Fredrik Mediå

Bacheloroppgave

Fra data til TV-grafikk

En applikasjon som dynamisk håndterer data ved bruk av flytdiagram

Mai 2020

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Bacheloroppgave

2020

NTNU

Kunnskap for en bedre verden

Fredrik Mediå

Fra data til TV-grafikk

En applikasjon som dynamisk håndterer data ved bruk av flytdiagram

Bacheloroppgave
Mai 2020

NTNU
Norges teknisk-naturvitenskapelige universitet.
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

Dataingeniørstudiet på NTNU har lært meg mye om utvikling, prosjekthåndtering og samarbeid. Det har også lært meg å sjonglere flere oppgaver samtidig. Før jeg begynte på studiet, jobbet jeg som frilans kameraoperatør i TV- og eventbransjen. Det var her min fascinasjon for grafikk begynte å spire.

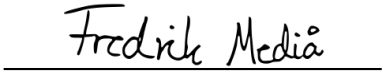
Når NEP Norway AS sin avdeling i Trondheim etterlyste programvare for å styre TV-grafikk, visste jeg at jeg måtte benytte bacheloren til å utvikle et slikt system. Prosessen har vært lærerik og utfordrende, og siden jeg har skrevet oppgaven alene, har jeg måtte begi meg ut på nye og ukjente utfordringer.

Denne oppgaven hadde ikke vært en realitet uten oppdragsgiveren min. Derfor ønsker jeg først å takke NEP Norway, avdeling Trondheim ved Daniel Hamstad og Stig Roar Aftret, for å ha gitt meg tilliten til å utvikle denne programvaren. De har bidratt med kontorplass og vært svært behjelpelig med å svare på spørsmål.

Deretter vil jeg takke en av hovedaktørene innen scoringsystemer for skytesport, Megalink AS, for deres imøtekommenhet og hjelp.

En stor takk rettes også til min veileder, Helge Hafting, for å ha gitt gode svar på spørsmålene mine i alle steg av prosessen.

Til slutt vil jeg takke alle som har lest korrektur: foreldrene mine, Mona og Trygve Karstensen, Mona Ullah, Mari Teiler-Johnsen, Petter Sakrihei Storvik, og samboeren min Anne-Maren Karlberg.



Fredrik Mediå
20. mai 2020, Trondheim

Oppgave

Denne oppgaven gikk ut på å lage en frittstående skrivebordsapplikasjon – NEP:GraphicsRouter, for NEP Norway AS. Skrivebordapplikasjonen er tilpasset deres behov for å håndtere grafikk i lavbudsjettsproduksjoner. Opprinnelig var oppgaven å håndtere grafikk for langrenn med hovedfokus på nøyaktighet i klokke. Dette ble imidlertid byttet ut med 10 meter luftpistol med hovedfokus på sanntidsdata, på grunn av praktiske hensyn for oppdragsgiver. Likevel forble konseptet rundt applikasjonen fremdeles den samme. Applikasjonen skal kunne fungere i flere sportsgrener, og skal kun være avhenging av hvordan grafikkoperatøren behandler data. Norgesmesterskapet i 10 meter luftpistol som skulle arrangeres på Ørlandet mars 2020, brukte Megalink AS som leverandør av digitale blinker. Applikasjonen måtte derfor kommunisere med Megalink AS sine programmer. Applikasjonen måtte også kunne håndtere ruting av data fra Megalink til riktig plass i grafikken hos Singular.Live.

For at en grafikkoperatør skal kunne gjøre jobben sin under en liveproduksjon, må knapper og tekstfelt være lett tilgjengelige. Grafikkoperatøren må derfor kunne bygge sitt eget kontrollpanel basert på egne behov. Knapper og tekstfelt skal knyttes til datarutingen.

