen med front-end løsningen som ble laget med HTML, CSS og JavaScript.

3.8.2 IntelliJ IDEA

IntelliJ er en IDE som er mye brukt i profesjonell setting og lisens-basert. Med IntelliJ har en tilgang til “code finish” og auto-import. Det er implementert Maven og Gradle støtte, i tillegg til ditt valg av VCS. Gjennom en mengde plugins kan du få kode hjelp støtte til rammeverk du bruker i prosjekter.
Vi startet ikke med IntelliJ før litt senere inn i prosjektet. Hovedgrunnen til at vi byttet var fordi filstrukturen i NetBeans var uoversiktlig sammenlignet med IntelliJ sin, se Figur 3-2 og Figur 3-3.



Figur 3-2: Prosjektstruktur i Intellij



Figur 3-3: Prosjektstruktur i NetBeans

3.8.3 Paint.net

Paint.net er et bilderedigeringsprogram som er gratis og kan bli sammenlignet med andre mer kjente kraftige bilderedigeringsprogram som Adobe Photoshop og lignende. Paint.net ble brukt i dette prosjektet for å lage de ulike ikonene som blir tatt i bruk.

3.8.4 Visio

Visio er et Microsoft program som lar brukeren enkelt lage diagrammer, prosessmodeller og visualisere data. Visio er et av programmene som studenter hos NTNU får tilgang til gratis. Vi brukte Visio til å bygge wireframes for GUI-utvikling og modellere en relasjonsdatabase for databaseutvikling.

3.9 Eksterne biblioteker, verktøy og tillegg

3.9.1 Bootstrap

For å designe grunnstrukturen til brukergrensesnittet tok vi i bruk Bootstrap. Bootstrap tilbyr en stor kolleksjon av klasser som er rettet mot responsiv design. I grensesnittet brukte vi Bootstrap sitt grid-layout. Grid-layoutet deler hele grensesnittet opp i en tabell med 12 kolonner, og selv definert mengde rader. Det betyr at vi kan nøyaktig plassere elementer på siden med enkle klasse-deklarasjoner.

3.9.2 Spring Boot

For å utvikle applikasjonen brukte vi Spring Boot [57] som er et sett av rammeverk som kan sees på som en ut-av-boksen applikasjon. Sprint Boot sine salgspunkt [58] er:

- Lage egne Spring applikasjoner
- Innebygd applikasjonsserver (Tomcat, Jetty eller Undertow).
- Enkle starter POM-filer for enklere Maven konfigurasjon.
- Automatisk konfigurasjon av Spring når mulig.
- Tilbyr produksjons-klare funksjoner.
- Ingen kode generering eller nødvendighet for XML konfigurering.

I tillegg til de nevnte punktene tilbyr Spring Boot mye hjelp for web-applikasjoner, SQL-database integrasjon og RESTful API. For applikasjonen tok vi i bruk flere av Spring Boot sine funksjoner. Hovedsakelig de som hjalp oss med å sette opp en web-applikasjon mot en SQL-database, slik som, SpringApplication for å starte selve applikasjonen, Spring Data for datahåndtering med entiteter og spring repositories og Spring Controller for hva slags funksjonaliteter som eksisterer på hvert web-område.

3.10 Distribuering av applikasjon

3.10.1 Jenkins

Allerede ganske tidlig i prosjektet installerte vi Jenkins (27) på en virtuell maskin etter ønske fra oppdragsgiver om å ta i bruk SonarQube. SonarQube kjøres ved hjelp av Jenkins. Jenkins har også vært brukt til å bygge prosjektet ved hjelp av maven for så å levere applikasjonen til en Docker container.

3.10.2 Docker

Etter ønske fra oppdragsgiver har vi brukt Docker. Opprinnelig har ønsket vært å bruke Docker slik at man får en ett-klikk-løsning for å starte applikasjonen i et nytt miljø. Dette har vist seg å bli problematisk da vi bruker en MySQL database som hører sammen med applikasjonen. MySQL databasen vår kan ikke kjøre i samme Docker-container som resten av applikasjonen, og må kjøre i en egen container. Vi kan likevel automatisere dette i stor grad, slik at det blir minimalt med klikk for å starte applikasjonen fra bunnen av.
Mot slutten av prosjektet fant vi et verktøy som heter Docker Compose (30). Dette er et verktøy som lar oss definere alle komponentene prosjektet er avhengige av, for så å bygge og kjøre det med én kommando.

3.10.3 Maven

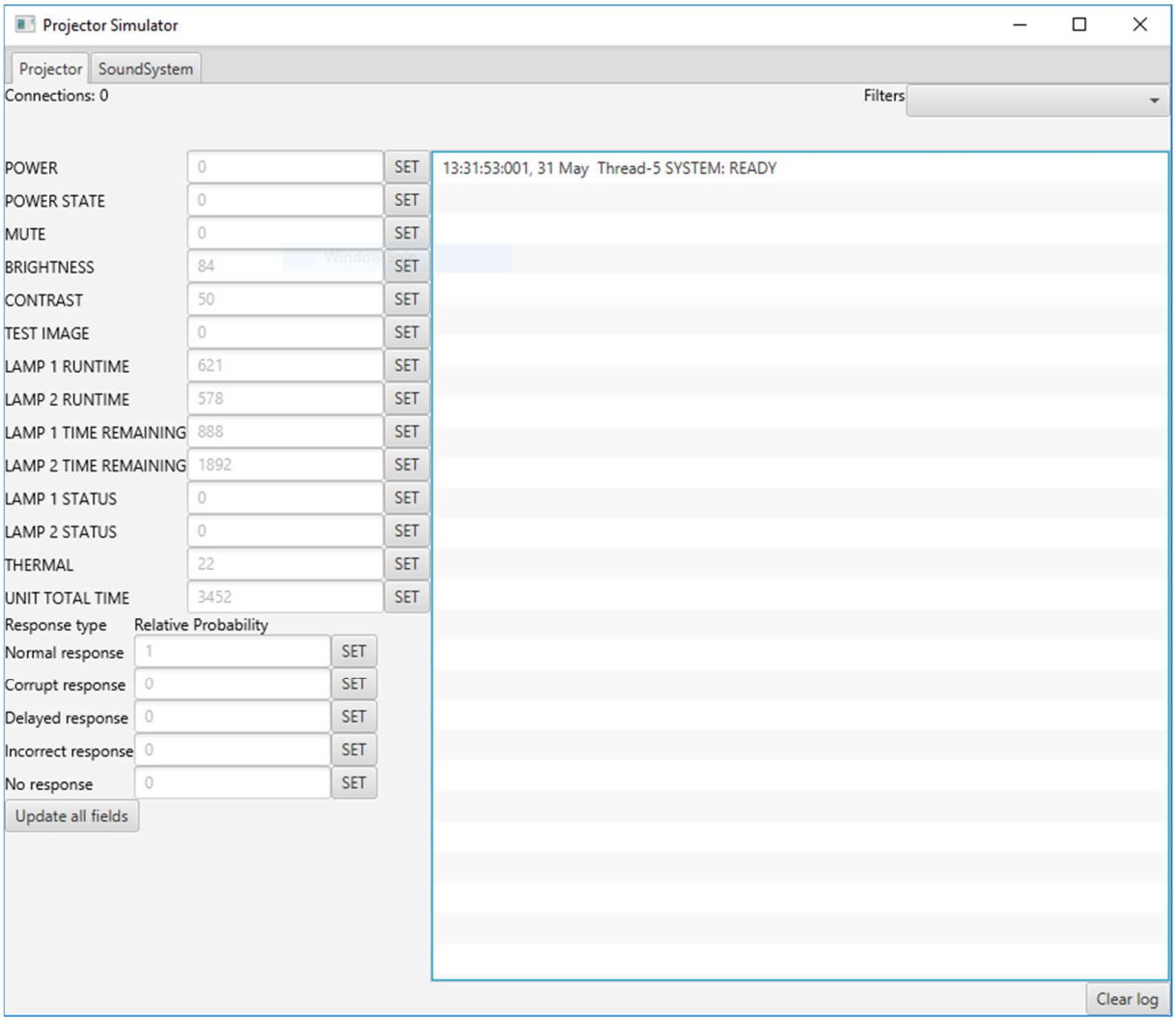
Maven ble brukt til å handtere alle avhengigheter gjennom POM filen og Maven Central Repository(MCR), ei samling av java bibliotek tilgjengeleg over nett. MCR kan finnes her: https://mvnrepository.com/repos/central. Ved hjelp av dette kunne alle gruppemedlemmer enkelt legge til avhengigheter som ble lastet ned hos dei andre. Maven ble brukt til å bygge og dermed flytte alle relevante filer til ei mappe slik at det var klar til bruk.

3.10.4 DevOps

DevOps ble implementer gjennom Jenkins, SonarQube, Docker, Git og Maven. Git gir beskjed til Jenkins når en ny push har skjedd, så bygger Jenkins programmet. Etter bygging blir det kjørt tester og levert til SonarQube for analyse. Etter suksessfull bygging og testing blir et Docker Image bygd og levert til Docker serveren.
DevOps ble implementert i rundt månedsskifte fra februar til mars. Vi tok nytte av oppsettet fram til midten av april når vi moduliserte prosjektet. Med prosjektet delt inn i forskjellige moduler ville det blitt brukt for mye tid på å implementere hele prosessen igjen.

3.11 Testing

3.11.1 Simulator



Figur 3-4: Simulator brukergrensesnitt

Vi utviklet en simulator, som vist på Figur 3-4, for å kunne imitere mottak og respons fra enheter som skulle styres. Den håndterer flere tilkoblinger samtidig og klarer dermed å fremstå som flere enheter. Den er bygd i JavaFX. Simulatoren ble brukt veldig mye i starten av prosjektet for å unngå unødvendig stress/skade på enheter pga. kommunikasjonsprotokollen er veldig streng på timing og venting.

3.11.2 Testing av Nettside

For testing av nettside brukte vi hovedsakelig Google Chrome for å utvikle den, men tok også i bruk Mozilla sine verktøy, Mozilla Firefox og til tider, Firefox Quantum Developer Edition. Vi testet også ut nettsiden på Safari og Edge, men brukte ikke de til noe annet enn testing etterhvert som det var nødvendig.

- Mozilla Firefox [59] er en nettleser som støtter alle de store operativsystemene, Windows, Linux, macOS og iOS. Firefox Quantum Developer Edition [60] er en utvidelse av Firefox rettet mot utviklere og inneholder flere nye verktøy, som: FireFox DevTools, Firebug, og flere.

- Google Chrome [61] er en nettleser utviklet av Google som støtter de fleste operativsystemer. Gamle versjoner av Chrome kjører på WebKit-motoren, men de nyere kjører på Blink-motoren som er utviklet av Google. Store deler av både Chrome og Blink er utgitt som open source.
- Safari [62] er nettleseren beregnet for å kjøre på Apple sine operativsystemer, iOS og macOS. Safari var originalt kun beregnet for å kjøre på Apple sine enheter, men har siden 2007 blitt gjort tilgjengelig på Windows, og ble i 2012 slutt på støtten for Windows.
- Microsoft Edge [63] er Windows sin standard nettleser etter at Internet Explorer [64] ble avviklet i 2015 da Edge kom ut. Edge er standard nettleseren for alle nyere Microsoft systemer, Windows 10, Windows 10 Mobile og Xbox One.

Grunnen til at vi tester på flere nettlesere er fordi de er bygget forskjellig, og vil derfor støtte ulike implementasjoner på ulike måter. Internett brukerbasen er splittet over flere nettlesere, men vi tok hensyn til de mest brukte Firefox, Chrome, Safari og Edge [65]. Se Figur 3-5.

2018	Chrome	Edge/IE	Firefox	Safari	Opera
April	78.6 %	3.9 %	11.2 %	3.3 %	1.5 %
March	78.1 %	4.0 %	11.5 %	3.3 %	1.6 %
February	77.9 %	4.1 %	11.8 %	3.3 %	1.5 %
January	77.2 %	4.1 %	12.4 %	3.2 %	1.6 %

Figur 3-5: Bruksstatistikk for nettlesere for Januar til april i 2018

3.11.3 SonarQube

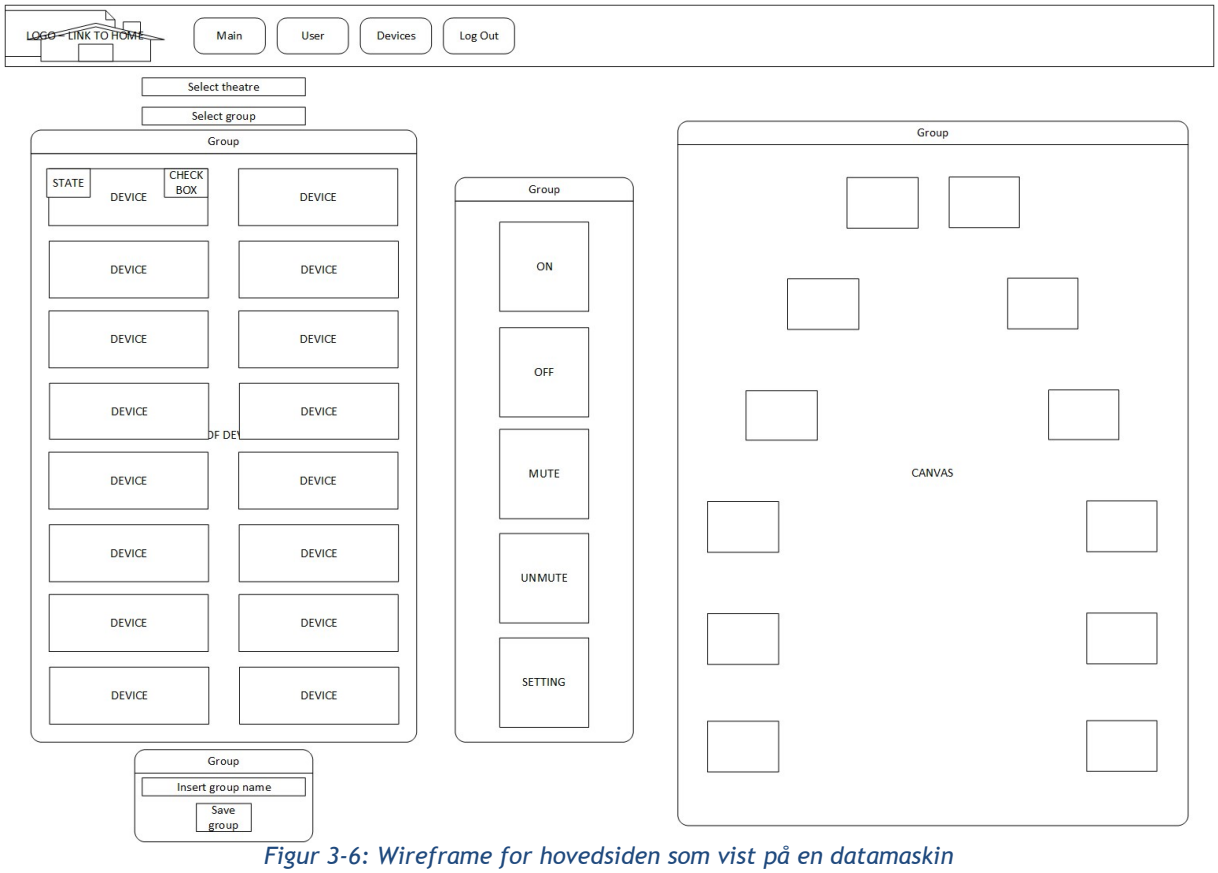
Vi tok i bruk SonarQube etter ønske fra oppdragsgiver. SonarQube ble mest brukt i startfasen av prosjektet, vi sjekket rapporten fra SonarQube hver gang prosjektet ble bygd. Lengre ut i prosjektet og mot slutten har vi brukt SonarQube svært lite, dette kommer av bl.a. at vi har hatt en del problemer med Jenkins. Vi har også hatt lite å vinne på å bruke SonarQube da vi stort sett har hatt få merknader. Mye av merknadene vi fikk var midlertidig kode for testing, debugging og sjekking. Et godt eksempel er «System.out.println(STRING)» metoden, denne fyrer av adversler i SonarQube.

3.12 Design

Da vi begynt å designe GUIen prioriterte vi først hvilke sider som var mest kritisk for at hoved funksjonalitet skulle fungere, vi kom fram til følgende rekkefølge:

- Main
- Device settings
- Device list
- Login
- Admin

Vi prioriterte Admin under Login fordi ordentlig admin-funksjonalitet krever en metode å identifisere brukeren på.
Vi fortsatte med å utarbeide noen wireframes slik at vi hadde et overblikk over hvordan vi ville at sidene skulle se ut. Vi lagde en wireframe for hver side som inneholder egne unike grupperinger, og for hver skjermstørrelse det var nødvendig. Fra starten av var målet å lage en nettside som er enkelt skalerbar. Vi brukte de følgende wireframene:

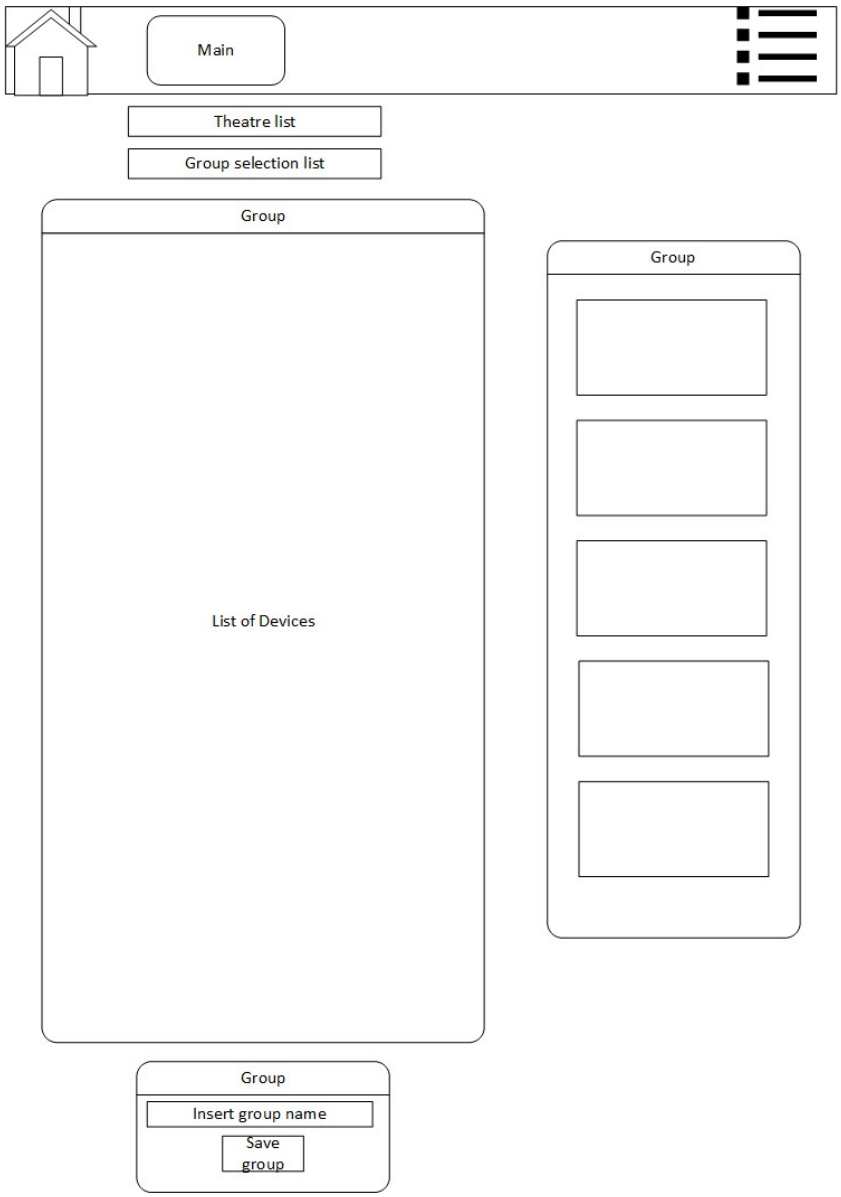


Figur 3-6: Wireframe for hovedsiden som vist på en datamaskin

På Figur 3-6 er det synlig at vi har tatt hensyn til flere gestalt-prinsipper i hvordan vi har kartlagt og tildelt plass til de ulike elementene over siden. Da vi utarbeidet hvordan vi ville ha siden kartlagt, passet vi på at elementene som hørte sammen ble lagt i nærheten av hverandre, som følger gestalt-prinsippene, 'proximity' og 'similarity'.

Siden er organisert i ulike grupper, som kan bli inndelt i tre hovedgrupper; enhetsliste, funksjonsliste og en visuell fremvisning, kjent som canvas. Enhetslisten inneholder teater-seleksjon, en gruppe-seleksjon, en visuell visning av det valgte teateret med gruppen, og muligheten til å lagre en gruppe. Funksjonslisten inneholder knapper som utfører funksjoner på de valgte listeelementene, funksjonene er: 'on', 'off', 'mute' og 'unmute'. Den visuelle fremvisningen viser hvor enhetene ligger i den fysiske verden.

Siden vil også skaleres nedover etter skjermstørrelse og når skjermen er en mobilstørrelse, så vil den visuelle fremvisningen av de fysiske posisjonene til enhetene forsvinne, som vist i Figur 3-7.



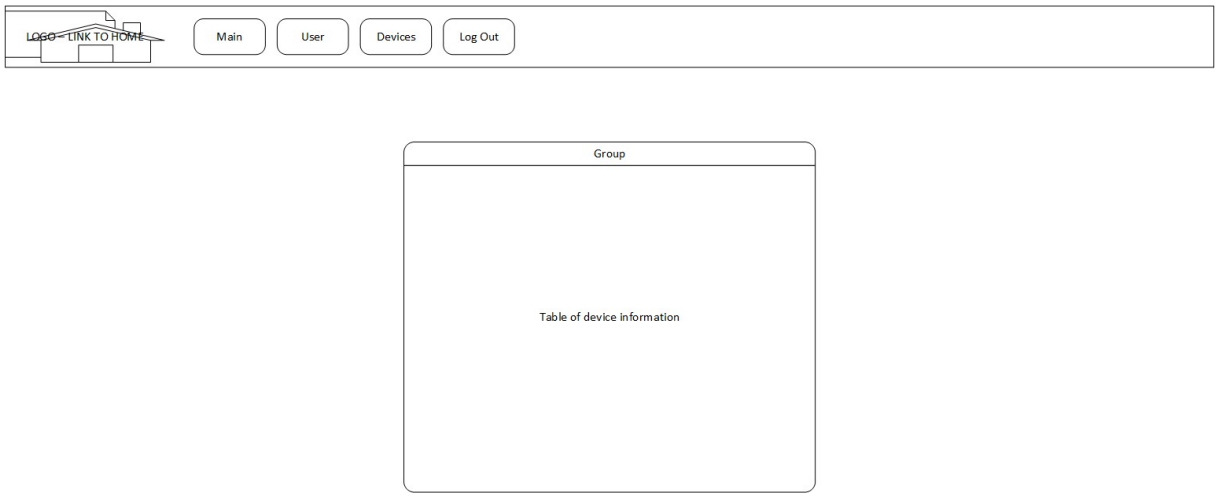
Figur 3-7: Wireframe for hovedsiden som vist på en mobil enhet

Navigasjonsmenyen som er synlig øverst finnes på alle sidene bortsett fra logg-inn siden. Navigasjonsmenyen inneholder lenker til hver egne side, og muligheten til å logge ut. Vi har delt nettstedet inn i de følgende sidene:

- Main
- Devices
- Admin
- Device settings page

Devices-siden er en side som skal visualisere relevant informasjon for brukeren. Måten vi valgte for å visualisere informasjonen var en tabell for hver enhet i databasen. Tabellene er basert på de større og mer detaljerte tabellene fra de individuelle enhets-sidene. De individuelle enhets-sidene inneholder mer detaljert informasjon i tillegg til muligheten til å utføre mer spesifikke forandringer på enhetene, slik som kontrast eller lysstyrke. Wireframe for detalj-sidene kan sees i Figur 3-8.