Opprinnelig skulle utvikling av en modul til Companion være en ekstraoppgave dersom tiden strakk til. Companion er en applikasjon som har støtte for kontrollering av mange forskjellige enheter som prosjektorer, lydmiksere og grafikkmaskiner. Grafikkmotoren til NEP Norway var ikke en av disse. Imidlertid visste det seg å være et sterkt ønske om å kunne samkjøre grafikk med andre enheter i produksjonen. Companionmodulen for å styre grafikk ble dermed endret til å få prioritet foran kontrollpanelet.

Utviklingen til denne oppgaven startet allerede som høstprosjekt i faget TDAT3022 Systemutviklingsprosjekt. Høstprosjektet gikk ut på å kommunisere med en Microsoft SQL-database til eTiming fra en MacOS datamaskin, og videre sende informasjonen til en grafikkmotor. Dette ble gjort gjennom å utvikle en applikasjon med ElectronJS og Create-React-App.

For å oppsummere ble oppgaven min å lage en skrivebordsapplikasjon for å styre data fra Megalink til NEP sin grafikkmotor, til bruk under NM i luftpistol, samt å lage en modul til Companion.

Sammendrag

Denne rapporten forklarer hvilke teknologier jeg har benyttet for å lage en programvare med intensjon om å gi grafikkoperatører i NEP Norway AS full kontroll over data fra scoringsystemer til bruk i TV-grafikk. Programmet som er utviklet bygger på TV-bransjens arbeidsflyt og problemløsing ved å gjenskape fysiske oppgaver virtuelt.

Resultatet er en applikasjon som er allsidig, og som enkelt kan videreutvikles etter behov. Sammenlignet med NEP Norway sin tidligere løsning på lavbudsjettsoppdrag, vil denne løsningen gi grafikken et profesjonelt løft, samtidig som det gir mulighet for mer sofistikert grafikk.

Abstract

In this report I am explaining which technologies I have used to make an application with the intention of giving the graphics operator the full control of populating television graphics with data from external score tracking systems. The application builds upon workflows already known to the television industry, by recreating a virtual representation of physical tasks.

The result is a versatile application which is easy to develop extentions to when ever needed. Comparing NEP Norway's earlier solution regarding low budget jobs, to this new system, the TV-graphics can now be more sophisticated.

Innhold

Figurer

Ordliste

Boilerplate

En boilerplate er en ferdig mappe-struktur som man tar utgangspunkt i for videre utvikling. Boilerplaten inneholder gjerne ferdige konfigurasjoner som korter ned tiden før man kan begynne.

CLI

CLI står for Command Line Interface. Et brukergrensesnitt i kommandolinjevinduet som lar det interaktere med tillhørende programvare.

Grafikkoperatør

En grafikkoperatør er personen som kontrollerer grafikksystemet. Operatøren får beskjed fra produsenten hvilke elementer som skal på og av luften.

I/O

I/O står for Input/Output, og er en betegnelse som beskriver kommunikasjon mellom to separate systemer. Eksempel kan være minnepen og en datamaskin.

Playout

Playout som i engelsk for utspilling. Beskriver en handling eller maskinvare som gjør digitale kommandoer om til grafikk.

Singlepage

En singleplagenettside er en side hvor nettsiden lastes en gang, og deretter dynamisk laster innhold etter behov. Hver gang man går til en ny underside laster man ikke inn nettstedet på nytt. Dette gjøres gjennom å kode hvilke deler av nettstedet som skal være synlig.

Snapshot

Snapshot er et utdrag av noe som endrer seg kontinuerlig. Utdraget kan sammenlignes med et stillbilde hentet fra en film. Det gir informasjon om et spesifikt tidspunkt langs en tidsakse.

Super

Super er et grafisk element som hovedsakelig plasseres i nedre del av skjermen. Kan inneholde informasjon som navn på person i bildet, neste program i sendingen, eller lignende. Noen kjenner dem bedre som Lower Thirds.

Yarn

Yarn er en CLI som gir brukere mulighet til å installere og avinstallere rammeverk, og automatisk registrerer disse som avhengigheter i prosjektets administrative filer..