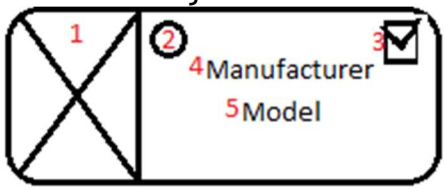
Administrator siden skal gi en administrator mulighet til å legge til, fjerne og oppdatere informasjon på enheter på en grei måte. Dette er gjort med HTML Forms. En administrator har mulighet til å administrere teater, enheter, grupper og brukere.



Figur 3-8: Wireframe for informasjonsside for en individuell enhet

For fargevalg på siden valgte vi med hensikt en mørkere farge slik at det ble mer komfortabelt for brukeren å bruke siden i mørkere rom. Som farge på de interaktive elementene brukte vi en mørk blå for navigasjonsmenyen og listen. For å komplementere blå-fargen valgte vi å bruke en litt mørkere grå-farge som bakgrunn.
Vi bruker også farger for å signalisere en enhet sin status.
For liste-elementene trengte vi noe som passet med det allerede eksisterende farge temaet og noe som representerte hva enhet liste-elementet skulle framstille.
Vi kom fram til at elementet trengte i vise følgende informasjon:

- Enhetstype
- Produsent
- Modell
- Status
- Seleksjon



Figur 3-9: Wireframe for listeelementet til en enhet

Hver enhet i listen skal bli representert av en boks som vist i Figur 3-9.
1 - Et ikon som viser hvilken type enhet det er (Projektor, lyd-system)
2 - Status-ikon som viser hvilken tilstand enhet er i, vil være avhengig av enheten, men alle vil ha standard-tilstandene (Av, på, utilgjengelig)
3 - Sjekkboks, vil vise om enheten er valgt eller ikke, bare valgte enheter vil kunne ha ulike funksjoner utført på seg.
4 - Viser produsenten til enheten (F.eks. Barco).
5 - Viser modellen til enheten (F.eks. F22).

3.12.1 Problemer med GUI-Design

Under utvikling ble det noen problemer med lister, listeelement og skalering av skjermen. Når skjermen blir mindre så vil det påvirke liste elementene med en relativ størrelse slik at oppførselen til listen blir uforutsigbar. Problemet ble løst med å finne en størrelse som ser bra ut på alle skjermstørrelsene.
Canvaset får ikke plass på mobile skjermer og mindre tablets i samme rad. Problemet ble løst med å fjerne canvaset fra de skjermene.
Knapper med multifunksjonalitet (en knapp som styrer av/på). Det viste seg å bli et stort problem å programmere oppførselen til knapper kombinert med en sjekkliste. Vi løste problemet med å splitte knappene slik at oppførselen ble opp til brukeren.

3.13 Dokumentasjon

3.13.1 JIRA

Først i prosjektet brukte vi litt tid på å lage brukerhistorier (user stories) og oppgaver(tasks) og prioriterte disse etter relevans for å fullføre prosjektet. For hver iterasjon (14 dager) i ettertid har vi gått gjennom og oppdatert og fylt etter disse brukerhistoriene og oppgavene etter behov.

3.13.2 Confluence

Confluence er en wiki-side. Der kan du lage sider og under sider, med bilder, lenker og tekst. Confluence kan kobles opp mot JIRA for å generer rapporter på framgang, sprint retrospektives mm. Vi brukte Confluence som felles skrive område når det kom til informasjon om elementer vi skulle bruke, og funn gjennom egne prøvelser.

3.13.3 Zotero

En stor andel av kildene vi har benyttet oss av kommer fra forskjellige nettsteder. Zotero har gjort jobben med kildehenvisning mye lettere fordi, i tillegg til hovedapplikasjonen, har Zotero en plugin til Google Chrome og en plugin til Microsoft Word.
Når vi så skal legge til en ny kilde fra et nettsted, aktiverer vi Google Chrome pluginen til Zotero mens vi er inne på det riktige nettstedet, Zotero henter så ut og lagrer informasjon som publiseringsdato, forfatter, tittel, mm. Etterpå må vi navigere til riktig sted i teksten i Microsoft Word og velge fanen «Zotero», under denne fanen trykker vi på «Add/Edit Citation» som åpner opp et søkefelt hvor vi kan søke etter kilden vi ønsker å legge inn.
Zotero håndterer også ting som automatisk sortering av referanseliste og automatisk oppsett og utbytte av valgt kildehenvisnigsstil.

3.13.4 Google Docs

Google Docs er en gratis kontorpakke tilbudt av Google som tilbyr tjenester som tekstbehandling, presentasjon og regneark. Det som skiller Google docs fra andre lignende programmer er at det er webbasert slik at brukerne kan dele et dokument og ha flere personer som redigerer samme dokument over web. Google Docs sin tekstbehandling ble brukt for å skrive rapporten i vårt tilfelle.

3.13.5 Microsoft Word

Det var ikke før i siste halvdel av prosjektet at vi tok i bruk Microsoft Word. Word lar flere brukere redigere et dokument, på linje med Google Docs. Fordelen med Word over Google Docs er struktur og rettskriving, ettersom det er mye mer utarbeidet enn i Google Docs.

4 Resultater

4.1 Systemarkitektur

4.1.1 Prosjektstruktur

Prosjektet er delt inn i 4 underprosjekter, VislabDevices, VislabController, BarcoF22Controller og DenonDN500AV.

4.1.1.1 VislabDevices

VislabDevices er kjernen til hele prosjektet. Det er et bibliotek som holder på «*Device*» og «*Projector*» grensesnittene, og henter opp implementerende klasser. Den holder på alle objekt som blir lagd på denne måten, og tillater at implementerende klasser kan hente og se disse objektene.

4.1.1.2 VislabController

Dette er nettsiden og overordnet handtering av moduler. Den har VislabDevices som avhengighet for å kunne kalle metoder som er tilgjengelig gjennom grensesnittene. Bygd på Spring rammeverket så setter den opp en webserver, starter sikkerhet og ordner linking og styling på hele nettsiden. Den håndterer grupper og flere enhets- typer samtidig gjennom å hente opp alle utvidelser som implementerer grensesnittene.

4.1.2 Moduler

Modulene til Vislab. Plattformen er laget med tanke på å kunne bli startet flere ganger uten å kunne påvirke hverandre. Det er for å kunne håndtere så mange enheter som ønskelig.

4.1.2.1 BarcoF22Controller

Utvidelse til VislabController, BarcoF22Controller er en selvstendig utvidelse som implementerer «*Projector*» grensesnittet. Den har og flere metoder som er spesifikk til Barco F22 projektorer. Metodene er tilgjengelig gjennom egen html grensesnitt. Gjennom metoder til VislabDevices kan en utvidelse vise frem sin egen HTML som kan gi utvidet kontroll til bruker.
Støttede funksjoner er:

- Testbilde
- Kontrast
- Kjøretid
- Lysstyrke
- Lampe kjøretid
- Lampe kjøretid igjen
- På/Av
- Temperatur
- Lampe status
- På/Av status

4.1.2.2 DenonDN500-AV

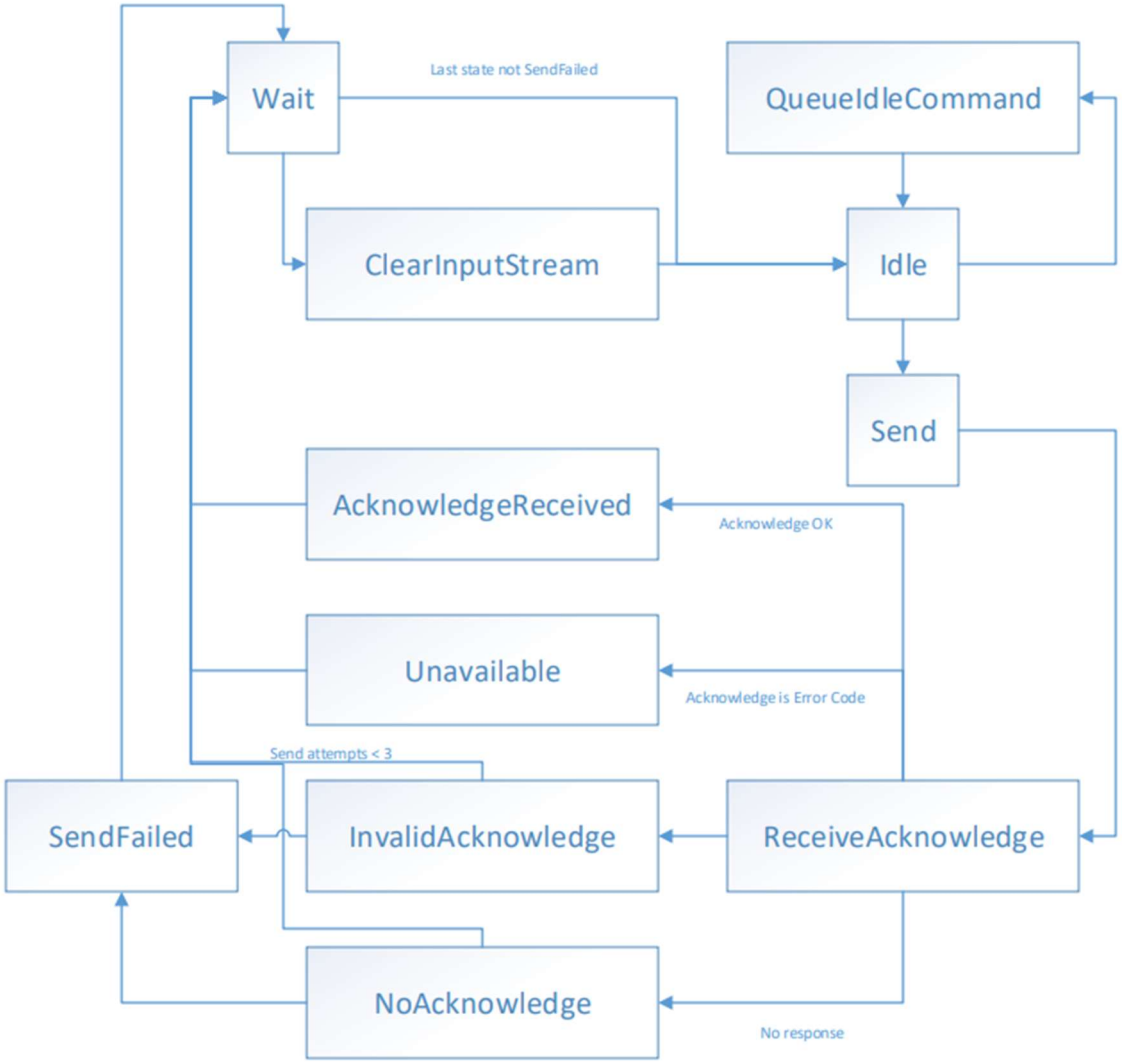
Utvidelse til VislabController, DenonDN500-AV er en selvstendig utvidelse som implementerer «*Projector*» grensesnittet. Den har og flere metoder som er spesifikk til DN500-AV. Metodene er tilgjengelig gjennom egen html grensesnitt. Gjennom metoder til VislabDevices kan en utvidelse vise frem sin egen HTML som kan gi utvidet kontroll til bruker.
Støttede funksjoner er:

- Volum
- På/Av
- Kilde
- Demp

4.1.3 State Pattern

I prosjektet blir State Pattern (beskrevet i kapittel 2.1) implementert to ganger. En i Barco F22 modulen, og en i Denon DN500-AV modulen. Struktur og størrelse er forskjellig mellom modulene.

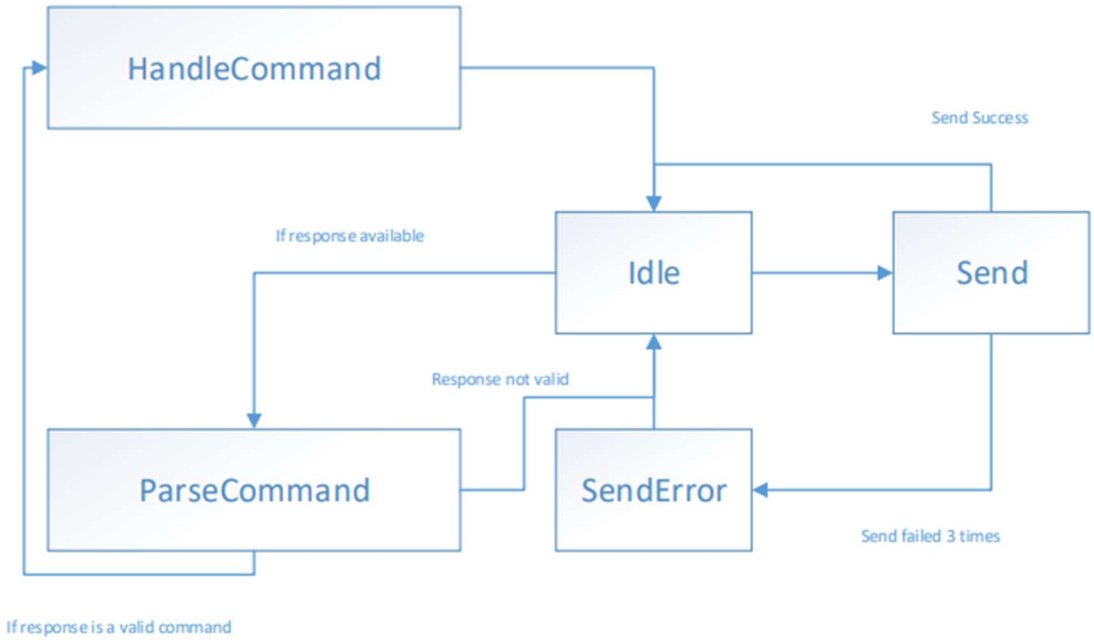
4.1.3.1 Barco F22 state pattern



Figur 4-1: Barco F22 tilstands diagram

State patternet til Barco F22 modulen inneholder 11 foskjellige tilstander, som vist i Figur 4-1. Hele systemet fungerer sekvensielt. Den går i samme sirkel hele tiden uten avvik.
Rekke følgen er: «Idle>Send>ReceiveAcknowledge>handering av svar>Wait>Idle».
Det er kun når et avvik skjer at andre elementer blir satt inn i mellom disse stegene, men den fortsetter videre.

4.1.3.2 Denon DN-500AV State Pattern

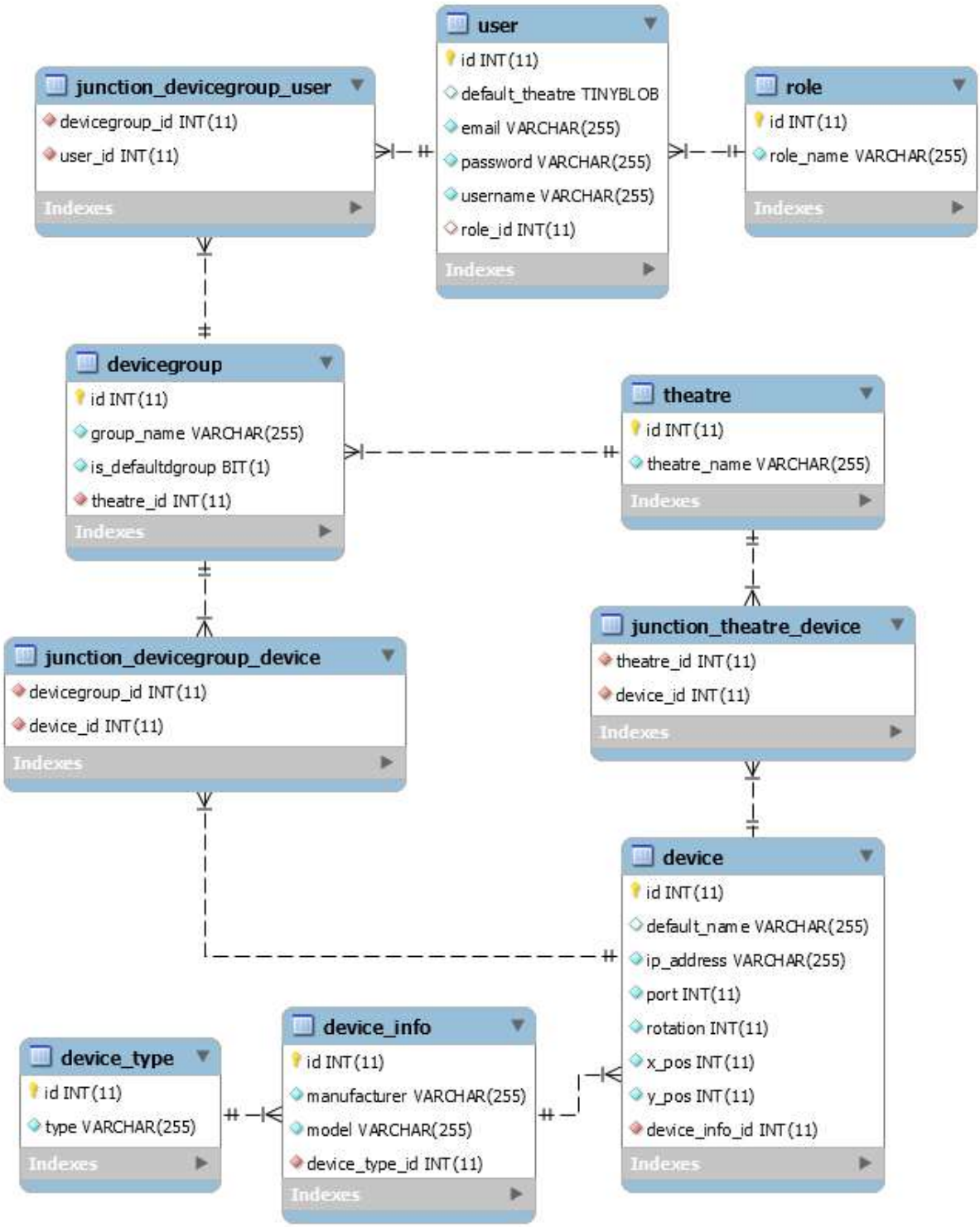


Figur 4-2: Denon DN500-AV tilstands diagram

State patternet til Denon DN500-AV modulen er mer komplekst selv om det kun inneholder fem forskjellige tilstander, som vist i Figur 4-2. Grunnen til dette er at du mottar meldinger fra enheten om eksterne endringer, noe som kan blokkere hele systemet om ikke handtert. Dette skjer ved at den leser alt som har blitt mottatt mens den var i andre tilstander før den sender nye meldinger. Når ei melding er en som vi har støtte for (se kapittel 4.1.3.2) blir feltene oppdatert slik at en bruker alltid får nyeste informasjon.

4.2 Database

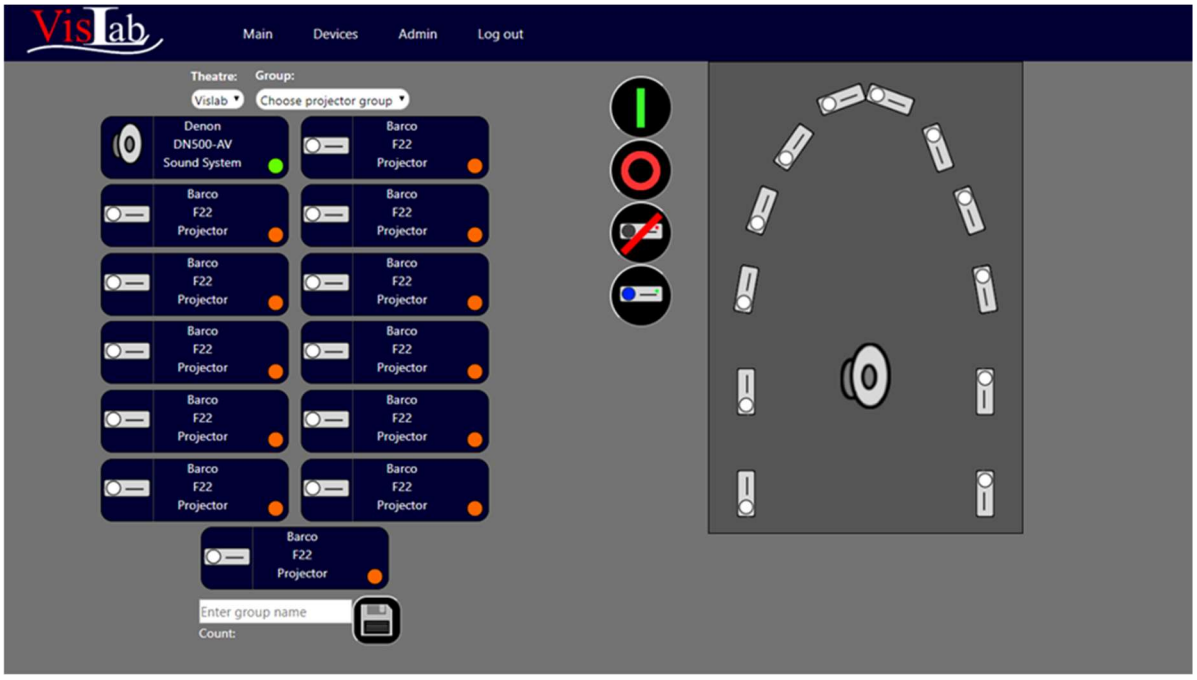
Databasestrukturen vår er som vist i Figur 4-3. Den er lagd ved hjelp av Spring rammeverket som forklart i kapittel 3.5.4. Spring har bygd tabellene så kompakte som mulig ved hjelp av kombinerte fremmednøkler i koblingstabellene. Vi har også designet entitetene for å spare plass ved å skille ut roller, enhetsinfo og enhetstyper i hver sine egne tabeller; role, device_info, og device_type.



Figur 4-3: Entitetsdiagram

4.3 GUI - Brukergrensesnitt

Main



Figur 4-4: Den endelige hovedsiden

Main-siden består av flere seksjoner som Figur 4-4 viser. En navigasjonsmeny øverst, som lar brukeren bestemme hvilken side som er oppe, og selve main-siden. Main-siden består av to seksjoner en liste over enhetene i det valgte teateret, og ett canvas som tegner opp den fysiske posisjonen til de ulike enhetene, relativ til hverandre.
Listen kan og bli delt opp i flere seksjoner, selve listen og listen sine kontroller-funksjoner. Kontroller-funksjonene til listen er som følger:

- to drop-down lister
  - Teater
  - Gruppe-seleksjon
- Gruppelagring
- fire knapper med funksjoner
  - På
  - Av
  - Mute
  - Unmute

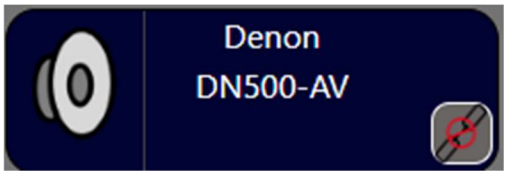
Teateret styrer hvilke enheter som vises i listen, gruppene er forhåndsbestemte seleksjoner av enhetene i det valgte teateret. Gruppelagring tar de valgte enhetene og lagrer de som en gruppe, i teateret. De fire knappene, av, på, mute og unmute utfører funksjonen sin på de valgte enhetene.