1 Introduksjon og relevans

I denne seksjonen skal jeg først ta for meg grafikk i et historisk perspektiv, og se på hvordan dette har endret seg over tid. Deretter går jeg videre til informasjon om oppdragsgiveren NEP Norway. Videre går jeg inn på hva det innebærer å være produsent og grafikkoperatør, for å gi et innblikk i hvem brukerne av denne applikasjonen er. Deretter vil jeg informere om konsekvensene COVID-19 hadde for denne oppgaven. Videre skal jeg presentere problemstillingen og til slutt oppgavens avgrensning.

1.1 TV-grafikk før og nå

På begynnelsen av 80-tallet hadde BBC kun to ansatte som jobbet med grafikk (Archbold 2006). Dersom tekst eller illustrasjoner skulle vises på tv, var det håndtegnet (Archbold 2006). Etter hvert som teknologien utviklet seg, ble den også stadig billigere. Det førte til at selskaper kunne kjøpe maskinvare som kunne erstatte håndtegninger med printede plakater, og senere som digitale bilder direkte inn i videostrømmen. I dag er grafikksystemer ofte dyrt fordi det spesialutvikles fra gang til gang, og TV-grafikk blir derfor ikke gjennomførbart på mindre arrangementer med små budsjetter.

1.2 NEP og TV-grafikk

NEP er et verdensomspennende selskap med høy kompetanse på sport, musikk og andre arrangementer. De har over 30 års erfaring, og kontorer i 24 forskjellige land (*NEP Group - Behind Powerful Production* 2020). På verdensbasis har de over 190 OB-busser (produksjonsbusser) og 53 studioer (*NEP Group - Behind Powerful Production* 2020). Med sine 4000 faste ansatte, og en stor base med frilansere, har de bidratt blant annet under innspilling av Solo: A Star Wars Story, med videoeffekter til Game of Thrones sesong 7, samt gjennomført produksjoner som The National Football League (NFL), Eurovision Song Contest og Eliteserien (*NEP Group - Behind Powerful Production* 2020).

NEP Norway er den norske avdelingen. De har kontorer i Oslo og Trondheim. Selskapet hadde en omsetning på kroner 401 millioner i 2019 (Proff.no 2020). De har gjennomført en rekke produksjoner som Hver gang vi møtes, Toppidrettsuka, Skal vi danse og Idol (*NEP Norway - Projects* 2020). Trondheimskontoret jobber mye med konferanser, festivaler og sportsproduksjoner. Sportsproduksjoner er en av NEPs sterkeste sider, og er et område som trekker et stort antall seere.

TV-seere som ser på fotballkamp eller skiskyting har en forventning om at informasjon de ikke kan se gjennom kamerabildet skal bli vist som grafikk. Starter TV-seeren en fotballkamp midt i sendingen, forventes det at stillingen og antall minutter spilt vises på skjermen. Både fordi vi er vant til det, men også fordi det er vanskelig å følge med dersom denne informasjonen ikke kommer frem.

Selv om NEP produserer en rekke store arrangementer, jobber de også med mindre produksjoner. Disse produksjonene har ikke alltid budsjett som tåler kostnader med grafikk. For at NEP skal kunne tilby grafikk til produksjoner med lavere budsjett, kreves det et fleksibelt program som gjør det enkelt og billig å oppdatere grafikk for bruk til strømming eller TV-sendinger.

1.2.1 Produsentens rolle

En produsent er personen som har det overordnede ansvaret for at en TV-produksjon blir slik som planlagt. Det er produsenten som tar valgene om hvilket kamera som skal benyttes, og hvilke historier som skal fortelles på sendingen. Produsenten har det siste ordet, og er personen de andre i produksjonen retter seg etter. Dette gjelder også grafikk. Det er produsenten som bestemmer hvilken grafikk som skal på luften, og når. I sportssendinger kan grafikk kan være alt fra navn på deltakere, tabeller over resultater, tider og poengsum.