Hver enhet i listen blir representert av en boks som viser:

- Enheten sin type (Projektor, lyd …)
- Status (Av, på, utilgjengelig, etc. …)
- Produsent
- Modell

Se Figur 4-5 og Figur 4-6.

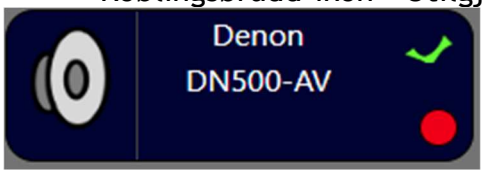
Figur 4-5: Listeelement for en enhet av type Barco F22



Figur 4-6: Listeelement for en enhet av type Denon DN500-AV

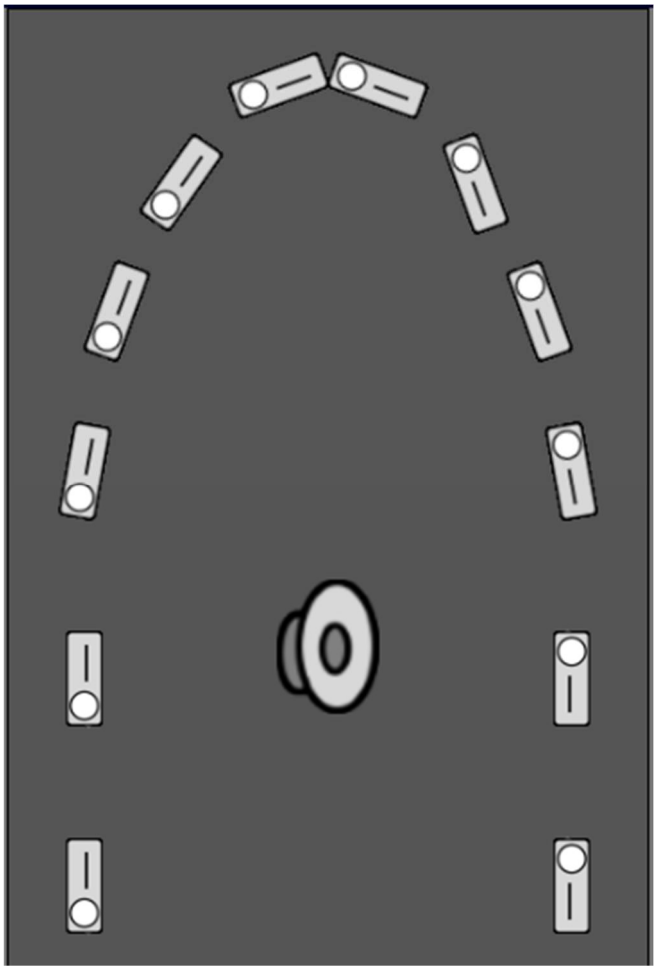
Enheten sin type blir representert som et ikon av dens tilsvarende enhetstype, se Figur 4-5 og Figur 4-6.
Enheten sin status har åtte mulige tilstander:

- Oransje - Deep sleep
- Rød - Av
- Oransje - Skrur på
- Grønn - På
- Gul - Standby
- Mørk Rød - Skrur av
- Svart - Kritisk feil
- Koblingsbrudd-ikon – Utilgjengelig



Figur 4-7: Et avmerket listeelement

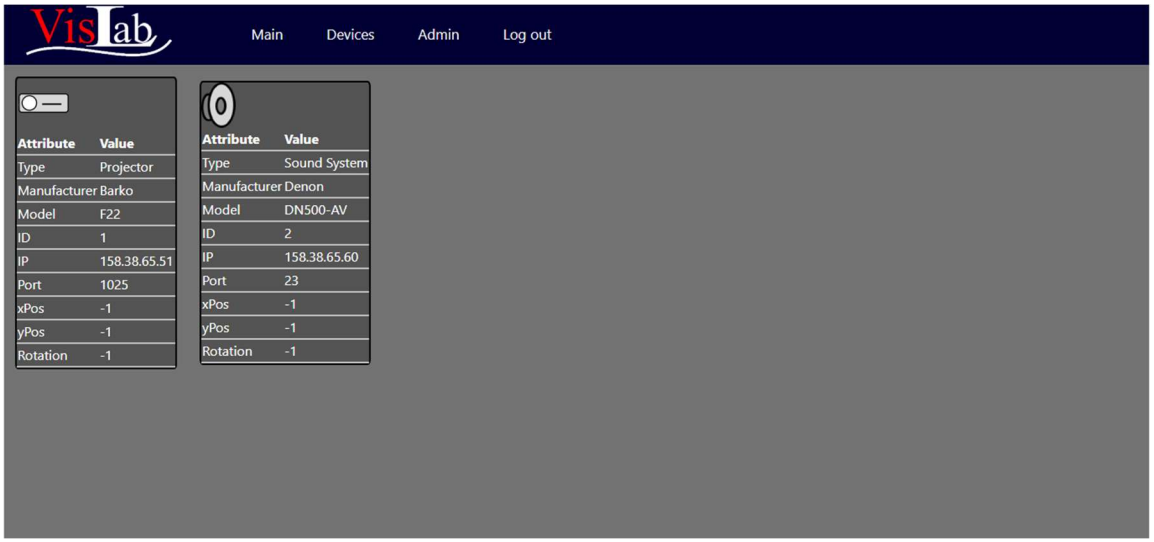
Om en enhet blir valgt blir dette vist som en hake, som i Figur 4-7.



Figur 4-8: Grafisk visning av enhetene

«Canvas» elementet er et rutenett som mottar en x-posisjon, en y-posisjon og en rotasjon i grader, se Figur 4-8. Koordinatene blir hentet ut av databasen som blir lagret med enheten. Disse enheten kan trykkes på for å åpne innstilling siden til den spesifikke enheten.

Device List og Device Page
Device list siden består av navigasjonsmenyen, som er felles for alle sidene bak innlogging, og hoved-seksjonen som inneholder en tabell for hver enhet som finnes i databasen.



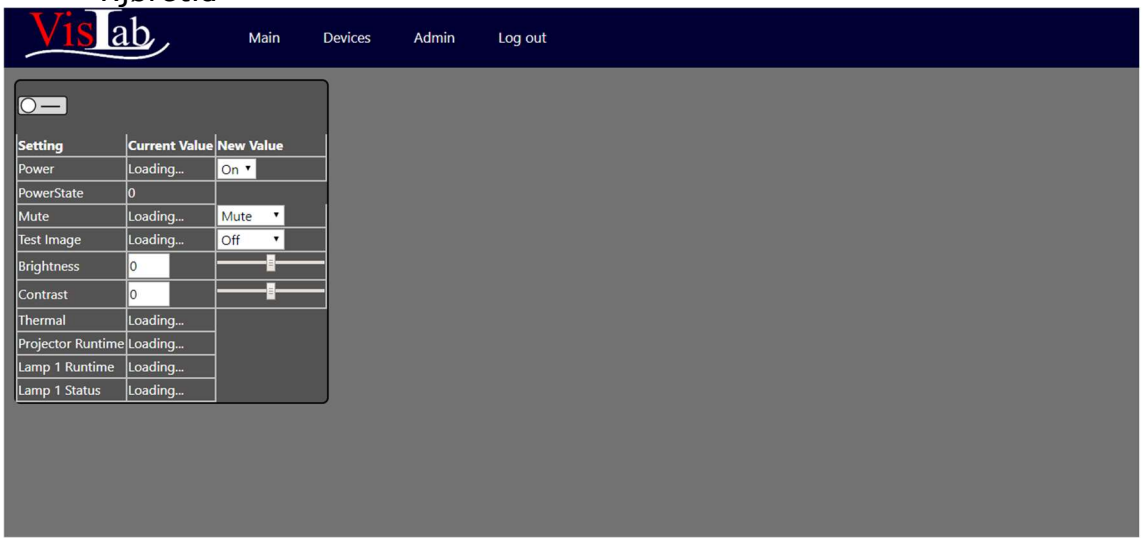
Figur 4-9: Oversikt over alle enhetene i databasen

Tabellen som vist i Figur 4-9 inneholder følgende informasjon over en enhet:

- IP-Adresse
- Port
- Enhetstype
- Produsent
- Modell
- X-Posisjon
- Y-Posisjon
- Rotasjon

Hver tabell inneholder en lenke til enheten sin egne side som inneholder en tabell som viser for projektorer, som vist i Figur 4-10:

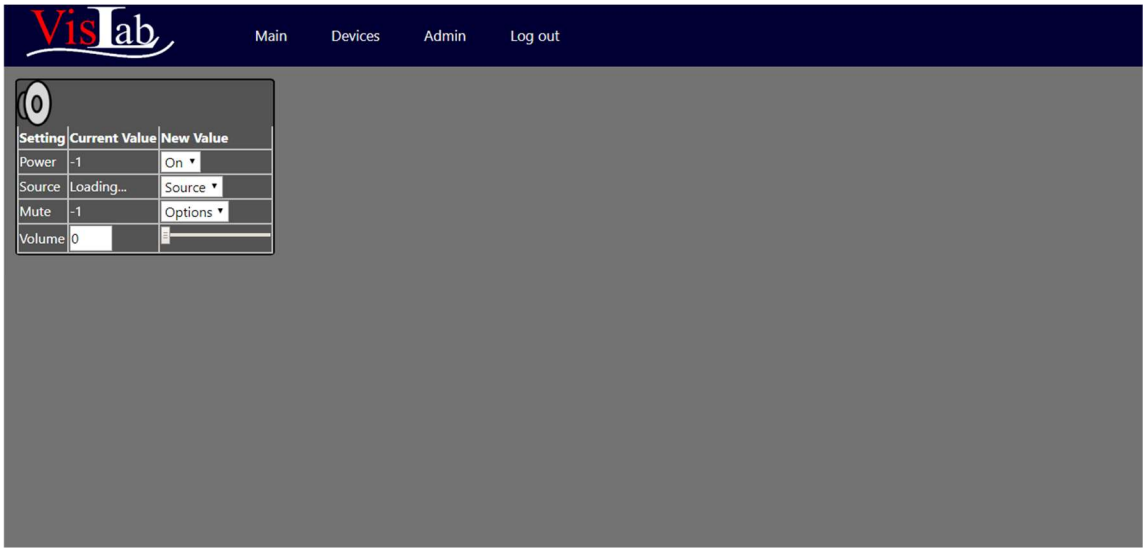
- Av/på status
- Mute/unmute status
- Testbilde status
- Kontrast
- Lysstyrke
- Lampestatus
- Kjøretid



Figur 4-10: Oversiktsside for en enkelt enhet av type Barco F22

For lydanlegg (se Figur 4-11):

- Av/på status
- Mute/unmute status
- Input kilde
- Volum



Figur 4-11: Oversiktsside for en enkelt enhet av type Denon DN500-AV

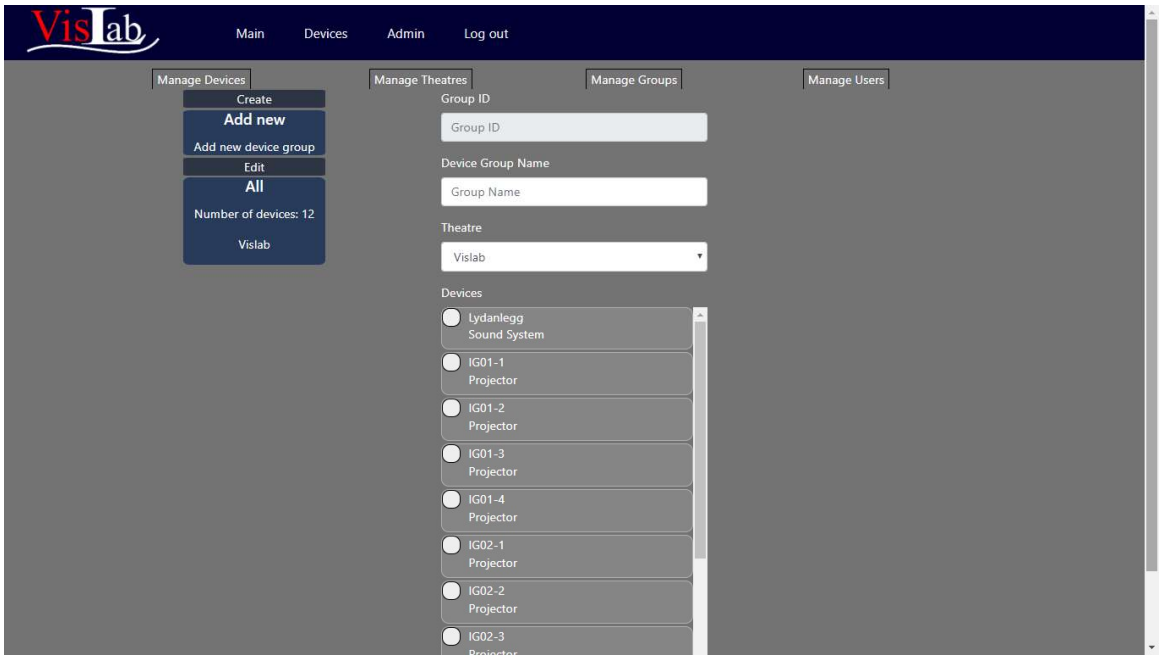
I begge tabellene er det mulig å endre noen verdier, i projektor tabellene er det mulig å endre (se Figur 4-10): av/på, mute/unmute, testbilde, kontrast og lysstyrke. I lydanlegg tabellen er det mulig å endre(se Figur 4-11): av/på, mute/unmute, input kilde og volum.

Admin

Admin siden inneholder flere funksjoner låst bak en rollesjekk for brukeren, det vil si at brukeren kun får tilgang til siden og dens funksjoner om kontoen de bruker har admin-rollen. Funksjonene som er reservert for adminbrukere er:

- Mulighet til å legge til, konfigurere eller fjerne en enhet.
- Mulighet til å legge til, konfigurere eller fjerne et teater.
- Mulighet til å administere grupper.
- Mulighet til å administrere brukere.

Admin siden (se Figur 4-12) består av fire kolonner, hver kolonne vil bli gjemt om de er inaktive. Hver kolonne inneholder funksjonsknapper og tekstfelt for å enten legge til eller endre feltene i enheter.



Figur 4-12: Siden for å administrere enhetsgrupperinger

Ikoner

Som visuelle hjelpemidler for brukeren lagde vi en del ikoner for å representere forskjellige enheter, tilstander og funksjoner. Figurene 4-13 til 4-21 forklarer disse ikonene.

Figur 4-13: Avmerkingsikon

Figur 4-14: Projektorikon

Figur 4-15: Lydanleggikon

Figur 4-16: På-ikon

Figur 4-17: Av-ikon

Figur 4-18: Bilde på

Figur 4-19: Bilde av

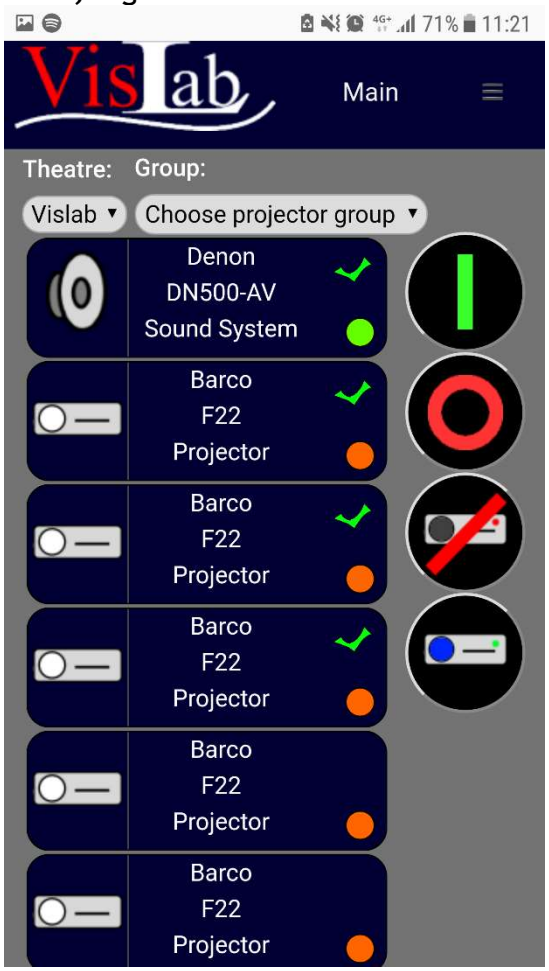
Figur 4-20: Koblingsbrudd

Figur 4-21: Lagre

4.4 Responsiv design

For å lage en responsiv nettside trengte vi elementer på siden som krympet med skjermstørrelsen, eller fjerne noen når skjermen ble mindre. For hele nettstedet har vi brukt en kombinasjon av disse alternativene:

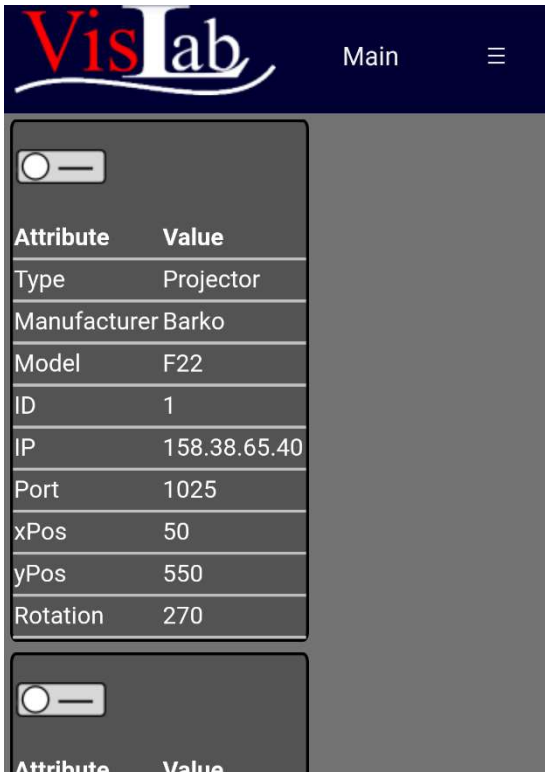
Main, Figur 4-22



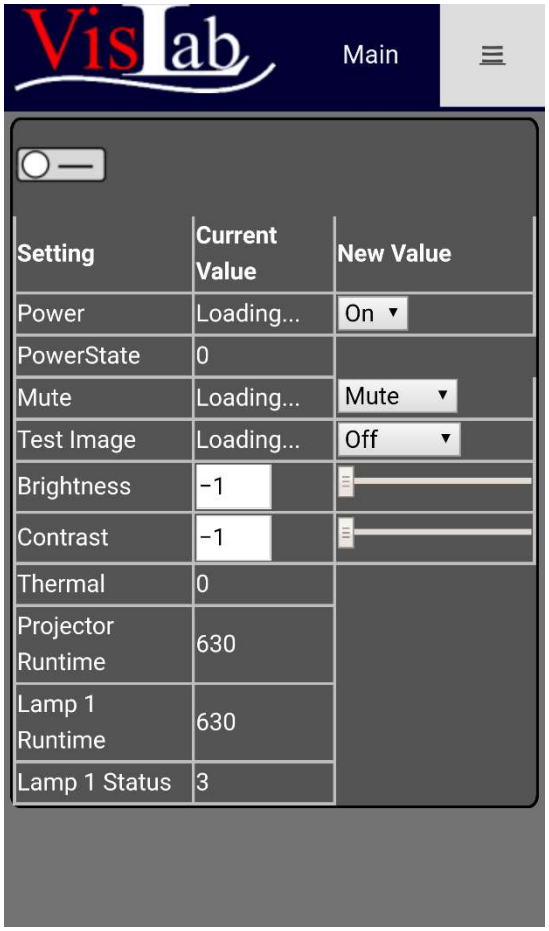
Figur 4-22: Hovedsiden som vist på en mobil enhet

Device List og Device Page, Figur 4-23 **og** Figur 4-24

Mange av sidene ble designet slik at minimum mengde med ekstra tanke trengte å bli lagt inn i andre skjermstørrelser. To av sidene får endringer i plasseringen av tabell(er).



Figur 4-23: Enhetslisten som vist på en mobil enhet



Figur 4-24: Oversiktssiden for en enkelt enhet som vist på en mobil enhet

4.5 JavaScript

JavaScript ble brukt til å programmere klient-side logikk og håndtering av data for brukeren. Vi brukte JS til å opprette noen HTML-elementer, implementere funksjoner som endrer på verdier i enhetene, opprette canvas funksjonalitet og hente data fra databasen. Da vi begynte på siden fant vi fort ut at vi trengte dynamiske HTML-elementer, og at en av de enkleste løsningene for å oppnå det var å bruke JavaScript. Vi valgte JavaScript fordi det er et godt etablert programmeringsspråk med en stor mengde dokumentasjon på internett, slik at vi mest sannsynlig aldri støtte på noen unike problemer.

```
function addListElements(device) {
    let ul = document.getElementById("selected-list");
    let imageCard;
    let name = device.type.toLowerCase().replace(" ", "_");
    imageCard = "<image src='Images/" + name + ".png' alt='" + name +"-image'
class='list-icon'></image>";

    var li = document.createElement('li');

    let counter = devices.length + 1;
    li.innerHTML =
        "<div class='image-card' style='cursor: pointer' onclick='moveToDevicePage(\""
+ device.id + "\")'>"
        + imageCard
        + "</div>"
        + "<input id='pro" + counter + "'" + " class='pro-checkbox' type='checkbox'
onclick='updateCount()'>"
        + "<label for='pro" + counter + "'" + " class='check-label' onclick='update-
Count()'></label>"
        + "<label for='pro" + counter + "'" + " class='text-label' onclick='update-
Count()'>" + device.manufacturer + "<br>" + device.model
        + "<br>" + device.type + "</label>"
        + "<span class='state-icon'></span>"
        + "<span class='disconnect-icon'>"
        + "<image src='../Images/disconnect.png' alt='disconnected' class='disconnect-
image'></image></span>";
    ul.appendChild(li);
    devices.push(new DeviceMap(device, li, document.getElementById("pro" + counter)));

    document.getElementsByClassName('pro-checkbox');
}
```

Figur 4-25: Eksempel på dynamisk listeelement

Vi trengte også noen funksjoner som kjørte i bakgrunnen kontinuerlig som oppdaterte statusen til enhetene slik at vi hadde tilgjengelig hvilke enheter som var på, av eller avkoblet. For å oppnå dette brukte vi en kombinasjon av JS sin timeout metode, og rekursiv-programmering.

```
function updateState() {
    for (let n = 0; n < devices.length; n++) {
        getPowerState(devices[n].id, n);
    }
    timeout = setTimeout(updateState, 2000);
}
```

Figur 4-26: Eksemepel på en enkel worker

All håndtering av data blir utført med hjelp av JS sin fetch-metode, som er en 'promise'. Det betyr at funksjonen skjer asynkront og vil returnere et promise i stedet for en faktisk verdi, som betyr at vi må bruke "promise-chaining" med hjelp av then-metoden for å kunne få inn data hvor vi vil ha den. Under kan vi se at når siden starter vil getTheatres()-metoden bli kalt, som får et JSON-objekt som svar, som gjøres om til en liste og sendes videre setUp()-metoden som starter opp resten av siden.

```
getTheatres().then(r => setUp(r));
```

Figur 4-27: Eksempel på en promise

JavaScript ble også brukt for å bygge simple metoder som slår enheten av/på, et eksempel på en slik metode kan bli sett under. Disse metodene benytter også fetch-metoden til JS, som betyr at de blir utført asynkront.

```
function powerOn() {
    for (let i = 0; i < devices.length; i++) {
        if (devices[i].checkbox.checked) {
            let pID = devices[i].id;
            fetch('api/main/powerOn?id=' + pID).then(response => {
                if (response.ok) {
                    response.json().then(p => console.log(p));
                }
            });
        }
    }
}
```

Figur 4-28: Eksempel på en enkel metode

4.6 API

Når applikasjonen kjører er det mulig å kommunisere med den ved hjelp av et sett med API-kall som sendes over HTTP, helst i en nettleser. Hele API-treet er dokumentert i vedlegg F.
Nettsiden i seg selv er tilgjengelig under følgende kall (så snart man har logget inn):
../ Dette er hovedsiden, dersom du ikke er logget inn blir du videresendt til **../login**
../admin Dette er administratorpanelet, her kan en innlogget administrator opprette, oppdatere og slette brukere, enheter, enhetsgrupper og teater
../login Dette er innloggingssiden
../device?id=<enhetsid> Denne siden gir detaljert status om en enkel enhet, her kan man også oppdatere parameterne som det er støtte for
../devicesettings Denne siden gir en oversikt over alle enhetene som ligger i databasen.

Resten av funksjonaliteten ligger under følgende kall:
../api/main gir grunnleggende funksjonalitet til alle enhetstyper
../api/BarcoF22 gir all funksjonalitet til enheter av type Barco F22
../api/DenonDN500AV gir all funksjonalitet til enheter av type Denon DN500-AV
../api/devicegroup gir funksjonalitet til å hente, oppdatere og fjerne enhetsgrupperinger
../api/theatre gir funksjonalitet til å hente, oppdatere og fjerne teatergrupperinger
../api/device gir funksjonalitet til å hente, oppdatere og fjerne enheter
../api/user gir funksjonalitet til å hente, oppdatere og fjerne brukerkontoer

4.7 Kjente feil eller mangler

Da vi begynte å nærme oss slutten la vi fort merke til at vi ikke hadde tid til å implementere eller fikse absolutt alt og vi måtte derfor begynne å prioritere hva ville vi skulle bli ferdig implementert til slutt, og hva vi ønsket fungerte ordentlig med minimum mengde feil.
Bruker kan ikke endre IP addresse til en enhet viss enheten er tilkoblet allerede. Verdien blir oppdatert i alle felt, men gammel tilkobling blir ikke brutt. Krever restart av applikasjon for å fikse.
Drivere stopper ikke skikkelig etter å bli fjernet fra siden. Krever restart av applikasjon for å fikse.
Under oppstart av java programmet gjennom Docker-compose kan det komme exceptions, dette er fordi starting av Java kontaineren ikke venter på at SQL kontaineren er klar, den venter bare på at den har startet.

4.7.1 Login siden

På grunn av tid, og bugs med sikkerhet så ville ikke bootstrap laste inn ordentlig for firefox, vi valgte derfor å ta bort bootstrap fra login siden som en liten fix. Designet på login siden er derfor veldig simpelt og ikke helt optimalt.

4.7.2 Design

På grunn av mangel på artistisk evne på gruppen ble designet på nettsiden veldig simpelt og ikke det aller fineste enkelte plasser.

4.7.3 Manglende implementasjoner

Der er en liten liste over ting som vi ønsket å få implementert:

- Full logging av feil som oppstår til egen fil.
- SSL i alle kommunikasjonslag. Der er ikke SSL fra applikasjonen og ned til databasen.
- Bedre valg av posisjon i Administrator-siden. Der er bare et tekstfelt som godtar tall, uten å vise hva tallet betyr. En administrator må dermed prøve å feile for å få en følelse av hvor mye er nok. Det naturlige ville ha være å vise frem samme visualisering som er på hovedsiden og vise endringene der.
- Lite eller ikke beskrivende feilmeldinger/koder til bruker.
- Ingen egen feilmeldings side, bruker ser Spring Boot sin default error side.
- Det ligger database struktur klar til å implementere standard teater for hver enkelt bruker, men tidspress gjorde at implementasjon på applikasjon og nett nivå ikke ble mulig.
- Drivere stopper ikke av seg selv. Disse burde stoppe og bryte tilkobling etter en lengre periode uten å være i bruk.
- Støtte for bytte av database type. En av styrkene til Spring er at den abstraherer bort hvilken database det er snakk om. Vi tar ikke nytte av dette med vårt oppsett. Vi støtter kun MySQL som database.
- Ingen/lite javaDoc i VislabDevices, BarcoF22Controller og Denon DN500-AV

5 Drøfting

5.1 Evalueringer

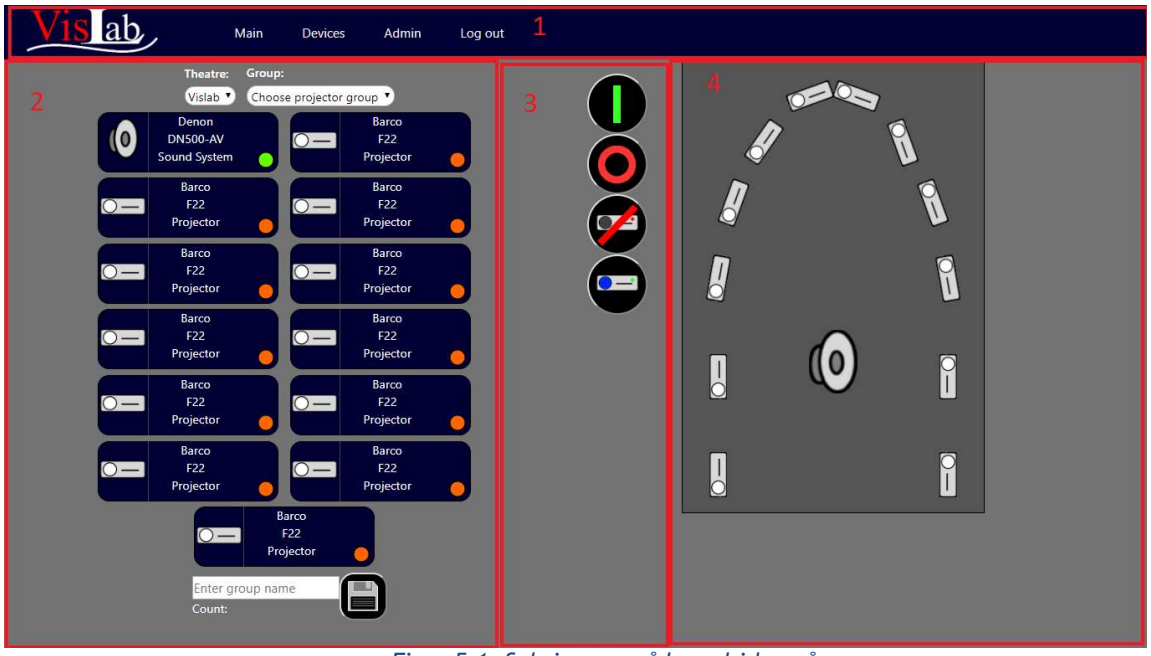
5.1.1 Resultatet

Resultatet er funksjonelt. Bruker kan slå på og av enheter, legge til enheter til siden, organisere disse inn i teater og lage grupper inne i teatrene for å slå på deler eller hele teateret med så få tastetrykk som mulig. Det er også mulig å utvide selve applikasjonen med flere enheter.

5.1.1.1 Design

Planen med brukergrensesnittet var å utforme noe som var simpelt, lett forståelig og funksjonelt. Det var aldri meningen at brukergrensesnittet skulle være noe nytt og innovativt, men heller brukervennlig. Utifra malen vi viste tidligere (kapittel 3.12) utviklet vi selve grunn-idèen bak utseendet veldig lite. Vi prøve å holde samme tema gjennom hele grensesnittet, vi har derfor utviklet en grunnside eller template som blir satt inn på alle sidene. Grunnsiden inneholder fargevalget, navigasjonsmenyen, og alle de viktigste stilvalgene.
Da vi valgte fargene til grensesnittet tok vi hensyn til at applikasjonen blir mest sannsynlig brukt i et dårlig belyst rom og[24] valgte derfor mørkere farger slik at det er enklere på øynene til brukeren.
Når det kommer til selve utseendet brukte vi en del egen CSS, for å unngå det klassiske Bootstrap utseendet som rammeverket bruker. Bootstrap ble ellers brukt for å posisjonere og plassere elementer.
Ikonene som blir brukt i grensesnittet er egenlagde ikoner, men de tar etter allerede eksisterende standarder, slik som en strek for på, en sirkel for av, en diskett for lagring og videre. For knappene på hovedsiden prøvde vi å designe noen knapper som kunne assosieres med knapper på enheter slik at det hjalp til å fremheve at grensesnittet er en kontroller for framvisningsenheter.
Selv synes vi at grensesnittet ble veldig enkelt og funksjonelt, men ikke spesielt pent. Annet enn det er vi fornøyd fordi grensesnittet utfører jobben den er designet for.

Gestaltprinsipp
For posisjonering og plassering av individuelle elementer brukte vi noen gestaltprinsipper for å øke forutsigbarheten til grensesnittet. Vi tok spesielt nytte av proximity og similariy for å organisere elementene.



Figur 5-1: Seksjonene på hovedsiden vår

En kan se at navigasjonsmenyen(1) så blir proximity-prinsippet brukt i det at de ligger nærme hverandre, som skal vise at knappene har lik funksjon i det at de er navigasjonsknapper.
Listevinduet(2) viser en dropdown-meny, en liste med enheter, og et tekstfelt med et lagringsikon som ligger i nærheten av hverandre. Proximity skal vise at de tre funksjonene jobber sammen.
Knappene(3) ligger tett sammen, og ser like ut fordi de har alle samme oppgave, utføre en eller annen funksjon på de valgte enhetene. Her blir både proximity og similarity tatt i bruk.

5.1.1.2 Brukervennlighet

Et av poengene med dette prosjektet var å utvikle en applikasjon som kunne bli brukt av alle på NTNU, inkludert de som ikke bruker en datamaskin som hovedverktøy for å utføre jobben sin. Det var et stort fokus for oss å utvikle noe som er enkelt forståelig for flest mulig. Det første brukeren møter etter login er enhetslisten, teater- og gruppevalg, lagring, funksjonsknappene og canvaset. Det krever allerede litt forkunnskap for å bruke siden, slik som hva et teater er i denne sammenhengen. Brukeren må trykke på hver enhet for å velge de slik at de kan bli interagert med. Dette kan bli litt tungvint etter hvert, men det er derfor brukeren kan lage sine egne grupper slik at det kun er nødvendig å gjøre én gang. Vi valgte også å lage brukergrensesnittet i Engelsk, grunnen til det er fordi NTNU har flerspråklige brukere.

Universell Utforming
Direktoratet for forvaltning og IKT (Difi) sier at IKT-løsninger i Norge skal være universelt utformet. Dette gjelder både private og offentlige virksomheter i Norge. For at reglene skal gjelde er det noen kriterier som må oppfylles, de er som følger:

- Rettet mot kunder eller innbyggere i Norge (allmenheten).
- Bruk til å informere eller tilby tjenester.
- En ny tjeneste, eller oppgradert etter juli 2014.

Vi faller under to av disse. Det vi utviklet faller under en intern løsning for NTNU ifølge Difi. Det vi utviklet er derfor ikke pålagt å følge regelverket, men det er fortsatt noen gode retningslinjer å følge for å gjøre den mest mulig tilgjengelig og brukervennlig for brukerne.

Fra kravspesifikasjonen trengte vi ikke å følge universell utforming. Vi valgte å følge reglene for universell utforming som retningslinjer, både fordi vi ville ende opp med noe som er brukervennlig og fordi vi så på det som en utfordring for oss selv. Difi har ti metoder som lar oss teste for universell utforming [66].

1 – Tastaturnavigering

Vi testet ut tab-navigering for nettstedet. Det funker overraskende bra med tanke på at det ikke egentlig var et fokus fordi applikasjonen var ment for bruk på desktop-maskiner og mobiltelefoner. Navigasjonen går, det vi mener, er riktig rekkefølge, begynner på logoen og går fra venstre til høyre ende før den hopper ned til neste nivå. Men selve hovedsiden kan ikke brukes med bare tastatur, fordi interaktiviteten og funksjonene i listelementene blir ikke funnet av tab-navigering. En annen negativ side er at selve markøren for tab-navigering ikke er veldig synlig, men fokus-vinduet, nede i venstre hjørne vil vise hvilket element den er på.

2 - Forstørring og responsiv design

Responsiv design var en av kravene for nettsiden og er derfor oppfylt.

3 – Farger og kontraster

Vi har passet på at tekst og viktige elementer har en farge som skiller den ut fra bakgrunnen den er på slik at selv med svart/hvitt, så burde det være synlig hva som er hva.

4 – Overskrifter

Vi har passet på å legge inn overskrifter hvor vi føler det er nødvendig og gir mening. Vi har tatt i bruk HTML sine overskrift elementer <h1> til <h6>.

5 – Lenker

Vi har brukt minimalt med lenker, og har ingen lenker på nettsiden som fører til et annet nettsted. Lenkene som er på siden er navigasjonslenker og er godt merket med at de enten ligger inne i bilder/ikoner eller på navigasjonsmenyen med andre lenker.

6 – Bilder

For bilder har vi tatt i bruk alt attributtet som HTML-bilder har. Alt-teksten er veldig kortfattet, men inneholder det essensielle som bildet/ikonet skal framstille.

7 – Skjema

Eneste skjema som er tilgjengelig på nettstedet er låst bak admin.

8 - Søkefunksjon

Nettstedet har ingen søkefunksjon.

9 – Sidetitler

Alle HTML-sidene har egen sidetittel.

10 – Kodevalidering

På HTML sidene har vi passet på at alt er nestet riktig, vi har ingen duplikat id-attributter, elementene blir startet og avsluttet riktig og elementer ikke har duplikat attributter. Da vi validerte HTML-koden klager den flere html-elementer, men klagene er fordi vi gjør veldig mye av side-byggingen med JavaScript, slik at innholdsrike sider ser ut som flere tomme HTML-elementer og et script-kall på bunnen. De egenlagde html-attributtene vi bruker utløser også valideringsfeil.

5.1.1.3 Scripts

Vi synes at vi implementere scriptene våre på en effektiv måte. Vi innså at vi kom til å ha noen metoder som var nødvendig på mer enn en side. Vi løste dette problemet med å implementere et "mainscript". Mainscriptet var scriptet som tok hånd om passe på at navigasjonsmenyen var tilstede hvor den skulle. Vi flyttet alle metoder som var nødvendig på flere sider til mainscriptet slik at de fikk tilgang til de metodene uten å ha ekstra redundans.

5.1.1.4 Applikasjon

Java-delen av applikasjonen oppfører seg som en server som tar imot kommandoer fra brukergrensesnittet. Grensesnittet sender en kommando, og som oftest så vil den kommandoen ta med seg noen parameter. Parameterne kan være alt fra en enkel enhets ID til en gruppe med enheter og et gruppenavn. Mye av logikken og konstruksjonen av objekter blir håndtert i JavaScript med JSON-format. Vi kunne ha programmert mye av denne logikken i Java om vi tok SpringMVC bedre i bruk.

5.1.1.5 Sikkerhet

Vi har implementert SSL inn i applikasjonen. Dette er kun på nettsiden og ikke til databasen. Cross-Site Request Forgery (CSRF) er vi ikke beskyttet mot, ettersom implementasjonen av SSL kom så sent, og javascriptet må oppdateres for å inkludere en CSRF-token.
Gjennom docker-compose er kommunikasjonen mellom database og applikasjon sikker, den skjer gjennom et lukket lokalt nettverk inne i docker.
Bruker passord blir kryptert når det lagres i databasen gjennom BCrypt, en krypterings algoritme som følger med i Spring Boot.

5.1.1.6 Database

I kravspesifikasjonen stod det ingenting om at vi trengte å ta i bruk en database, men den gjorde det veldig tydelig at vi trengte en form for persistent data. VI følte det ble mest naturlig å implementere en database for å holde på den persistente dataen, og valgte derfor en relasjon database fordi det var det vi var kjent med fra tidligere skoleprosjekter. Vi kunne også valgt ingen database og håndtere persistent data med enten filer eller XML-lagring. Det finnes andre typer databaser vi kunne valgt også, slik som en NoSQL database slik som MongoDB.

5.1.2 Rammeverk og verktøy

Under utføringen av prosjektet tok vi i bruk noen rammeverk for å gjøre noen oppgaver mer effektiv og enklere å utføre. Her vil vi diskutere hva som gikk bra eller dårlig med de brukte rammeverkene og hvor det kunne vært en fordel å bruke et rammeverk.

5.1.2.1 Bootstrap

Vi valgte å bruke Bootstrap som CSS-rammeverk i dette prosjektet. Vi begynte ikke med Bootstrap før en god del inn i utviklingen fordi vi trodde at all CSS-kodingen ikke kom til å bli så mye arbeid. Vi oppdaget at det var feil. Grunnen vil la merke til at vi trengte å gjøre CSS-kodingen litt enklere var på grunn av all layoutet til nettsidene. Vi la merke til at noe som var tilsynelatende enkel posisjonering, ikke var så enkel. Vi tok derfor i bruk Bootstrap for å bruke de ulike posisjoneringsklassene Bootstrap inneholder. Både generell posisjonering, slik som grid-systemet til Bootstrap, og enkel sentrering av elementer i element. Vi brukte veldig lite av rammeverket sine tilhørende utseender.
Det viktigste Bootstrap ga prosjektet vårt var mulighetene og alternativene det ga for responsiv design. Bootstrap i seg selv har veldig mye bra dokumentasjon å hente både idéer og veiledning fra. Vi er fornøyde med Bootstrap og måten vi brukte rammeverket på, fordi det var enkelt å bruke og lære seg, lot oss enkelt implementere responsiv design og mye bra dokumentasjon.

5.1.2.2 Spring

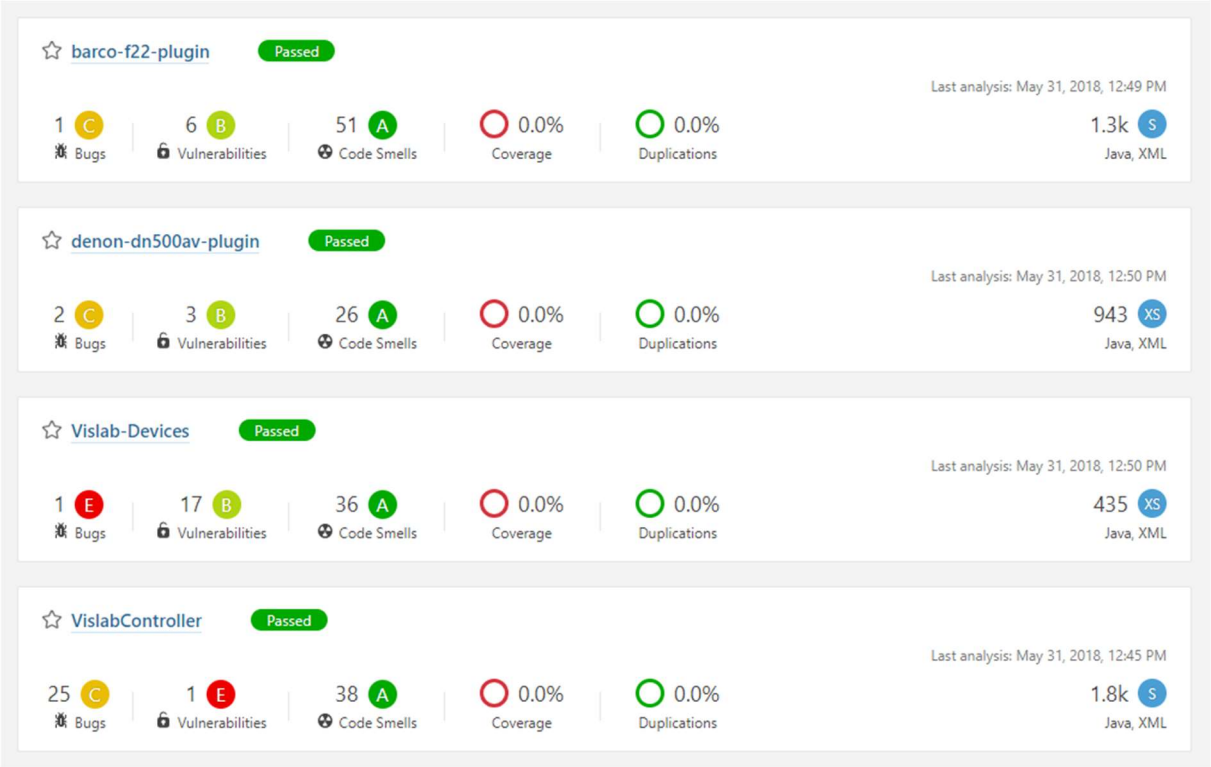
Vi brukte flere elementer fra Spring rammeverket. Det som ble minst brukt var Spring MVC. Dette rammeverket kunne ha gjort deler av arbeidet vi gjør i JavaScript på server nivå. Med å ha tatt i bruk MVC, hadde vi ikke trengt å føre over så mye data som vi gjør til bruker. Spring Boot har gjort veldig mye arbeid som vi slepper ved å sette opp innebygd Tomcat web server, som fanger opp ServletRequests slik at vi kan gjøre logikk.

5.1.2.3 Docker

Docker ble mye brukt løpet av prosjektet. Fra å starte våre egne program, til å bli brukt av Jenkins til å kjøre de jobbene som trengs. Docker var komplekst å komme i gang med, men når syntaksen ble forstått gjorde docker mye av det tunge arbeidet, slik at vi kunne fokusere mer på programmering.

5.1.2.4 SonarQube

SonarQube var til stor hjelp på starten av prosjektet. Den pekte på små feil som vi gjorde som ville ha blitt vanskelig eller umulig å fikse om de ikke hadde blitt funnet tidlig. SonarQube er veldig kraftig når det har blitt konfigurert riktig. Å få gjort en analyse i SonarQube tar tid, ettersom det krever en del manuelt oppsett første gangen. Maven kan brukes til å få en analyse etter at bygging har skjedd. Mot siste halvdel av prosjektet ble SonarQube lite brukt ettersom oppsettet måtte gjøres om. Prosjektene var avhengig av hverandre for å fungere, tester ble vanskelige å skrive ettersom alt ble gjort på egne tråder.



Figur 5-2: SonarQube analyse resultat

Vi utførte en manuell analyse ved avslutting av prosjektet for å se hva analysen viste. Resultatet kan sees i Figur 5-2. Alle prosjektene passerte SonarQube sin standard kvalitets sikring.