1.2.2 Grafikkoperatorens rolle

Grafikkoperatøren styrer grafikken som skal komme på skjermen. Denne personen sitter med kunnskapen om grafikksystemet og hvordan det fungerer. Det er grafikkoperatørens rolle å gjøre klar grafikk som skal benyttes gjennom sendingen, sørge for at grafikksystemet fungerer slik det skal, og spille ut grafiske elementer for produsenten når produsenten ber om det. En utfordring grafikkoperatøren kan ha, er å respondere raskt nok på produsentens ønsker. Det er derfor viktig med enkle løsninger, som krever få klikk slik at tiden mellom ønske og handling er kort.

1.3 COVID-19 og utslaget det ga for oppgaven

COVID-19 ble første gang påvist i Norge 26. februar 2020 (Folkehelseinstituttet 2020). For å forhindre smitte satte regjeringen restriksjoner for hvor mange som kunne samles på et sted. Dette fikk følger for NEP sine oppdrag som enten ble utsatt på ubestemt tid eller direkte avlyst. Blant de utsatte oppdragene var Nor-

gesmesterskapet i 10m luftpistol. Ansatte ved bedriften ble også permittert på kort varsel når regjeringen åpnet for dette. Der i blant min oppdragsgiver.

1.4 Problemstilling

I dag eksisterer det en rekke grafikksystemer. Disse systemene koster ofte mye å anskaffe, vedlikeholde, og benytte. Store selskaper som CNN, BBC (*About Vizrt* 2020) og NRK (Hofseth 2018), benytter blant annet Vizrt for playout av grafikk. NEP bruker selv Vizrt på større produksjoner, men på mindre produksjoner blir dette for kostbart. Løsningen er derfor å bruke Singular.Live. Singular.Live er et skybasert playout-system som enkelt lar deg designe og kontrollere grafikk gjennom nettleseren.

Den avgjørende forskjellen mellom Vizrt og Singular.Live er hovedsakelig pris. Vizrt er et mer avansert system, og derav kommer også kostnaden. Singular.Live er gratis å bruke, så lenge deres vannmerke er aktivt. De lar deg bruke deres designverktøy samt teste grafikken uten kostnad. Kostnadene kommer dermed ikke før man skal bruke grafikken i produksjon uten vannmerket.

Problemet med Singular.Live, er at det kreves ytterligere håndtering av data før det kan presenteres som grafikk. Med bakgrunn i dette, blir min problemstilling for oppgaven som følger:

> Hvordan kan en grafikkoperatør dynamisk håndtere og tilpasse data fra eksterne leverandører til bruk i TV-grafikk?

Singular.Live tilbyr et REST API for kontrollering av deres systemer. Ved å benytte API-et kan ekstern programvare kommunisere og kontrollere Singular.Live. Gjennom et tilpasset grensesnitt som virker intuitivt for grafikkopratøren skal det være mulig å plassere riktig data på riktig plass i grafikken.

Grunnen til at vi ønsker å kunne håndtere data fra eksterne leverandører dynamisk er markedet. I mindre produksjoner kan endringer komme raskt, og disse endringene vil kunne ha effekt på hvilke data som skal presenteres i grafikken. I mindre produksjoner kan også leverandør av data variere fra gang til gang.

1.5 Avgrensning og disposisjon

Siden NEP Norway ønsket et grafikkprogram til NM i Pistolskyting på Ørlandet, ble det naturlig å avgrense oppgaven min mot dette. Overføringsverdien i produktet mitt vil likevel være stort, da systemet som nevnt skal være dynamisk. Jeg har hatt som mål å gjøre programmet enkelt å endre, slik at det kan brukes til flere sportsgrener, og andre typer arrangementer.

Videre i denne oppgaven vil jeg presentere relevant teori, deretter se på teknologi og metode, vise til resultater, diskutere resultatene og til slutt presentere en konklusjon på problemstillingen.