5.1.2.5 Maven

Vi valgte maven som byggsystem på grunn av erfaringer med dette og at vi kan finne klasser gjennom et sentralt repository. Dette gjorde utvikling veldig enkelt og fjernet masse tid på å lete, laste ned og legge inn biblioteker manuelt. Gjennom konfigurasjon kunne vi og la maven bygge opp fil og mappe struktur som programmet trenger for å fungere. Bygg-delen av maven ble ikke tatt i bruk før mot slutten av prosjektet, før dette var Intellij sin integrerte bygg system som ble brukt, for å kunne enklere debugge programmet.

5.1.2.6 JUnit

I starten ble det skrevet tester i JUnit. Dette ble det fort slutt på ettersom prosjektet fikk flere tråder. Der finnes rammeverk for å skrive tester på tråder, men vi bestemte som gruppe at å sette oss inn i disse ville ta for mye tid. Testing av funksjoner til enhetene blir og vanskelige ettersom det måtte bli bygd en mini simulator kun for testen.

5.1.3 Prosjektet

5.1.3.1 Scrum

Tidligere nevnt i rapporten (kapittel 3.3) så brukte vi Scrum som arbeidsrammeverk. Vi er for det meste ganske fornøyd med hvordan det gikk å arbeide med Scrum. Der var noen elementer av Scrum vi ikke utnyttet helt, slik som de daglige møtene, men opplevelsen var mer positiv enn negativ med Scrum. Noen fordeler vi dro nytte av med Scrum var:

- Fleksibelt arbeid, kunne tilpasse hva vi synes var nødvendig å prioritere etter hver sprint.
- Mindre planlegging i starten, noe som resulterer at vi kommer raskere i gang å arbeide. Planen skriver nesten seg selv om verktøy slik som JIRA blir brukt.
- Dele store oppgaver ned i mindre oppgaver som vi kan enten løse en og en, eller samarbeide om slik at de blir gjort fortere.
- Kontinuerlig tilbakemelding fra oppdragsgiver, det at oppdragsgiver involver seg i utviklingsprosessen gjør det mye enklere for oss som utviklere å prioritere riktig, og få enkelte elementer på plass slik som ønsket.

Men ikke alt med Scrum følte vi var helt positivt, eller vi ikke klarte å utnytte ordentlig, de var som følger:

- De daglige møtene, vi hadde enten ingen daglige møter, eller vi brukte de ikke optimalt, slik at vi etter en liten stund ikke hadde noen.
- Erfaring med Scrum. Vi følte til tider at vår mangel på Scrum-erfaring førte til ineffektiv tid brukt på konseptene i Scrum, slik som oppretting av User Stories og Tasks følte vi ble tungvint på grunn av vi ikke gjorde det riktig.

5.1.3.2 Samarbeid

Som gruppe anser vi at samarbeidet ble godt utført med et par unntak. Det vi synes gikk bra var:

- Vi gjorde vårt beste i å fullføre hver sprint vi planla.
- Våre forskjellige kompetansenivå mikset bra sammen, slik at folk fikk arbeide med det de ønsket, med ett unntak.
- Respekterte hverandre sine situasjoner og grenser.

Unntak:

- Det ble til tider tregt oppmøte fra noen gruppemedlemmer, men det gikk for det meste bra.
- Noen hadde nødt til å arbeide med design og utforming av GUI.

Som gruppe oppholdt vi oss i Visualiserings Labben hos NTNU i Ålesund. Vi satt på linje og arbeidet, og om det var noe bare snakket vi med hverandre slik at hele dagen bestod av kontinuerlig arbeid og kommunikasjon. I de sjeldne tilfellene hvor vi ikke befant oss på skolen og jobbet hadde vi alltids en gruppesamtale på Facebook. Gjennom hele prosjektet har vi hatt god kommunikasjon mellom gruppemedlemmene.

5.2 Videreutvikling av Prosjektet

Ved leveranse av produktet, versjon 1.0. Dette er den første fullførte og fullverdige versjonen av systemet. Denne versjonen inneholder det meste av det som er gitt i kravspesifikasjonen (LCD-skjermer støtter ikke HTTP-kommunikasjon). En del av funksjonaliteten i systemet var heller ikke en del av den originale krav spesifikasjonen, men kom heller fram i samtaler med produkteier.
Det vil bli listet hva som kan bli forbedret med systemet, og funksjonalitet som kan implementeres, som ble diskutert, men aldri implementert på grunn av tidsmangel eller lignende konflikter.

5.2.1 Sortering på Device List

Device List siden inneholder flere tabeller som holder på informasjonen til de forskjellige enhetene. Vi tenkte det hadde vært lurt med en funksjon som lar brukeren sortere de på en eller annen måte. Slik som avansert søk med kriterier. Kriteriene kan være attributter slik som hvilke teater de hører til i, hvilken produsent eller modell de har.

5.2.2 Administrator Funksjoner

Administrator siden har en grei grunnstruktur, men noe av utseendet er ikke spesielt bra. Noen av funksjonene på administrator siden fungere ikke optimalt, for eksempel forandring av IP på en enhet fungere ikke slik det skal.

5.2.3 Forbedring av Brukergrensesnitt

Grunnstrukturen til brukergrensesnittet er vi svært fornøyd med, men det kosmetiske utseendet kan ikke vi si oss spesielt fornøyd med. Ingen i gruppen følte seg artistisk begavet slik at vi endte opp med et grensesnitt tatt rett ut av MySpace sin era på Internett.

5.3 Arbeidsfordeling

Gruppen delte seg inn i egne områder når det kom til type arbeidsoppgaver. Fordelingen så omtrent slik ut:
10011:

- Kommunikasjonsdriver
- Modulisering av prosjektet

10035:

- Web
- GUI
- litt alt - db og Docker

10048:

- Docker
- Database
- API

5.4 JavaScript

Da prosjektet nærmet seg slutten ble det mer og mer tydelig at vi skulle ha tatt i bruk et rammeverk for å handtere bruker interaksjon på nettsiden. Mye av arbeidet utført av JS var dynamisk generering av HTML-elementer, endring av tekstfelt og lignende, slike oppgaver er tidkrevende å skrive i JS og kunne ha blitt utført mye enklere og raskere med hjelp av forskjellige rammeverk slik som thymeleaf og angular.
Etter å ha arbeidet en del med JavaScript la vi merke til noe av funksjonaliteten vi ønsket var enten vanskeligere enn forventet å implementere eller fungerte ikke som forventet slik at vi var nødt til å bruke enda mer tid på det. Om vi skulle gjort det på nytt, eller hatt nok tid til å implementere ville vi brukt noen JavaScript-biblioteker som Ajax eller jQuery.

5.5 CSS og Bootstrap

I starten av prosjektet hadde vi bare vanlig CSS for utforming av GUI. Det viste seg fort at CSS alene var veldig tidkrevende, derfor bestemte vi oss for å ta i bruk Bootstrap for å begrense tiden vi brukte på CSS. Bare de mer spesifikke elementene på siden har helt egen CSS nå.

6 Konklusjon

Hensikten med prosjektet var å utvikle en funksjonell og brukervennlig web-applikasjon som kan:

- Hente data fra en server.
- Vise hentet data til brukeren.
- Sende enkle kommandoer over HTTP til en server.
- Fleksibel i bruk med tanke på enheten.

Applikasjonen vi kom fram til er/kan alle de punktene nevnt over. I starten av prosjektet kom vi opp med problemstillingen:
*"Utvikle et brukergrensesnitt som fungerer like godt på små skjermer, som store og utvikle enhetskontrolleren som håndterer kommunikasjon mellom brukergrensesnitt, applikasjon og enhetene." (*parafrasert fra kapittel 1.3).
I denne rapporten blir denne problemstillingen besvart av resultatene vi oppnådde med metodene vi valgte å bruke. Fra resultat kan vi se at vi lagde en løsning som:

- Responsiv nettside.
- Enkel å utvide, modulær.
- Brukervennlig til en viss grad.
- Kan brukes på alle standard skjermstørrelser.
- Innfrir de fleste universell utforming krav.
- Rolle-basert tilgang.
- Enkel å utvide REST-API.
- Forenkler bruken av implementerte enheter.
- Tilbyr enkle generelle operasjoner, men også mer avanserte og spesifikke.
- Reduserer vanskelighetsgraden og kunnskapsnivået som er nødvendig for å bruke enhetene.

Produktet vi har endt opp med er ingen fasit på hvordan en kan løse en slik oppgave, men det er en samling av beslutninger vi har tatt underveis i arbeidsprosessen. Om vi kunne gjort dette prosjektet på nytt er det flere beslutninger vi kunne gjort annerledes slik at vi endte opp med et annet resultat fortere. Flere av beslutningene vi tok, lærte vi senere at de ikke alltid var den mest effektive, men disse opplevelsene kan vi i utviklingsteamet ta med oss videre som erfaringer slik at vi tar mer effektive beslutninger videre.

Å arbeide med et slikt prosjekt var en god del annerledes enn hva vi allerede var kjent med i forbindelse med skoleprosjekter. I dette prosjektet måtte vi bestemme egne arbeidsperioder og –mengder, samtidig som vi kommuniserte med andre gruppemedlemmer om hva som skulle bli gjort under sprintene slik at vi alltid arbeidet mot samme mål i fellesskap.

Gjennom drøfting og evalueringen er det tydelig at vi er fornøyd med det vi har gjort. Selv om det er simpelt utformet, er funksjonaliteten i bakgrunnen kraftig. Programmet selv gjør utviding veldig enkelt ved at det ikke trenger å rekompilere hovedprogrammet. Moduler blir laget som egne små Java program som blir lastet inn når hovedprogrammet starter.
I kravspesifikasjonen stør det flere punkt om LCD skjermer. Disse ble ikke oppnådd på grunn av at skjermene på labben ikke har Ethernet port, og kan dermed ikke

styres gjennom TCP/IP. De resterende punktene føler vi som gruppe at vi nådde, og håper at oppdragsgiver kan ta i bruk systemet uten problemer.

7 Referanser

1. Esfandiari R. Device Control Application. Aalesund: NTNU Aalesund; 2016 des s. 112.

2. State pattern. I: Wikipedia [Internett]. 2017 [sitert 5. mars 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=State_pattern&oldid=805406208

3. What is Agile? What is Scrum? [Internett]. cPrime. 2013 [sitert 31. mai 2018]. Tilgjengelig på: https://www.cprime.com/resources/what-is-agile-what-is-scrum/

4. The Scrum Framework Poster [Internett]. Scrum.org. [sitert 31. mai 2018]. Tilgjengelig på: http://www.scrum.org/resources/scrum-framework-poster

5. Scrum (software development). I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Scrum_(software_development)&oldid=843063776

6. Johnson J. Designing with the Mind in Mind: Simple Guide to understanding User Interface Design Guidelines. Second. Morgan Kaufmann; 2014.

7. Responsive web design. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Responsive_web_design&oldid=841820825

8. Lov om likestilling og forbud mot diskriminering (likestillings- og diskrimineringsloven) - Lovdata [Internett]. [sitert 25. mai 2018]. Tilgjengelig på: https://lovdata.no/dokument/NL/lov/2017-06-16-51/

9. Oppbygging av WCAG 2.0 | Universell utforming [Internett]. [sitert 25. mai 2018]. Tilgjengelig på: https://uu.difi.no/krav-og-regelverk/wcag-20-standarden/oppbygging-av-wcag-20

10. Creating Extensible Applications (The Java™ Tutorials > The Extension Mechanism > Creating and Using Extensions) [Internett]. [sitert 31. mars 2018]. Tilgjengelig på: https://docs.oracle.com/javase/tutorial/ext/basics/spi.html

11. Howe S. Learn to Code HTML & CSS: Develop & Style Websites. New Riders; 2014.

12. contributors MO Jacob Thornton, and Bootstrap. Bootstrap [Internett]. [sitert 4. mai 2018]. Tilgjengelig på: https://getbootstrap.com/

13. JavaScript [Internett]. MDN Web Docs. [sitert 5. mai 2018]. Tilgjengelig på: https://developer.mozilla.org/en-US/docs/Glossary/JavaScript

14. JSON [Internett]. [sitert 28. mai 2018]. Tilgjengelig på: https://www.json.org/

15. What is Unified Modeling Language (UML)? [Internett]. [sitert 31. mai 2018]. Tilgjengelig på: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

16. Wireframes | Konigi [Internett]. 2009 [sitert 28. mai 2018]. Tilgjengelig på: http://konigi.com/uiref/wireframes/

17. Website wireframe. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Website_wireframe&oldid=841570391

18. HTTP [Internett]. MDN Web Docs. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://developer.mozilla.org/en-US/docs/Web/HTTP

19. HTTP request methods [Internett]. MDN Web Docs. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

20. HTTP response status codes [Internett]. MDN Web Docs. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

21. What is an Event-Driven Program? - Definition from Techopedia [Internett]. Techopedia.com. [sitert 5. mai 2018]. Tilgjengelig på: https://www.techopedia.com/definition/7083/event-driven-program

22. Barnes DJ, Kölling M. Objects First with Java. Fifth. Pearson; 2012. 546 s.

23. Java (programming language). I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=842316815

24. Java virtual machine. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Java_virtual_machine&oldid=838292482

25. Java Development Kit. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Java_Development_Kit&oldid=838825201

26. Representational state transfer. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=842551527

27. Java Platform, Enterprise Edition. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på:

https://en.wikipedia.org/w/index.php?title=Java_Platform,_Enterprise_Edition&oldid=841001353

28. Java Persistence API. I: Wikipedia [Internett]. 2018 [sitert 28. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Java_Persistence_API&oldid=836910942

29. Lesson: Concurrency (The Java™ Tutorials > Essential Classes) [Internett]. [sitert 31. mai 2018]. Tilgjengelig på: https://docs.oracle.com/javase/tutorial/essential/concurrency/

30. Kristoffersen B. Databasesystemer. 4. Utgave. Universitetsforlaget; 2016.

31. Relational database management system. I: Wikipedia [Internett]. 2018 [sitert 30. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Relational_database_management_system&oldid=842515092

32. Database. I: Wikipedia [Internett]. 2018 [sitert 30. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Database&oldid=842863465

33. MySQL :: Supported Platforms: MySQL Database [Internett]. [sitert 29. mai 2018]. Tilgjengelig på: https://www.mysql.com/support/supportedplatforms/database.html

34. Version control. I: Wikipedia [Internett]. 2018 [sitert 31. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Version_control&oldid=842520051

35. Git. I: Wikipedia [Internett]. 2018 [sitert 31. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Git&oldid=843622974

36. Atlassian. Git Workflow | Atlassian Git Tutorial [Internett]. Atlassian. [sitert 31. mai 2018]. Tilgjengelig på: https://www.atlassian.com/git/tutorials/comparing-workflows

37. Atlassian. Sourcetree | Free Git GUI for Mac and Windows [Internett]. SourceTree. [sitert 31. mai 2018]. Tilgjengelig på: https://www.sourcetreeapp.com

38. Microsoft Azure. I: Wikipedia [Internett]. 2018 [sitert 29. mai 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Microsoft_Azure&oldid=843379101

39. Prisoversikt – Hvordan Azure-priser virker | Microsoft Azure [Internett]. [sitert 29. mai 2018]. Tilgjengelig på: https://azure.microsoft.com/nb-no/pricing/

40. Spring Framework Overview [Internett]. [sitert 30. mai 2018]. Tilgjengelig på: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/overview.html

41. Jenkins [Internett]. Jenkins. [sitert 25. april 2018]. Tilgjengelig på: https://jenkins.io/index.html

42. Continuous Code Quality | SonarQube [Internett]. [sitert 25. april 2018]. Tilgjengelig på: https://www.sonarqube.org/

43. Docker (software). I: Wikipedia [Internett]. 2018 [sitert 30. januar 2018]. Tilgjengelig på: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=842989879

44. Overview of Docker Compose [Internett]. Docker Documentation. 2018 [sitert 23. mai 2018]. Tilgjengelig på: https://docs.docker.com/compose/overview/

45. Maven – Welcome to Apache Maven [Internett]. [sitert 21. mars 2018]. Tilgjengelig på: https://maven.apache.org/

46. Atlassian. What is DevOps? [Internett]. Atlassian. [sitert 31. mai 2018]. Tilgjengelig på: https://www.atlassian.com/devops

47. JUnit 5 User Guide [Internett]. [sitert 30. mai 2018]. Tilgjengelig på: https://junit.org/junit5/docs/current/user-guide/

48. Atlassian. Jira | Issue & Project Tracking Software [Internett]. Atlassian. [sitert 31. mai 2018]. Tilgjengelig på: https://www.atlassian.com/software/jira

49. Atlassian. Confluence - Team Collaboration Software [Internett]. Atlassian. [sitert 31. mai 2018]. Tilgjengelig på: https://www.atlassian.com/software/confluence

50. Barco. Barco pw392 Communication Protocol [Internett]. Barco; [sitert 1. februar 2018]. Tilgjengelig på: https://www.barco.com/en/product/f22-series

51. Denon Professional. Denon Professional - AV Surround Preamplifier - DN-500AV - Owner's Manual [Internett]. [sitert 11. april 2018]. Tilgjengelig på: https://denonpro.com/products/view/dn-500av

52. What is Scrum? [Internett]. Scrum.org. [sitert 6. april 2018]. Tilgjengelig på: http://www.scrum.org/resources/what-is-scrum

53. Java Platform, Enterprise Edition (Java EE) | Oracle Technology Network | Oracle [Internett]. [sitert 23. mai 2018]. Tilgjengelig på: http://www.oracle.com/technetwork/java/javaee/overview/index.html

54. ServiceLoader (Java Platform SE 7) [Internett]. [sitert 28. mai 2018]. Tilgjengelig på: https://docs.oracle.com/javase/7/docs/api/java/util/ServiceLoader.html

55. META-INF/services generator - [Internett]. [sitert 4. april 2018]. Tilgjengelig på: http://metainf-services.kohsuke.org/

56. Pivotal Software, Inc. Spring Framework 5 [Internett]. spring.io. [sitert 27. mai 2018]. Tilgjengelig på: http://spring.io/

57. Spring Boot Reference Guide [Internett]. [sitert 31. mai 2018]. Tilgjengelig på: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

58. Spring Projects [Internett]. [sitert 31. mai 2018]. Tilgjengelig på: http://spring.io/projects/spring-boot

59. Ny, rask nettleser for Mac, PC og Linux | Firefox [Internett]. Mozilla. [sitert 24. mai 2018]. Tilgjengelig på: https://www.mozilla.org/nb-NO/firefox/

60. Firefox Developer Edition [Internett]. Mozilla. [sitert 24. mai 2018]. Tilgjengelig på: https://www.mozilla.org/nb-NO/firefox/developer/

61. Chrome for datamaskiner [Internett]. [sitert 24. mai 2018]. Tilgjengelig på: https://www.google.com/chrome/

62. Safari (nettleser). I: Wikipedia [Internett]. 2018 [sitert 24. mai 2018]. Tilgjengelig på: https://no.wikipedia.org/w/index.php?title=Safari_(nettleser)&oldid=18540142

63. Microsoft Edge. I: Wikipedia [Internett]. 2017 [sitert 24. mai 2018]. Tilgjengelig på: https://no.wikipedia.org/w/index.php?title=Microsoft_Edge&oldid=17896507

64. Internet Explorer. I: Wikipedia [Internett]. 2017 [sitert 24. mai 2018]. Tilgjengelig på: https://no.wikipedia.org/w/index.php?title=Internet_Explorer&oldid=18081726

65. Browser Statistics [Internett]. [sitert 24. mai 2018]. Tilgjengelig på: https://www.w3schools.com/BROWSERS/default.asp

66. Hvordan teste universell utforming av ditt nettsted | Universell utforming [Internett]. [sitert 31. mai 2018]. Tilgjengelig på: https://uu.difi.no/krav-og-regelverk/kom-i-gang/hvordan-teste-universell-utforming-av-ditt-nettsted

Vedlegg

Vedlegg A	Kravspesifikasjon
Vedlegg B	Kildekode
Vedlegg C	Forprosjektrapport
Vedlegg D	Arbeidslogg
Vedlegg E	Sprint evaluering møtereferat
Vedlegg F	Web-API
Vedlegg G	Nye moduler
Vedlegg H	Fil og mappe struktur
Vedlegg I	README.txt
Vedlegg J	Klassediagram

Vedlegg A: Kravspesifikasjon

Functional Requirements

The functional requirements are described by (but not limited to) the following user stories:

- *As a user I want to be able to remotely control devices like projectors, LCD-screens and surround amplifiers over ethernet (TCP/IP) wired and wireless.*
    - *As a user controlling projectors*
        - *I want to be able to switch the device power on and off (or standby)*
        - *I want to be able to turn on and off a test image*
        - *I want to be able to mute and un-mute the image*
    - *As a user controlling LCD-screens*
        - *I want to be able to switch the device power on and off (or standby)*
        - *I want to be able to select input source*
- *As a user I want to be able to organise the devices to be controlled into logical groups, independent upon type of device. Typically this function would be used to create a group for all projectors that makes up one "theatre" or "lab". For the vis lab we only have one such "Theatre", but such an application should support larger solutions like the nautical simulators at HIALS made up of 5 (or more) independent "bridges" (or theatres).*
    - *Within each group of devices, as a user I want to be able to select one or multiple devices by clicking on the devices. The selecting-a-device should work like a toggle-button. One click selects another click de-selects. There should be NO use of holding the shift key to enable selection of multiple devices.*
    - *Within each group of devices, as a user I want to be able to define "selection patterns" (Example: all projectors on the left half of the vials, all projectors on the right half, all 3 3D projectors in the vis lab etc.) to select a predefined set of devices.*
    - *Within each group of devices, as a user I want to be able to perform the following actions on the selected devices:*
        - *Switch power on/off*
        - *Enable/disable test image (for projectors)*
        - *Mute- and un-mute devices (and projectors specifically)*

Technical/Architectural requirements

The technical/architectural requirements are described by (but not limited to) the following:

- *All devices to be controlled must support control over TCP/IP, preferably wired and not wireless. This to ensure stability, and reduced risk of hacking.*
- *The application shall be implemented as a web-application, with the server part running in an application server like Tomcat or Glassfish on a server in the datacenter at the VisLab.*

- *The User Interface of the application should be implemented as a web client solution supporting "responsive design", enabling control from any device with a web browser (PC/Mac, iPad/Tablet, iPhone/smartphone etc.).*
- *The application should be designed in a modular and layered way making the application easy to expand with new devices to be controlled.*

Vedlegg B: Kildekode

Kildekoden vår kan finnes på GitHub, hvor vi har opprettet en organisasjon kalt Device Controller Vislab, https://github.com/Device-Controller. Siden kildekoden vår har blitt delt inn i flere biter, kan den finnes direkte ved følgende lenker:

- **Controller**, Hovedapplikasjonen: https://github.com/Device-Controller/Controller
- **DenonDN500AV**, Kommunikasjonsmodulen til Denon DN500-AV: https://github.com/Device-Controller/DenonDN500AV
- **BarcoF22Controller**, Kommunikasjonsmoduler til Barco F22: https://github.com/Device-Controller/BarcoF22Controller
- **template**, En mal for hvordan man kan opprette en ny kommunikasjonsmodul: https://github.com/Device-Controller/template
- **Device-Interface-library**, Bibliotek som har oversikt over alle opprettede objekter: https://github.com/Device-Controller/Device-Interface-library

Kildekode er også levert inn i egen *.zip-fil.

Vedlegg C: Forprosjektrapport

FORPROSJEKT – RAPPORT
FOR BACHELOROPPGAVE

NTNU
Kunnskap for en bedre verden

TITTEL:
Device Controller - VisLab

STUDENTNUMMER(E):			
460822 - Kristoffer Rogne 140079 - Thomas Skarshaug Todal 271778 - Erik Haram Nygård			
DATO:	EMNEKODE: *	EMNE:	DOKUMENT TILGANG:
	IE303612	**Bacheloroppgave**	- Åpen
STUDIUM:		ANT SIDER/VEDLEGG:	BIBL. NR:
DATA		10/2	- Ikke i bruk -

OPPDRAGSGIVER(E)/VEILEDER(E):
NTNU/Arne Styve

OPPGAVE/SAMMENDRAG:
Hensikten med oppgaven er å utvikle en applikasjon som tillater enkel, brukervennlig og oversiktlig bruk av enhetene på visualiseringslabben. For å tilnærme oss en løsning for den oppgaven tar vi Scrum i bruk som arbeidsmetode. Det ble utdelt en spesifikasjon i hva som skal være med i løsningen og oppgaveprioritetene er utformet fra den.

Postadresse
Høgskolen i Ålesund
N-6025 Ålesund

Norway

Besøksadresse
Larsgårdsvegen 2
Internett
Foretaksregisteret
www.ntnu.no

Telefon
70 16 12 00
Epostadresse
postmottak@ntnu.no

Telefax
70 16 13 00

NO 971 572 140

Bankkonto
7694 05 00636

1 INNLEDNING

Oppgaven var valgt på bakgrunn at den byr på utfordringer innen mange av fagområdene vi har lært; webteknologi, objektorientert programmering, datamodellering og databaseapplikasjoner, informasjonssikkerhet, systemadministrasjon, sanntids datateknikk. Oppgaven utfordrer oss på de nevnte emnene i følgende sammenhenger: Spring, GUI, kommunikasjon mellom enheter, og flere.

Oppdragsgiver er Norges teknisk-naturvitenskapelig universitet (NTNU)

Oppgaven går ut på å lage et kontrollsystem for å styre enheter som projektorer, display, lydforsterkere etc. på visualiseringslabben ved skolen. Formålet med oppgaven er å gjøre det enklere å bruke visualiseringslabben.

2 BEGREPER

2.1 Ord / Forkortelser

Device Controller

Enhetskontroller.

GUI

Graphical User Interface/Grafisk Brukergrensesnitt.

VisLab

Simulering- og visualiseringslabben ved NTNU campus Ålesund.

Rollback / Rulle tilbake

Gå tilbake til et tidligere stadium.

Commit

Begå en endring i GIT systemet.

IDE

Integrated Development Environment/ Integrert utviklingsmiljø, et stykke programvare som samler de viktigste funksjonene rundt programvareutvikling med et eller flere programmeringsspråk på ett sted.

2.2 Teknikker

Scrum

Scrum er en arbeidsmetode der teamet er selvdrevet og kan definere egne mål for å fullføre prosjektet. Gjennom en kort iterativ prosess (sprint) blir målene nådd. Målene som skal nås blir lagt i ei liste (Backlog). Denne lista vokser og krymper i løpet av prosjektets livsløp.
Det er daglige møter i løpet av en sprint der medlemmene snakker om kva som har blitt gjort, og hva som viste seg å være problematisk å utføre. Disse møtene er korte, mellom 10-15 min lange.
Når sprinten er over vil det bli demonstrert til oppdragsgiver(e), som gir tilbakemeldinger på utført arbeid og ønskelige endringer.

Sprint

Arbeidsperiode i Scrum, 1 - 4 uker lang er vanlig. På starten av en sprint blir det valgt ut hvilke arbeidsoppgaver som skal utføres. Ved sprint slutt blir arbeidet evaluert og avgjørelser om det var for mye eller for lite arbeid i sprinten blir tatt.

2.3 Verktøy

Spring (REST)

Et web-tjeneste rammeverk. Abstrakterer bort mellomlaget mellom database og web server, slik at vi kan fokusere på applikasjonslogikk.

JIRA

Jira er et verktøy som lar medlemmene i prosjektgruppen planlegge og samle alle de nødvendige oppgavene som må utføres på en plass. Jira tillater at oppgaver blir tildelt et antall poeng som viser hvor mye tid hver oppgave antas å ta, i tillegg til at en oppgave kan prioriteres etter hvor kritisk den er til prosjektet. Det vil hjelpe med å utføre oppgaven på en ordentlig og helhetlig måte.

Confluence

Confluence er et verktøy som tillater flere folk i et prosjekt å samle all informasjonen de bedømmer som nødvendig til en plass.

Netbeans

Netbeans er en IDE, som brukes til å skrive programvare i flere programmeringsspråk, i dette prosjektet blir Netbeans i hovedsak brukt for å skrive programvare i programmeringsspråket Java.

GIT

Git er et versjonerings verktøy som holder styr på commits som blir gjort inn i systemet. Det er enkelt å rulle tilbake programvare om det viser seg at ei endring ikke fungerer. Git holder styr på hvem som la inn endringer og tilbyr løsninger om det skulle oppstå konflikter mellom to endringer.

3 PROSJEKTORGANISASJON

3.1 Prosjektgruppe

Studentnummer	Navn	E-Post
140079	Thomas Skarshaug Todal	thomatod@stud.ntnu.no
460288	Kristoffer Rogne	kristrog@stud.ntnu.no
271778	Erik Haram Nygård	erikhny@stud.ntnu.no

Oppgaver for prosjektgruppen - organisering

Roller	Person	Beskrivelse
Scrum Master	Kristoffer Rogne	Skal lede Scrum-møtene.
Utvikler	Kristoffer Rogne	Utvikle programvare
Utvikler	Thomas Skarshaug Todal	Utvikle programvare
Utvikler	Erik Haram Nygård	Utvikle programvare
Sekretær	Erik Haram Nygård	Skal passe på at rapporten blir skrevet.
Designansvarlig	Thomas Skarshaug Todal	Ha kontroll på at GUI-designet blir gjort på en akseptabel måte.

Oppgaver for SCRUM Master

Gruppen vil ikke ha en tradisjonell prosjektleder, men heller en SCRUM master. Oppgavene til SCRUM master vil være å lede SCRUM møtene. Det skal stilles tre spørsmål: Hva har blitt gjort siden sist møte, Hva vil du gjøre i dag, Hva har vist seg å være problematisk.

3.2 Styringsgruppe (veileder og kontaktperson oppdragsgiver)

Navn	Rolle	E-Post
Arne Styve	Veileder, Oppdragsgiver, Produkteier	asty@ntnu.no

4 AVTALER

4.1 Avtale med oppdragsgiver

Vi har avtale med oppdragsgiver at vi skal møtes annenhver uke for å vurdere arbeidet som har blitt utført, og diskutere videre steg.

4.2 Arbeidssted og ressurser

Vi har tilgang til vislabben ved NTNU. Labben inneholder utstyret som skal jobbes med.
Hver andre fredag blir en framdriftsrapport presentert til oppdragsgiver.
Arbeid vil foregå på vislab, eventuelt på L167 om labben er utilgjengelig.
Arbeidsplassen er nært flere ressurspersoner om problemer eller uklarheter skulle oppstå.

4.3 Gruppenormer – samarbeidsregler – holdninger

For å oppnå optimalt samarbeid mellom gruppemedlemmene har vi blitt enig om et antall punkter:

- Alle skal møte opp i tide, om det ikke er mulig skal beskjed bli gitt i god tid.
- Tilsynelatende godt humør.
- Ikke starte unødvendig konflikt
- Utføre arbeidsoppgavene innen rimelig tidsrom.
- Alle gruppemedlemmer er pliktig å skrive i rapporten minst en gang i uken.

Gruppen står for å utvikle programvare som oppfyller kravspesifikasjonen, ikke har noen kritiske feil og har sikkerhetsegenskaper som oppfyller dagens sikkerhetsstandard.

"Man skal ikke plage andre, man skal være grei og snill, og for øvrig kan man gjøre hva man vil." - Thorbjørn Egner 1955

5 PROSJEKTBESKRIVELSE

5.1 Problemstilling - målsetting - hensikt

	Beskrivelse
Problemstilling	Det finnes en løsning for å kontrollere enhetene på vislabben, denne løsningen er ikke veldig brukervennlig og heller ikke særlig kraftfull. Problemstillingen vil bli å utvikle en kontroller til disse enhetene som kan enkelt utvides. Løsningen skal tillate enklere bruk og vedlikehold.
Effektmål	Løsningen skal gjøre bruk og vedlikehold av utstyr enklere og raskere.
Resultatmål	Sluttleveransen av prosjektet skal føre til større og mer effektiv bruk av verktøyene på vislabben.
Målsetting	- Lage en brukervennlig GUI - Gjøre løsningen så generell som oppgaven tillater - Responsiv design

5.2 Krav til løsning eller prosjektresultat – spesifikasjon

Prosjektet skal inneha visse egenskaper som blir spesifisert i spesifikasjonen, disse egenskapene er:

Fjernstyring av enheter som projektorer, LCD-skjermer og lydforsterker over TCP/IP, både kablet og trådløs. For projektorer skal det være følgende funksjonalitet:
- Skru av og på (eller standby-modus)
- Aktivere/deaktivere et testbilde
- Skru av og på signal til projektoren

For LCD-skjermer skal vi ha følgende funksjonalitet:
- Skru av og på (eller standby-modus)
- Velge signalkilde

Enhetene som kobles til skal kunne organiseres i større grupper som kan styres som én enhet, og ha all funksjonalitet samlet på ett sted. En gruppe vil typisk være alle enhetene i samme lab, eller i en av simulatorene til skolen. Det skal kunne være mulig å velge de ulike enhetene i en gruppe, for å legge til nye elementer i en gruppe, skal du kunne merke flere elementer samtidig ved å trykk én gang på hver enhet, et ekstra trykk vil avmerke enheten. Det skal ikke være noe nødvendighet å holde inne shift-tasten for å velge flere enheter. Innenfor en gruppe enheter skal det være mulig å definere markeringsmønster (undergrupper), f.eks.: projektorene på venstre side, eller projektorene på høyre side.

For hver enhet i en gruppe skal det være mulig å:
- Skru av og på
- Aktivere/deaktivere et testbilde
- Skru av og på signal til enhetene

De tekniske kravene er beskrevet av, men ikke begrenset til, følgende:
- Alle enheter som skal styres må støtte styring over TCP/IP, fortrinnsvis kablet, og ikke trådløst. Grunnen til dette er for å sikre stabilitet, og økt sikring mot hackeangrep.
- Applikasjonen skal implementeres som en web-applikasjon med server-delen som en applikasjonsserver som Tomcat eller Glassfish på en server i vislabben.
- GUIen til applikasjonen skal bli implementert som en web-klient som støtter "responsive-design" som tillater kontroll fra en hvilken som helst enhet med en nettleser (PC/MAC, Nettbrett, Smarttelefon).
- Applikasjonen skal bli utviklet på en modulær og lagvis måte som tillater utvidelse og videreutvikling.

5.3 Planlagt framgangsmåte for utviklingsarbeidet – metode

Vi skal bruke SCRUM, der arbeid blir planlagt for en sprint framover. Planlagt rekkefølge for de ulike komponentene i applikasjonen vil være:
1. Applikasjonsserver
2. Kommunikasjonsgrensesnitt
3. Webserver
4. Database
5. Videreutvikling av GUI

Med Scrum så vil hver komponent bli oppdelt i flere issues som videre blir fordelt inn i tasks, stories og epics. En av Scrum sine største svakheter er usikkerhet når en blir ferdig med hele prosjektet, det vi kan gjøre for å redusere denne svakheten er å prioritere riktig i arbeid som blir valgt i hver sprint, slik at om vi ikke blir helt ferdig til slutt, er i hvert fall det mest kritiske på plass.

5.4 Informasjonsinnsamling – utført og planlagt

En begrenset løsning for prosjektet finnes allerede med en enkel GUI og begrenset bruk av projektorene. Under utvikling av dette prosjektet vil denne løsningen bli brukt som referanse om hvordan vi kan utføre enkelte oppgaver, men hovedsakelig vil prosjektet bli utviklet uavhengig av det tidligere arbeidet.
Produsenten av projektorene har et eget system for å styre projektorene som det kunne vært mulig å benytte seg av, men siden vi har satt som mål å lage et mer universelt system med støtte for flere enheter vil nok ikke dette være særlig relevant for oss. Dette kan derimot bli brukt som referanse.

5.5 Vurdering – analyse av risiko

Konsekvens -> Sannsynlighet	Liten	Middels	Kritisk	Katastrofal
Veldig stor (>50%)				
Stor (25-50%)				
Middels (5-24%)	H	G	A, B	
Liten (1-4%)	F	C	E	
Minimal (<1%)	D		I	

ID	Beskrivelse	Ansvar	Tiltak
A	Går tom for tid	Alle	Prioritere oppgaver bedre
B	Sikkerhetsprotokoller i NTNU sitt nettverk	Alle	Snakke med IT-avdelingen og finne en løsning
C	Projektor/LCD-monitor/lydforsterker svikter	Alle	Melde fra til NTNU
D	Vi trenger noe utstyr/programvare som koster for mye	Alle	Finne alternativ løsning
E	Sykdom i gruppa	Den det gjelder	Ikke bli syk
F	Personlige skader	Den det gjelder	Tilpass arbeidssted
G	Manglende oppmøte	Den det gjelder	Avhengig av bakenforliggende grunn
H	Forstyrrelser	Alle	Finne et annet sted å jobbe eller bli kvitt forstyrrelsen.
I	Strømbrudd	Kommunen	Tvinne tomler

Realisering av prosjektet innenfor den gitte rammen

Vi anser det som høyst sannsynlig at vi kan realisere prosjektet innenfor de gitte rammer. Dersom prosjektet skulle feile, vil det mest sannsynlig være på grunn av tidsbegrensing.

Ytterligere presisering og/eller avgrensing av prosjektet

Prosjektet er godt spesifisert i utgangspunktet.

Suksessfaktorer og trusler mot suksess

Dersom en del av prosjektet vil ta for lang tid å utvikle, eller noe utstyr som vi er avhengige av svikter, eller vi har behov for noe utstyr/programvare som blir for kostbart, kan prosjektet være i fare for å mislykkes.

Mulige risikoelementer, sikkerhetsaspekter og miljøpåvirkning

Det er viktig å ha datasikkerhet i fokus ettersom vi skal behandle brukeropplysninger som passord, brukernavn og e-post, det er også viktig med hensyn på å hindre tilgang til styring av utstyret av uvedkommende. Det er derfor viktig at vi benytter krypterte forbindelser der vi kan, samt. kryptering av passord.

5.6 Framdriftsplan – styring av prosjektet

Hovedplan

Gjennomføringen vil starte med en kort periode hvor vi klart definerer hvilke arbeidsoppgaver som er kritiske for fremdriften av prosjektet og hvilke som kan nedprioriteres og utsettes til senere. Etter vi har opprettet en klar liste over hva som er kritisk og hva som ikke er det, kan vi begynne å velge ut hvilke verktøy vi synes er nødvendig for å utvikle de forskjellige komponentene til applikasjonen.
Etter verktøy er valgt kan vi begynne å delegere de ulike oppgavene til gruppemedlemmene for gjennomføring, og senere gjennom arbeidsprosessen, testing. For de forskjellige komponentene vil ulike arbeidsmetoder bli utført. Noen av komponentene vil bli hovedsakelig utviklet av én person, og noen av komponentene vil ha tett samarbeid mellom gruppemedlemmene.
Fullføringen av hver komponent vil være en milepæl i prosjektet, en komponent er fullført når den oppfyller funksjonaliteten som er nedsatt i kravspesifikasjonen.

Styringshjelpemidler

Jira vil bli brukt for å holde en oversikt over alle de forskjellige oppgavene som må utføres for å fullføre prosjektet. Nye oppgaver kan legges til når som helst, så det er ikke nødvendig å ha alt helt klart før man setter i gang.
Confluence vil bli brukt for å planlegge tidsbruk og forskjellige detaljer tidlig i prosjektet og dokumentere detaljer senere i prosjektet som en wiki side. Se vedlegg 1.

Utviklingshjelpemidler

Git vil bli brukt som et utviklingshjelpemiddel hvor dens oppgave vil være å holde styr på versjoner av programvaren vi skal utvikle og samtidig styrke samarbeidet mellom utviklerne fordi den tillater at hvert gruppemedlem har tilgang til den siste versjonen av softwaren.
Netbeans vil bli brukt til å utvikle selve softwaren.

Intern kontroll – evaluering

Mål vil bli representert i form av epics i Jira og når alle oppgavene registrert under en epic er fullført vil være kriteriene på at et mål er nådd. Etter hver sprint vil det bli presentert et burndown diagram som viser fremgangen gjennom sprinten.

5.7 Beslutninger – beslutningsprosess

På grunn av arbeidsmetoden (SCRUM) vi har valgt å bruke for å arbeide med prosjektet har vi litt større frihet med enkelte beslutninger. I stedet for å planlegge hele omfanget til oppgaven i starten kan vi heller velge å prioritere de forskjellige komponentene til applikasjonen etter viktighet og relevans til kravspesifikasjonen. Prioriteringen vi har tatt blir nevnt i pkt. 5.3 og kan bli utvidet dersom vi oppdager at vi har nok tid til overs etter vi har fullført de prioriterte komponentene. Utvidelsene vil være å legge til støtte for flere enheter eller forbedre løsningen.

Store beslutninger underveis i oppgaven vil bli behandlet etterhvert som de dukker opp. Avhengig om hvordan beslutningene blir behandlet internt i gruppen kan vi blande inn veileder for oppklaring.

6 PLANLAGTE MØTER OG RAPPORTER

6.1 Møter

Møter med styringsgruppen

Vi skal ha møter hver 14. dag fra og med fredag 26.01 kl. 13.00.

Prosjektmøter

Daglig standup SCRUM møte. Hensiktene til møtene vil bli å diskutere framgang og eventuelle problemer. Oppdaget problemer i møtene vil bli avtalt og løst på et senere tidspunkt. Møtene vil kun bli brukt til å kartlegge framgang og potensielle problemer.

6.2 Periodiske rapporter

Framdriftsrapporter (inkl. milepæl)

Jira vil bli brukt for å loggføre de ulike arbeidsoppgavene i prosjektet noe som vil framgangen og gi mulighet for å beregne arbeidseffektivitet og total tidsbruk. Issues i Jira kan samles under en Epic som tilsvarer en milepæl. Confluence vil bli brukt for å dokumentere ulike beslutninger, innhentet informasjon og verktøy tatt i bruk.

7 PLANLAGT AVVIKSBEHANDLING

- Kritiske feil som oppstår kan resultere i en rollback av programvaren.
- Skade på utstyr vil bli rapportert til oppdragsgiver.
- Om feil på utstyr oppstår vil dette bli rapportert til oppdragsgiver.

8 UTSTYRSBEHOV/FORUTSETNINGER FOR GJENNOMFØRING

Tilgang til vislabben og de ulike enhetene i labben, slik som projektorene.
Server til database og webside.

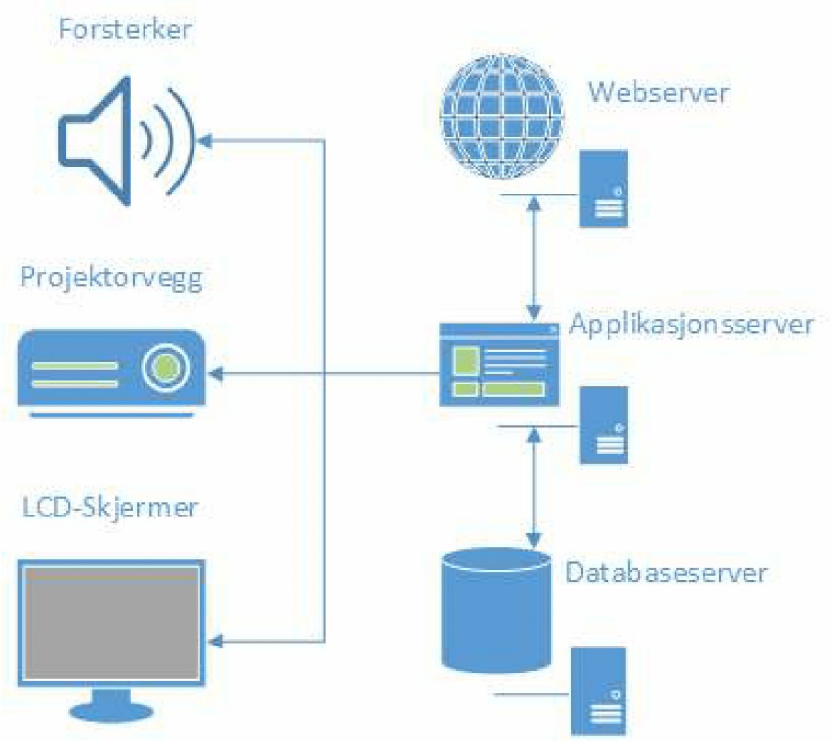
9 REFERANSER

10 VEDLEGG

Vedlegg 1	Planlagt framgang
Vedlegg 2	Dataflytdiagram

Vedlegg 1: Planlagt fremdrift

Kristoffer Rogne Erik Haram Nygård Thomas Skarshaug Todal



Vedlegg 2:Dataflytdiagram

Vedlegg D: Arbeidslogg

Dato	Navn	
Uke 1		
29.01.18	Erik	Skrev oppgaver på JIRA, GUI, kommunikasjon.
29.01.18	Erik	Undersøkte Spring rammeverket for prosjektet.
29.01.18	Erik	Utførte Spring øvelser for å få en enkel forståelse.
29.01.18	Thomas	Skrev oppgaver på JIRA.
29.01.18	Thomas	Undersøkte Docker og Microsoft Azure.
29.01.18	Thomas	Startet en VM på labben med Docker (linux maskin).
29.01.18	Kristoffer	Skrev oppgaver på JIRA.
29.01.18	Kristoffer	Undersøkte SonarQube.
29.01.18	Kristoffer	Fekk en lokal versjon av SonarQube til å fungere.
30.01.18	Erik	Endret på oppgaver i JIRA etter tilbakemelding fra veileder.
30.01.18	Erik	Startet sprint, fokus på kommunikasjon og undersøkelser.
30.01.18	Erik	Undersøkelser på: Spring, Docker, SonarQube og Maven.
30.01.18	Thomas	Endret på oppgaver i JIRA etter tilbakemelding fra veileder.
30.01.18	Thomas	Installerte SonarQube på VM som ble lagd i går, port 9000.
30.01.18	Kristoffer	Endret på oppgaver i JIRA etter tilbakemelding fra veileder.
30.01.18	Kristoffer	La til “user stories” om gruppering, valg og administrering.
30.01.18	Kristoffer	Las igjennom kommunikasjonsprotokollen til Barko F22.
30.01.18	Kristoffer	Startet sprint.
30.01.18	Kristoffer	Fortsatte undersøkelse av SonarQube.