2 Teori

I dette kapittelet skal jeg ta for meg relevante begreper og bakgrunnskunnskaper som er viktig å kjenne til. Her har jeg valgt å ta for meg User Experience, for å sette fokus på viktigheten av brukervennlighet i utviklingsprosessen av en applikasjon. Herunder kommer begreper som skeuomorfisme og nodediagrammers viselle egenskaper. Videre tar jeg for meg Model-View-ViewModel designmønster som gir innsikt i oppbygging av et grafisk brukergrensesnitt som består av både visuelle og funksjonelle egenskaper. Deretter presenterer jeg prinsippet bak I/O-systemer som har overføringsverdi til denne oppgaven. Helt til slutt presenterer jeg utviklingsmetodene jeg har brukt i dette prosjektet.

2.1 User Experience (Brukeropplevelse)

„Kvalitet ved et produkt måles ved hvor lett de som skal bruke det, forstår, bruker og liker produktet. Vi kan dermed si at en brukeropplevelse ikke bare handler om teknologi, men også om hvilke handlinger vi benytter teknologien til" (Bakke 2017).

2.1.1 Skeuomorfisme

Ordet Skeuomorfisme ble først brukt av H. Colley March i 1889 for å beskrive strukturelle avledninger; 'The forms of ornament demonstrably due to structure require a name. If those taken from animals are called zoomorphs, and those from plants phyllomorphs, it will be convenient to call those derived from structure, skeuomorphs.' (Dictionary 2020).

Skeuomorfi er en metode for å gjøre overgang fra fysiske oppgaver til digitale oppgaver så enkel som mulig, ved å etterligne de fysiske egenskapene og bruksområder (Pratas 2014). Dette gjør det lettere for brukere å lære seg grensesnittet dersom de har lignende erfaringer fra den fysiske verden (Pratas 2014).

2.1.2 Nodediagrammers visuelle egenskaper

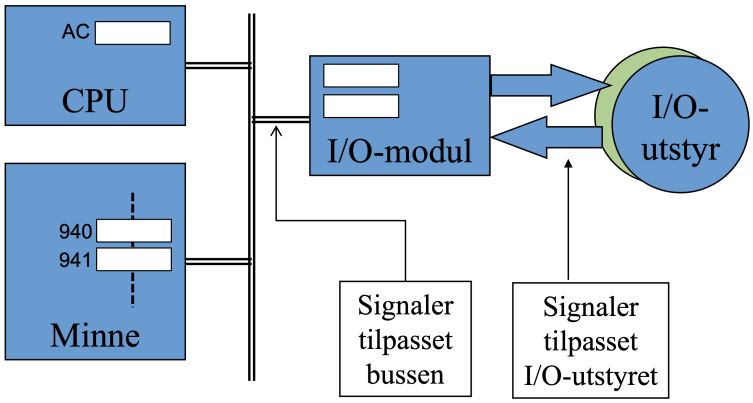
Å visualisere nodediagrammer har som mål å gi et godt overblikk over store mengder data (Wybrow mfl. 2014). Interaksjon med visualiserte nodenettverk kan redusere den kognitive innsatsen til brukeren ved å la dem lokalisere og fokusere på deler av diagrammet i gangen (Wybrow mfl. 2014).

2.2 Model-View-ViewModel designmønster

MVVM er en arkitektur og designmetode for å splitte data og brukergrensesnitt (Omari, Erramdani og Rhouati 2020). Model representerer data, brukergrensesnittet representerer View. View blir skapt gjennom å sette sammen data fra Model med en forhåndsgitt mal. Tanken bak MVVM er at informasjon flyter begge veier. Model sørger for at View har riktig data, mens View sørger for at Model får riktige endringer. View og Model har ikke direkte kontakt med hverandre (Omari, Erramdani og Rhouati 2020). ViewModel håndterer kommunikasjon mellom View og Model (Omari, Erramdani og Rhouati 2020). Slik unngår man uendelige løkker hvor View oppdatere Model, som oppdatere View og så videre. View inneholder ingen logikk. Dette er det Model som tar seg av. View sørger kun for det grafiske.

2.3 Prinsippet bak I/O-systemer

En datamaskin kan kobles til mange ulike enheter. Disse enhetene kalles I/O-enheter, og står for Input/Output-enheter. For at en datamaskin skal kunne kommunisere med et tastatur, en minnepenn eller en harddisk, trengs det kontrollere (Maribu udatert). Disse kontrollerne kommuniserer med I/O-enheter ved å benytte drivere som er tilpasset I/O-enheten.