31.01.18	Thomas	Jobbet med NetBeans tilkobling til SonarQube
31.01.18	Thomas	Lagde egne brukere for SonarQube login.
31.01.18	Kristoffer	Åpnet porten til SonarQube på VM
31.01.18	Kristoffer	Fikk SonarQube til å fungere over internett, manuelt oppsett.
31.01.18	Kristoffer	Undersøkte muligheter for å automatisere SonarQube.
01.02.18	Erik	Fortsatte undersøkelse av Spring.
01.02.18	Erik	Lagde funksjonell demo i Spring.
01.02.18	Erik	Diskusjon om implementasjon om GUI. Rammeverk ønsket.
01.02.18	Erik	Undersøkte WAF, Vaadin og SpringMVC.
01.02.18	Thomas	Fortsette undersøkelse av Docker.
01.02.18	Thomas	Fant nyttige ressurser til Docker.
01.02.18	Thomas	Satt opp docker på VM, tilkobling feilet.
01.02.18	Kristoffer	Laget på/av funksjon til Barko F22 i java.
01.02.18	Kristoffer	Laget mute funksjon til Barko F22 i java.
01.02.18	Kristoffer	Skrev flere oppgaver om hvordan vi skal løse kommunikasjon.
02.02.18	Erik	Diskuterte problematikken rundt å vise kjøretid
02.02.18	Erik	Satt opp malen til bachelor rapporten.
02.02.18	Thomas	Startet ny VM med Docker, kunne koble til.
02.02.18	Thomas	Docker kjører på VM og klar til å motta kontainere.
02.02.18	Kristoffer	Skrev om SonarQube på Confluence
02.02.18	Kristoffer	Skrev flere oppgave på JIRA

02.02.18	Kristoffer	Såg på spesifikasjonene til LCD skjermene.
Uke 2		
05.02.18	Erik	Lagde loggside for veke 2.
05.02.18	Erik	Skrev punkt i rapporten.
05.02.18	Erik	Skrev oppgaver på JIRA.
05.02.18	Erik	Vi ble ferdig med den første sprinten tidlig.
05.02.18	Thomas	Mer undersøkelse om Docker.
05.02.18	Thomas	Endret på oppgaver i JIRA.
05.02.18	Kristoffer	La til nye oppgaver i JIRA.
07.02.18	Erik	Diskuterte mulige problemer og utfordringer.
07.02.18	Erik	Planla diskusjonspunkt for samtalen med veileder.
07.02.18	Thomas	Diskuterte mulige problemer og utfordringer.
07.02.18	Thomas	Planla diskusjonspunkt for samtalen med veileder.
07.02.18	Kristoffer	Diskuterte mulige problemer og utfordringer.
07.02.18	Kristoffer	Planla diskusjonspunkt for samtalen med veileder.
08.02.18	Erik	La til nye oppgaver på JIRA.
08.02.18	Erik	Laget oversikt over oppgaver for å se omfanget.
08.02.18	Thomas	La til nye oppgaver på JIRA.
08.02.18	Thomas	Laget oversikt over oppgaver for å se omfanget.
08.02.18	Kristoffer	La til nye oppgaver på JIRA.
09.02.18	Erik	Ble ferdig med oppgave oversikta.
09.02.18	Erik	Fikset oppgave linking og avhengighet i JIRA

09.02.18	Thomas	Omorganiserte og gav nye navn til oppgaver i JIRA.
09.02.18	Kristoffer	La til navnelapp på alle oppgaver i JIRA
Uke 3		
12.02.18	Erik	Fikset JIRA oppgaver
12.02.18	Erik	Startet sprint
12.02.18	Erik	Laget utkast av wireframes
12.02.18	Erik	Startet å lære spring
12.02.18	Thomas	Redefinerte oppgave lenker i JIRA
12.02.18	Thomas	Startet sprint
12.02.18	Thomas	Arbeidet med Lampe kjøretids funksjon
12.02.18	Thomas	Begynte arbeid med Command klassen
12.02.18	Kristoffer	Fikset JIRA oppgaver
12.02.18	Kristoffer	Startet sprint
12.02.18	Kristoffer	Startet å lage kommunikasjons handterer til Barko F22
15.02.18	Erik	Fortsatte arbeid med spring
15.02.18	Erik	Bygde ei veldig enkel nettside med spring.
15.02.18	Thomas	Forbedret Command klassen
15.02.18	Kristoffer	Fullførte kommunikasjons handteringa for Barko F22.
16.02.18	Erik	Såg nærmere på hvordan vi skal bruke spring
16.02.18	Thomas	Arbeidet med implementasjon av Comand klassen
16.02.18	Kristoffer	Lagde oversettelses klasse for svaret fra Barko F22.
Uke 4		

19.02.18	Erik	Arbeidet med bachelor rapporten.
19.02.18	Thomas	Arbeidet med Command klassen og alle klasser som arver fra den.
19.02.18	Kristoffer	Satt opp Jenkins til å bygge og sende koden til SonarQube for analyse når endringer blir gjort i git.
21.02.18	Thomas	Fikset bugs som SonarQube pekte ut.
21.02.18	Thomas	La arbeid med Command inn med hoved arbeidet.
21.02.18	Thomas	Startet å arbeide med Dockerfile for å lage en kontainer.
21.02.18	Kristoffer	Utførte teste med Jenkins og SonarQube.
21.02.18	Kristoffer	Fikset flere bugs i kommunikasjons håndteringen.
22.02.18	Erik	Implementerte HTML
22.02.18	Erik	Skrev grunnleggende CSS
22.02.18	Erik	Skrev grunnleggende JavaScript
22.02.18	Erik	Hjelpte til med å sette opp grunnleggende nettside med spring
22.02.18	Thomas	Laget Dockerfile, undersøkte hvilken parametere som trengs.
22.02.18	Kristoffer	Laget enkel nettside som kaller til kommunikasjons håndtereren.
22.02.18	Kristoffer	Fikset bug der kommunikasjons håndtereren gjentart kommandoer uten å vente.
22.02.18	Kristoffer	Skrev noen linjer med JavaScript.
23.02.18	Kristoffer	Fikset små bugs i kommunikasjon håndtereren.
Uke 5		
26.02.18	Erik	Startet sprint.
26.02.18	Erik	Skrev i rapporten.
26.02.18	Erik	HTML/CSS/JavaScript koding.

26.02.18	Thomas	Starte sprint.
26.02.18	Thomas	Refaktorerte Command til at alle felles kommandoer arver fra en felles klasse.
26.02.18	Kristoffer	Starte sprint.
26.02.18	Kristoffer	Diskuterte design av nettside med gruppe.
26.02.18	Kristoffer	Omarbeidet kommunikasjons tråden sin struktur.
27.02.18	Kristoffer	Laget timere for å opprettholde kommunikasjonsprotokollen sine tidsrammer.
27.02.18	Kristoffer	Laget "idle" funksjonalitet
27.02.18	Thomas	Fortsatte på refaktorering av Command
28.02.18	Thomas	Fortsatte på refaktorering av Command
28.02.18	Kristoffer	Skrev den første JUnit testen.
01.03.18	Erik	Skrev JavaScript
01.03.18	Thomas	Ble ferdig med Command klassen
01.03.18	Kristoffer	Fikset bugs med tidsrammer, kommunikasjon og mottak av respons.
02.03.18	Erik	Laget en JavaScript worker som henter info.
02.03.18	Thomas	Startet arbeid med Docker igjen
02.03.18	Thomas	Laget noe grafikk som skal bli brukt i GUIen
02.03.18	Kristoffer	Fikset en stor bug med sending av kommando, var ikke pålitelig.
Uke 6		
05.03.18	Erik	Ryddet or refaktorerte i koden
05.03.18	Thomas	Fortsatte arbied på docker, lagde fungerende Dockerfile med hjelp av Kristoffer
05.03.18	Kristoffer	Lagde enkel kommandolinje simulator. Den skal simulere mottak og sending av kommandoer fra enhetene.

05.03.18	Kristoffer	Testing og fiksing av bugs i kommunikatoren
06.03.18	Erik	Ryddet opp i koden
06.03.18	Erik	Forbedret JavaScript
06.03.18	Thomas	Ordnet handtering av exceptions.
06.03.18	Kristoffer	Startet arbeid på State Pattern for å kjøre kommunikasjonen.
07.03.18	Kristoffer	Utvidet simulatoren slik at den svarer mer rett (echo).
09.03.18	Erik	Skrev noe Java kode (Spring, Controller)
09.03.18	Erik	Fjernet worker fra JavaScriptet.
09.03.18	Erik	Skrev i rapporten
09.03.18	Kristoffer	Lagde GUI i JavaFX til simulatoren. Kan no endre verdier og hvordan svaret blir sent.
09.03.18	Kristoffer	Fullførte første iterasjon av State Pattern
Uke 7		
12.03.18	Erik	Skrev nye oppgaver på JIRA
12.03.18	Erik	Startet ny sprint
12.03.18	Erik	Jobbet med HTML og CSS (GUI)
12.03.18	Thomas	Startet ny sprint
12.03.18	Thomas	Lagde/oppdaterte oppgaver på JIRA
12.03.18	Thomas	Undersøkte mulige database designs.
12.03.18	Thomas	Lagde MySql Docker kontainer på docker serveren vår.
12.03.18	Thomas	Installerte MySql Workbench
12.03.18	Kristoffer	Lagde/oppdaterte oppgaver på JIRA
12.03.18	Kristoffer	Lagde interface til Barko F22 projektoren slik at frontend

		og backend ikke henger sammen.
13.03.18	Erik	Skrev opp att CSSen fra grunnen av
13.03.18	Erik	Startet arbeid med responsivt design
13.03.18	Erik	Diskuterte databasestruktur.
13.03.18	Thomas	Startet arbeid med entitets diagram
13.03.18	Kristoffer	La till manglende tilstander i tilstands maskina. Inneheld no alle tilstander som trengs for kommunikasjon med Barko F22.
13.03.18	Kristoffer	Diskuterte databasestruktur.
14.03.18	Erik	CSS er no lettere å jobbe med, med tanke på responsivt design
14.03.18	Erik	Startet arbeid med tema og fargevalg på nettsida.
14.03.18	Thomas	Diskuterte farger og tema.
14.03.18	Thomas	Skrev i rapporten
14.03.18	Kristoffer	Diskuterte farger og tema.
14.03.18	Kristoffer	Skrev i rapporten
14.03.18	Kristoffer	Lagde visualisering av prosjektor oppsettet i HTML ved hjelp av Canvas elementet.
15.03.18	Erik	Små CSS fiksing
15.03.18	Erik	Delvis ombygging av oppsettet
15.03.18	Thomas	Fortsatte arbeid med entitets diagrammet.
15.03.18	Thomas	Startet å bygge database strukturen
15.03.18	Kristoffer	Bug Testing av tilstandsmaskinen, fikset problemer med tilkobling og brudd av kommunikasjon.
15.03.18	Kristoffer	Lagde JavaScript for å la bilder bli plassert basert på senter av bilde inne i canvaset.

16.03.18	Erik	La til includeHTML metoden, den lar deg lage en mal som alle sider får.
16.03.18	Kristoffer	La til visualisering når du kan trykke på eit element inne i canvaset (musepekeren endrer seg og en ring rundt elementet).
Uke 8		
19.03.18	Erik	Mindre CSS endringer
19.03.18	Erik	Hjelpte med å ordne mappestruktur.
19.03.18	Erik	Fikset noen bugs med HTML
19.03.18	Erik	Ordnet med JavaScript og Spring kontrollerene
19.03.18	Thomas	Skrev i rapporten
19.03.18	Kristoffer	Fullførte restrukturering av kommando.
19.03.18	Kristoffer	Lagde en dummy database klasse, som held på info om enheter, i java.
20.03.18	Erik	Implementerte JavaScript funksjoner som henter informasjon og viser den fram.
20.03.18	Erik	Utvidet projektor siden med HTML, CSS og JS.
20.03.18	Thomas	Jobbet med kobling fra applikasjon og database.
20.03.18	Kristoffer	La til funksjonalitet på canvas der den henter ut posisjonsinformasjon fra database og plasserer elementene i canvaset.
20.03.18	Kristoffer	La til JavaDoc på en mengde klasser for Barko F22
20.03.18	Kristoffer	Fikset problemer lagd av store og små bokstaver.
21.03.18	Erik	Skrev i rapporten
21.03.18	Thomas	Skrev i rapporten
21.03.18	Kristoffer	Skrev i rapporten
22.03.18	Erik	La til funksjonalitet for å kunne håndtere mer enn én enhet på nettsiden.

22.03.18	Thomas	La til alle entitets klassene i applikasjonen.
22.03.18	Kristoffer	Små endringer i interfaces og hvordan tilkoblinger skjer.
22.03.18	Kristoffer	Utvidet simulatoren til å kunne enkelt lage nye simulerte enheter i tillegg til den vi har.
23.03.18	Kristoffer	La simulatoren inn i hoved prosjektet for å kjøre noen tester.
23.03.18	Kristoffer	Utførte teste med docker.
Uke 9/10		
26.03.18 -> 06.04.18	Erik	Skrev i rapporten
26.03.18 -> 06.04.18	Thomas	Skrev i rapporten
26.03.18 -> 06.04.18	Kristoffer	Skrev i rapporten
Uke 11		
09.04.18	Erik	Startet ny sprint
09.04.18	Erik	Laget/oppdaterte oppgåver i JIRA
09.04.18	Erik	Implementerte GitFlow i prosjektet.
09.04.18	Erik	CSS/HTML arbeid.
09.04.18	Thomas	Startet ny sprint
09.04.18	Thomas	Fortsatte med database implementasjon.
09.04.18	Kristoffer	Implementerte GitFlow i prosjektet.
09.04.18	Kristoffer	Tok i bruk JavaSPI for å kunne gjøre prosjektet utvidbart.
09.04.18	Kristoffer	Separerte ut Barko F22 delene av prosjektet til et eget prosjekt.
10.04.18	Erik	Fikset dynamisk listen.
10.04.18	Erik	CSS arbeid

10.04.18	Thomas	Database (MySql) integrasjon inn i prosjektet.
10.04.18	Kristoffer	Fiksing av bygging, kjøring og pakking av prosjektet no som det er modulært.
10.04.18	Kristoffer	Fikset Barko F22 prosjektet til å kunne pakke det og arve fra vårt hoved prosjekt.
11.04.18	Erik	Laget dynamisk innstillinger side.
11.04.18	Erik	Fikset noen JavaScript funksjoner.
11.04.18	Thomas	Database (MySql) integrasjon inn i prosjektet.
11.04.18	Kristoffer	Laget egne java annotasjoner for å gjøre utviding av prosjektet enklere.
11.04.18	Kristoffer	Implementerte dei nye annotasjonene inn i Barko F22 pakken.
12.04.18	Erik	Startet arbeid med å implementere promise-chains i JavaScript
12.04.18	Thomas	Database (MySql) integrasjon inn i prosjektet.
12.04.18	Kristoffer	Laget vår første Spring tilkobling til databasen.
12.04.18	Kristoffer	Starte arbeid med Denon DN500AV pakken
13.04.18	Erik	Fullførte implementasjon av promise-chains
13.04.18	Kristoffer	Startet arbeid med get og put ned til databasen gjennom spring.
Uke 12		
16.04.18	Erik	Lagde log-in HTML og CSS
16.04.18	Erik	Fikset tilstands ikon.
16.04.18	Kristoffer	Fortsatte arbeidet i kontrollerene for henting og lagring av entiteter.
16.04.18	Kristoffer	Fikset kommunikasjons driverene slik at kun en blir lagd for hver entitet.
17.04.18	Erik	Forbedret designet på ikonene

17.04.18	Erik	Implementerte dei nye ikonene.
17.04.18	Thomas	Looking into sikker innlogging og passord kryptering
17.04.18	Thomas	Første forsøk på implementasjon av sikkerhet.
17.04.18	Kristoffer	Fikset visualisering fordi vi byttet fra Dummybasen til en faktisk database og variabelnavn er litt forskjellig.
18.04.18	Erik	CSS forbedringer
18.04.18	Erik	Forbedret logoen
18.04.18	Erik	Skrev i rapporten
18.04.18	Thomas	Startet på nytt med sikkerhet.
18.04.18	Kristoffer	Framgang på kontrollpanelet der en administrator kan legge til/endre og fjerne element fra databasen
18.04.18	Kristoffer	Flyttet device spesifikk side til modulene i stedet for i hoved applikasjonen.
19.04.18	Erik	Ryddet opp i CSS
19.04.18	Erik	Separerte ut knapp funksjonalitet til egne knapper. Enklere å sikre jevn oppførsel og letter for brukere å forstå
19.04.18	Erik	Implementerte funksjonalitet på instillings siden.
19.04.18	Thomas	Arbeid med sikkerhet og kryptering.
19.04.18	Kristoffer	Fikset problem som oppsto med Barko F22, hver tredje svar ble sendt med et ekstra linjeskift først. Gjentakelse av linjeskift blir ignorert.
19.04.18	Kristoffer	Framgang med gruppevalg gjennom en liten meny.
20.04.18	Erik	Fullførte gruppevalg funksjonaliteten.
20.04.18	Erik	Oppsett restrukturering
20.04.18	Thomas	Starte ny utgivelse, prøver å gi ut på docker.
20.04.18	Kristoffer	Fullførte gruppevalgs menyen.

20.04.18	Kristoffer	Startet med login, slik at personer uten rett tilgang ikke kan bruke nettsiden.
Uke 13		
23.04.18	Erik	Startet ny sprint
23.04.18	Erik	Ordnet oppgavene i sprinten.
23.04.18	Erik	Startet arbeid med å fikse små problemer med forskjellige ikon og bilder.
23.04.18	Thomas	Fullførte utgivelse.
23.04.18	Thomas	Startet ny sprint
23.04.18	Thomas	Skrev i rapporten
23.04.18	Kristoffer	Fullførte utgivelse.
23.04.18	Kristoffer	Skrev i rapporten
24.04.18	Erik	Implementerte visning av tilkoblings brudd.
24.04.18	Erik	Fikset bilde/ikon på enhets siden.
24.04.18	Erik	Fikset problemer med responsivt design.
24.04.18	Thomas	Implementerte Theater entitet.
24.04.18	Thomas	Oppdaterte en mengde oppgaver på JIRA.
24.04.18	Thomas	Starte å implementere applikasjonslogikk for Theatre.
24.04.18	Kristoffer	Login fungerer, låser ute personer som ikke har logget inn.
25.04.18	Erik	Skrev script for tilkoblings brudd.
25.04.18	Erik	La til støtte for Theatre i nettsiden.
25.04.18	Erik	Implementerte enhetsliste “forwarding”
25.04.18	Thomas	Fullførte applikasjons siden av Theatre.
25.04.18	Thomas	Skrev i rapport

25.04.18	Kristoffer	Fullførte login.
25.04.18	Kristoffer	Fikset slik av JavaScript fetch legger til login info i hver forespørsel.
25.04.18	Kristoffer	Laget kommandoer for Denon DN500AV
25.04.18	Kristoffer	Startet arbeid med å kunne pakke prosjektet gjennom Maven i stedet for Intellij sin bygging.
27.04.18	Erik	Fikset henting av enheter slik at den ikke er tom når noen av funksjonene kjører.
27.04.18	Thomas	Refaktorerte en mengde funksjoner i kontroller klassene.
27.04.18	Kristoffer	Refaktoriserte annotasjonene, interface og SPI til et eget bibliotek.
27.04.18	Kristoffer	Implementerte det nye biblioteket i alle prosjektene.
Uke 14		
30.04.18	Erik	Front-end logic for Theatre.
30.04.18	Erik	Startet å fiske kontroll sidene for enhetene.
30.04.18	Thomas	Refaktorering av API
30.04.18	Kristoffer	Fikset en bug med @DeviceSPI annotasjonen
30.04.18	Kristoffer	Forsøk på å laste inn moduler når programmet kjører
02.05.18	Erik	Konverterte kontroll sidene til instillingsliste.
02.05.18	Erik	Fikset bugs med API
02.05.18	Erik	Fikset ikon.
02.05.18	Thomas	Fullførte refaktorering og dokumentering av API
02.05.18	Thomas	Startet arbeid på kontroller til Denon DN500AV
02.05.18	Kristoffer	Jobbet med kommunikasjon i Denon DN500AV pakken. Den er asynkron.
03.05.18	Erik	Undersøkte og forsket med Bootstrap.

04.05.18	Erik	Fortsatte å redesigne sida med Boostrap, virker som om det er verdt det
04.05.18	Thomas	Hjelpte Erik med noen design valg.
04.05.18	Thomas	Fullførte Denon DN500AV kontrolleren.

Vedlegg E: Sprint evaluering møtereferat

26.01.2018
Diskusjonspunkt: (på forhånd)
- Alternative enheter til å teste applikasjonen med (Android nettbrett, iPad, iPhone, MacBook)
- Tilgang til lab. for alle gruppemedlemmer
- Siste-liten forprosjektrapport spørsmål

Notat:
- Gjennomføre ordentlig risikovurdering til forprosjektrapporten
- Ønske om å ta i bruk Docker i prosjektet
- Begynn å bruke Confluence til å koordinere prosjektet
- Ønske om å ta i bruk SonarQube i prosjektet
- Labben har 3D projektorer som kan taes inn i prosjektet dersom det er tid til slutt

09.02.2018
Diskusjonspunkt: (på forhånd)
- Trenger hjelp til Sprinter og "issues"
- Vise hva som er gjort til nå
- Trenger oppklaring om LCD-monitorene
- Trenger spesifikasjon til lydanlegget
- Diskuter Docker
- Diskuter Azure
- Spørsmål om Sprinter
- Confluence ser ut som det er knyttet til et annet JIRA-prosjekt

Notat:
- Diskutert muntlig, ingenting å notere

23.02.2018
Diskusjonspunkt: (på forhånd)
- Vise hva som er gjort til nå
- Snakke om hva som skal skje videre i prosjektet

Notat:
- Diskutert muntlig, ingenting å notere

09.03.2018
Diskusjonspunkt: (på forhånd)
- Det er oppført 2 innleveringsdatoer for bacheloroppgaven

Notat:
- Oppgaven har 2 datoer pga. Innleveringsplattformen, den skal leveres 01.06.2018

23.03.2018
Diskusjonspunkt: (på forhånd)
- Ingenting å ta opp

Notat:
- Lydanlegget skal ha følgende funksjonalitet:
  - På/av
  - Inngangskilde

    - Demping
    - Volum
- Vi skal lage egne bibliotek for de forskjellige enhetene, som skal importeres inn i applikasjonen som *.jar-filer
- DevOps
- Sjekke opp hvordan vi skal henvise til en annen rapport som kilde

09.04.2018
Diskusjonspunkt: (på forhånd)
- Ingenting å ta opp

Notat:
- Finne ut av hvorfor feilmeldingen: "Could not find or load no.rogne.Main" kommer frem når én av *.jar-filene (GeneralDictionary) mangler fra classpath
- Finne en ledig IP-adresse som lydanlegget kan bruke

20.04.2018
Diskusjonspunkt: (på forhånd)
- Vise hva som er gjort til nå
- Diskutere IP-adressen til lydanlegget

Notat:
- Fikse farge på fonten til netsiden
- Lage ikon for frakoblet tilstand
- Håndtere hva som skal skje dersom en enhet ikke er tilkoblet
- Bakgrunnsfarge på knapper
- Implementere teaterfunksjonalitet

04.05.2018
Diskusjonspunkt: (på forhånd)
- Pakking av prosjektet (levering)
- Admin funksjonalitet og bruker/admin skille

Notat:
- En bruker skal kunne sette et eget navn til en enhet
- Admin fuksjonalitet er viktigere enn å skille mellom bruker og admin
- Levering av prosjektet med github-lenke i tillegg til ferdig kompilert applikasjon på Docker
- Fortsett implementasjonen av Bootstrap

14.05.2018
Diskusjonspunkt: (på forhånd)
- Ingenting å ta opp

Notat:
- Diskutert muntlig, ingenting å notere

25.05.2018
Diskusjonspunkt: (på forhånd)
- Ingenting å ta opp

Notat:
- Diskutert muntlig, ingenting å notere

Vedlegg F: Web-API

- /
- /user
- /admin
- /login
- /device
  - Param: int id
- /devicesettings
- /api
  - /main
    - /powerOn
      - Param: int id
    - /powerOff
      - Param: int id
    - /mute
      - Param: int id
    - /unMute
      - Param: int id
    - /powerState
      - Param: int id
    - /supported
  - /BarcoF22
    - /mute
      - Param: int id
    - /unMute
      - Param: int id
    - /powerState
      - Param: int id
    - /lampStatus
      - Param: int id
      - Param: int lampNum
    - /lampRuntime
      - Param: int id
      - Param: int lampNum
    - /getContrast
      - Param: int id
    - /getBrightness
      - Param: int id
    - /getThermal
      - Param: int id
    - /getProjectorRuntime
      - Param: int id
    - /testImage
      - Param: int id
      - Param: int image
    - /setContrast
      - Param: int id
      - Param: int value

- /setBrightness
  - Param: int id
  - Param: int value
- /DenonDN500AV
  - /powerOn
    - Param: int id
  - /powerOff
    - Param: int id
  - /getPower
    - Param: int id
  - /getMute
    - Param: int id
  - /mute
    - Param: int id
  - /unMute
    - Param: int id
  - /getVolume
    - Param: int id
  - /setVolume
    - Param: int id
    - Param: int value
  - /setSource
    - Param: int id
    - Param: String value
  - /getSource
    - Param: int id
  - /getSources
    - Param: int id
- /devicegroup
  - /getall
  - /add
    - From data
  - /setifdefault
    - Param: Integer[] ids
    - Param: boolean default
  - /adddevices
    - Param: int id
    - Body: Device[] deviceArray
  - /update
    - Form data
  - /remove
    - Param: int id
- /theatre
  - /getall
  - /add
    - Form data
  - /update
    - Form data
  - /getgroups
    - Param: String theatrename

- /getdevices
  - Param: String theatrename
- /remove
  - Param: int id

- /device
  - /getall
  - /types
  - /getone
    - Param: Optional<int id>
    - Param: Optional<String ipAddress>
  - /add
    - Form data
  - /update
    - Form data
  - /remove
    - Body: Device[] deviceArray
- /user
  - /getall
  - /getcurrent
    - Principal principal
  - /getone
    - Param: Optional<int id>
    - Param: Optional<String username>
    - Param: Optional<String email>
  - /getroles
  - /add
    - Form data
  - /update
    - Form data
  - /remove
    - Param: int id

Vedlegg G: Nye moduler

Nye moduler til Vislab. plattformen

For å lage nye moduler er det noen punkt som må følges.

Krav til java:

- Den nye modulen må ha "VislabDevices" som dependency gjennom maven. Dette kan oppnås med å kjøre "install" maven kommandoen på "VislabDevices" prosjektet, dermed blir det lagt til i ditt lokale maven repository.
- Alle klasser som implementerer Projector eller Device interface må bli annotert med @ProjectorSPI eller @DeviceSPI
- Klasser som implementerer Projector eller Device må kunne instansieres med en tom konstruktør.
- Pakkenavnet på modulen må starte med grunnpakke navn "vislab"
- Modul jar filen må legges inn i plugins mappa til VislabController plattformen for at den skal bli funnet.
- Avhengigheter som ikke er med i VislabController, må legges til i plugin mappen i tillegg til den nye modul jar filen.

Om det ønskes å ha mer kontroll tilgjengelig for enheten som blir lagt til støttar vi og å lage en egen kontrollpanelside gjennom Spring WAF. Kontrolleren som skal handtere kall må arve fra DeviceManager, tilgjengeleg i et eget lite bibliotek. Denne klassen håndterer alle enheter og aktive drivere til dem på ein slik måte at alle sider henter fram samme instanser gjennom getDevice(int id) metode, som returnerer rett enhets side.

For å bruke hovedtemaet til nettsida er det to element som burde linkest til:

templatescript.js: Script som legger inn navigasjonsmeny, gir menyen funksjon og legger til nødvendige parametere som gjør det mulig å gjøre HTTP requests til egen side. Templatescript betraktes også som hovedskriptet som implementerer klasser, variabler og funksjoner som blir brukt over hele nettsiden.

theme.css: CSS som inneholder base farge valg og enkel utforming.

Eksempel prosjekt for ny modul tilgjenglig[9], under "Template"

Vedlegg H: Fil- og mappestruktur

Kildekode java (.java filer):

"BarcoF22Controller/src/main/java/vislab/no/ntnu/" og dets undermapper
"VislabDevices/src/main/java/vislab/no/ntnu/" og dets undermapper
"VislabController/src/main/java/vislab/no/ntnu/" og dets undermapper
"DenonDN500-AVController/src/main/java/vislab/no/ntnu/" og dets undermapper

HTML, JavaScript og CSS:

"BarcoF22Controller/src/main/resources/static" og dets undermapper
"VislabDevices/src/main/resources/static" og dets undermapper
"VislabController/src/main/resources/static" og dets undermapper
"DenonDN500-AVController/src/main/resources/static" og dets undermapper

Bilder:

"VislabController/src/main/resources/static/images"

Kompilerte jars

"BarcoF22Controller/Vislab/plugins"
"VislabController/Vislab" og dets undermapper
"DenonDN500-AVController/Vislab/plugins"

Interne konfigurasjons filer:

"VislabController/src/main/resources/application.properties"

SQL script + konfigurasjons filer + README:

SQL: "VislabController/data/init.sql"
README: "VislabController" og "VislabController/Vislab"
Konfigurasjons fil (application.properties): "VislabController" og "VislabController/Vislab"
SSL Keystore (ssl-server.jks): "VislabController" og "VislabController/Vislab"
Dockerfile: "VislabController" og "VislabController/Vislab"
Docker-compose.yml: "VislabController" og "VislabController/Vislab"

Ekstra informasjon

Alle prosjekter (untatt VislabDevices) flytter alle relevante filer for program oppstart inn i ei "Vislab" undermappa i grunnmappa til prosjektet når dei blir kompilert med maven. Elementene inne i denne mappen er strukturert slik som dei skal settes sammen.

Vedlegg I: README.txt

```
Ready to deploy structure can be found in the "Vislab" sub-directory

Launching the Vislab java program:

All commands assume that your location is the folder where
VislabController.jar is.



CONFIG:

There are config options available in the application.properties file:

"MYSQL_ROOT_PASSWORD"  is the password for the user "root" in the mysql
database

"MYSQL_DATABASE" is the name of the database

"MYSQL_USER" is the name of the user our program will use to access
data and tables.

"MYSQL_PASSWORD" is the password for the user our program uses.

"SQL_ADDRESS" is the address for the database to use. When launched
with docker compose, a container name is a valid address

"SQL_PORT" the port for the database.


SSL config. Replace fields and file ass neccesary.


"server.ssl.key-alias" name for the keystore to use

"server.ssl.key-password" password for the keystore

"server.ssl.key-store" location of file. (IMPORTANT: must be prefixed
with "file:")

"server.ssl.key-store-provider" who provided the keystore

"server.ssl.key-store-type" what type the keystore is



STARTING

Docker compose is configured  to work out-of-the-box, but changes are
```

HIGHLY recommended, as username, passwords and databasename are weak and non descriptive.

The first startup will use the config file to define what should be used later on.

The MySql database conatiner and the java program is on a private network that can only be accessed through the open ports the docker-compose.yml,

the containers can use this network to communicate and thus dont need to open the database to external access.

"docker-compose up -d" will build and start two containers, one with our java program and one MySql 5.7 database.

STOPPING

"docker-compose down" will stop both containers

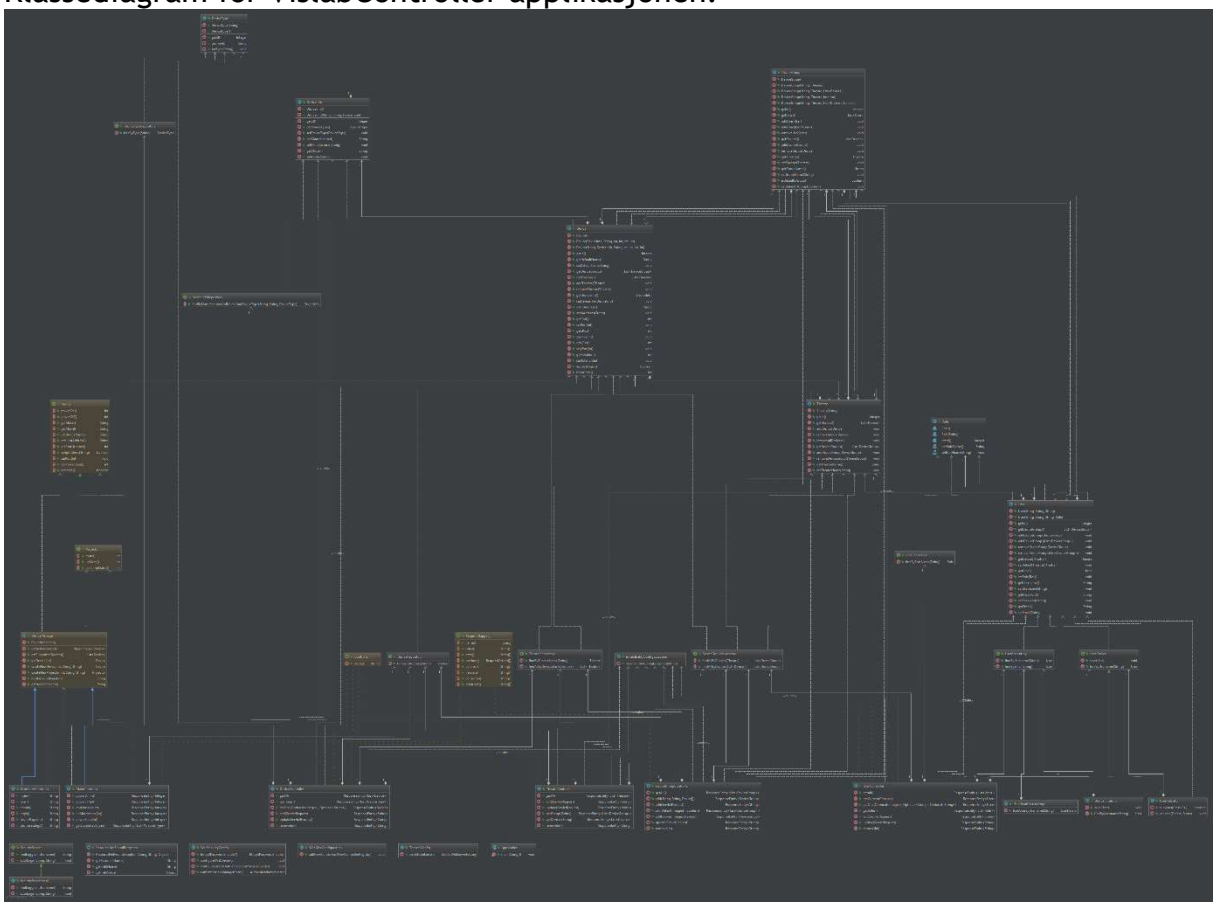
"docker rmi opt1:ver" will remove the given images (replace “opt1” with appropriate image name and “ver” with appropriate version.

The MySql database container stores its info in a Docker Volume so information is not lost when rebuilding the image/container.

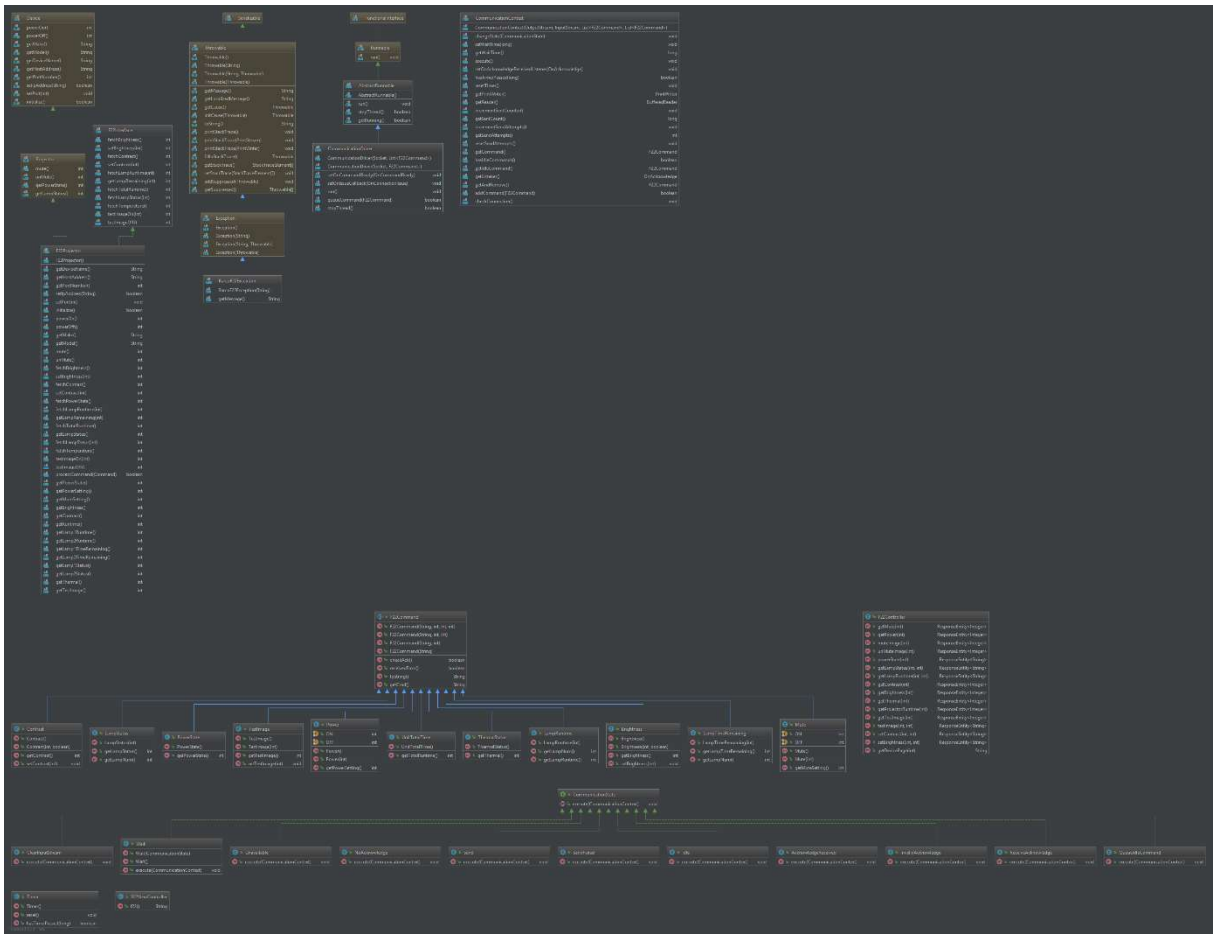
"docker volume rm VOLUME_NAME" can be used to remove the volume to get a fresh database. ("docker volume ls" to see all the volumes.

Vedlegg J: Klassediagram

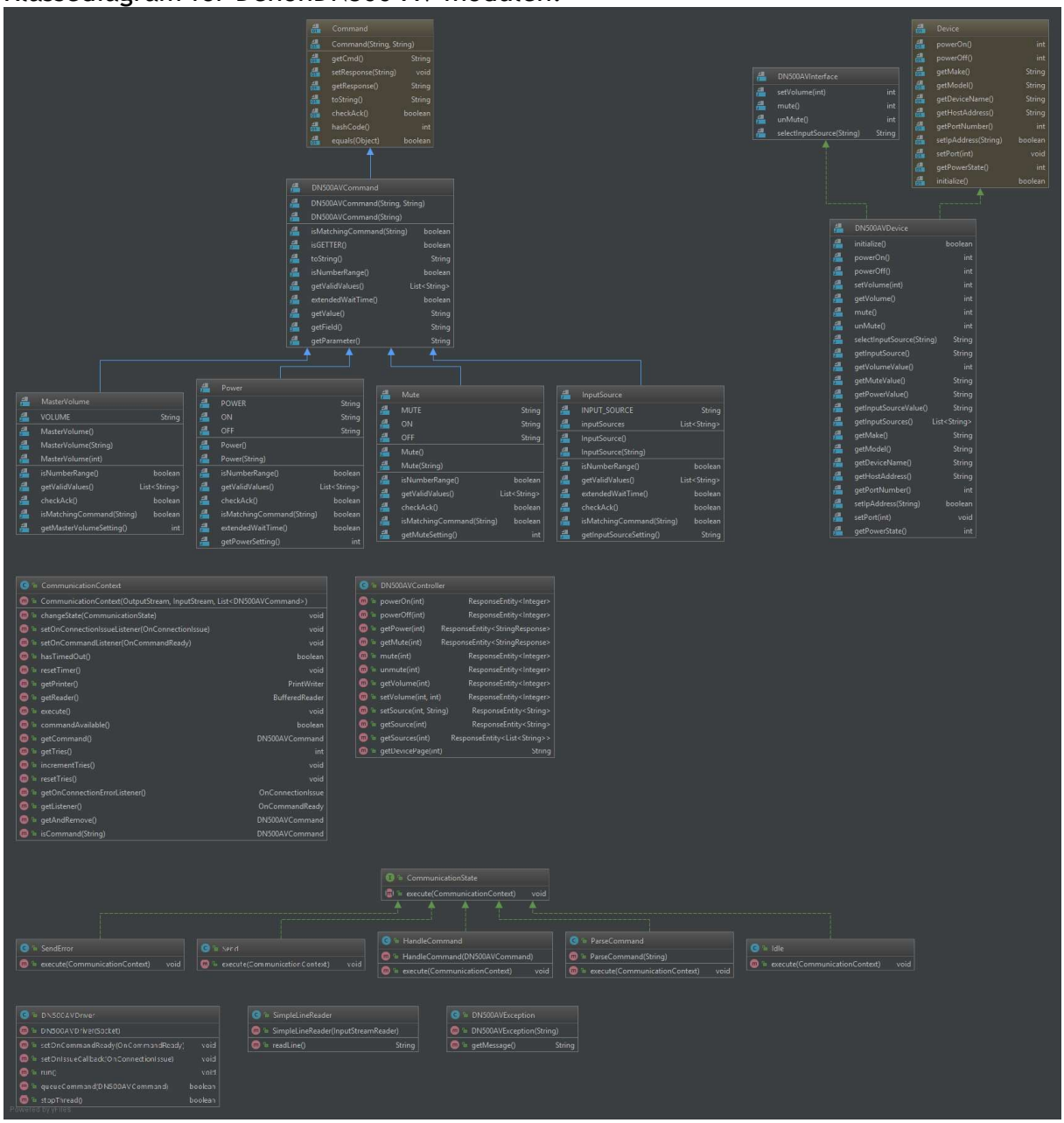
Alle diagrammene er lagt ved i *.zip-fil sammen med kildekode.
Klassediagram for VislabController applikasjonen.



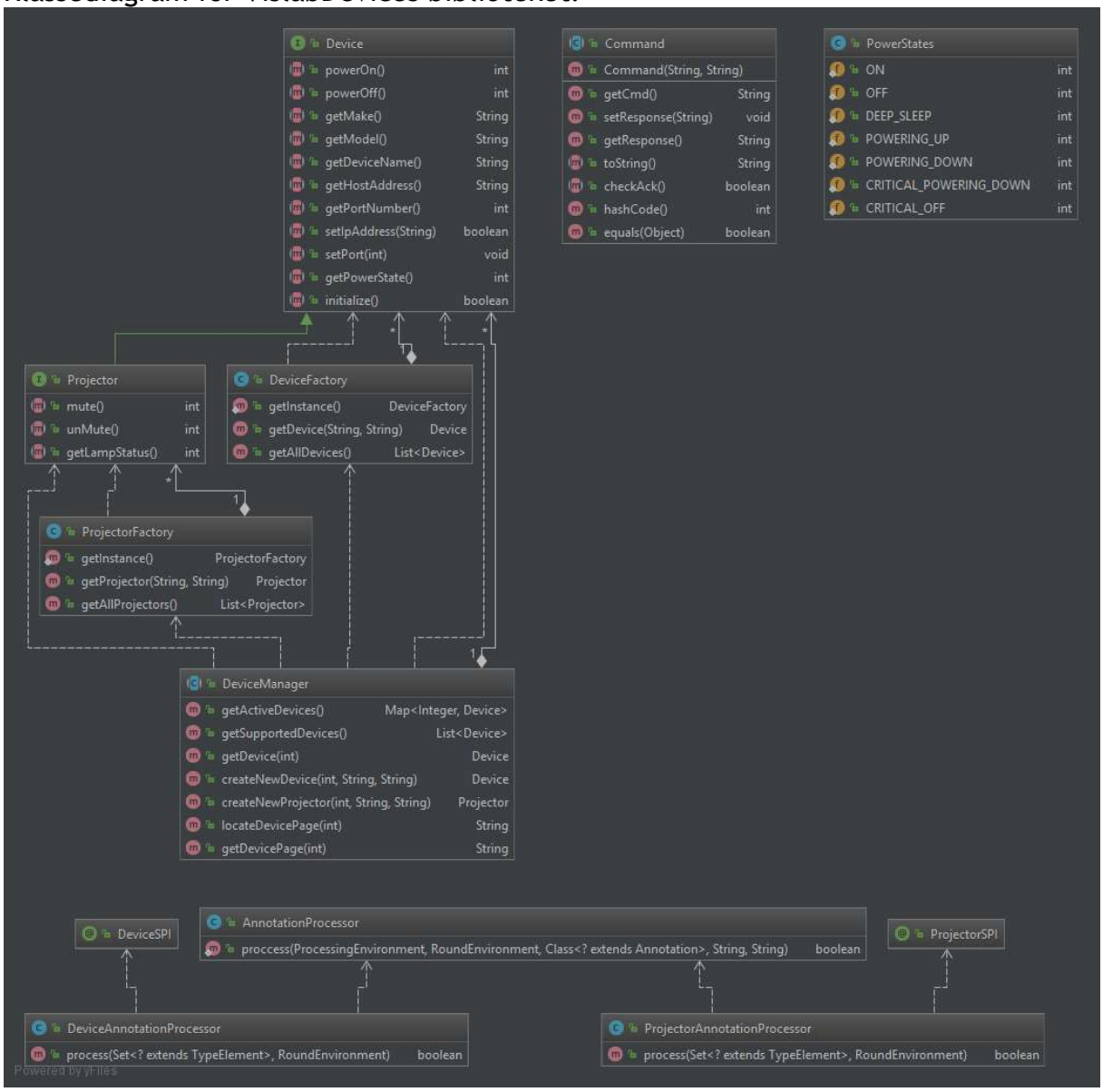
Klassediagram for BarcoF22Controller modulen.



Klassediagram for DenonDN500-AV modulen.



Klassediagram for VislabDevices biblioteket.



Klassediagram for Template prosjektet.