Figur 1: Forenklet illustrasjon av I/O-modulens plassering mellom operativsystemet og I/O-enheter. Illustrasjonen er hentet fra forelesningen om IO i faget TDAT1003 datateknikk og operativsystem og er laget av Geir Ove Rosvold og Donn Morrison

I figur 1 er det I/O-modulen som tilpasser og tolker signalene som sendes til og kommer fra I/O-utstyret. I/O-modulen står også for å tilpasse og tolke signaler som sendes til og kommer fra operativsystemet. Prinsippet bak I/O-systemer er

dermed å flate ut forskjeller mellom maskinvare og operativsystem. Det gjør det mulig å lage utallige typer I/O-enheter. For at I/O-enheten skal fungere mot en datamaskin trengs det derfor kun en driver.

2.4 Utviklingsmetoder

Det eksisterer en rekke utviklingsmetoder. Hvilken utviklingsmetode som passer for hvert enkelt prosjektet varierer etter prosjektets behov. Nedenfor vil jeg presentere to utviklingsmetoder som har hatt relevans for mitt prosjekt.

2.4.1 CBFSD

Chaos-model er en utviklingsmetodikk som er utviklet av L. B. S. Raccoon (Wu mfl. 2005). Metodikken fokuserer på å lage programvare, fremfor prosjektstyring (Wu mfl. 2005). Chaos-model bygger på at man skal løse det viktigste problemet først (Wu mfl. 2005). CBFSD, står for 'Chaos-model based framework for embedded software development' og er en utvidet metodikk med Chaos-model som grunnlag (Wu mfl. 2005). CBFSD deler prosessen opp i tre parallelle undernivåer: Task Level, Code Level og Component Level (Wu mfl. 2005).

Task Level innebærer å håndtere programvaren som helhet, eller som frittsåtende deler av en helhetlig programvare (Wu mfl. 2005). Dette nivået tar altså for seg av organisering og tidsfrister (Wu mfl. 2005). Task Level fungerer som et administrativt nivå.

Code Level er nivået hvor programutviklere bestemmer hvilke rammeverk, språk og arkitektur, som realiserer programmets funksjoner (Wu mfl. 2005).

Component Level er nivået som skal ta seg av kompleksiteten av programvaren (Wu mfl. 2005). Jo høyere kompleksitet programvaren har, jo flere undernivå inngår i komponentnivået (Wu mfl. 2005). Dette nivået er veldig ustabilt, siden endinger i Code Level eller Task Level ofte vil føre til endringer også her (Wu mfl. 2005).

2.4.2 Vannfall

Vannfallsmetoden er en velkjent og tradisjonell utviklingsmetode. Metoden består av seks tydelige steg som beskriver hvordan et prosjekt skal gjennomføres (Liu, Xia og Zhang 2014). Det fullførers et og et steg før man går videre til neste.

Planlegging er første steg og her planlegges løpet til prosjektet. Det settes frister for diverse prosjektfaser, underleveranser og så videre (Liu, Xia og Zhang 2014).

Kravanalyse er andre steg. Her settes kravene til sluttproduktet. Utvikling av use-caser og lignende hører hjemme her. Dette steget blir ofte slått sammen med planleggingssteget (Liu, Xia og Zhang 2014).

Design er tredje steg. Her designes prototyper som wireframes på brukergrensesnittet (Liu, Xia og Zhang 2014).

Programmering er fjerde steg. Her starter man utviklingen av produktet. Det velges teknologier og rammeverk som skal benyttes for å gjennomføre oppgavene (Liu, Xia og Zhang 2014).

Testing er femte steg. For at produktet skal fungere for brukerne testes det på en gruppe mennesker (Liu, Xia og Zhang 2014). Her blir tilbakemeldinger og brukerønsker videreført i et eventuelt nytt programmeringssteg (Liu, Xia og Zhang 2014).

Drift og vedlikehold er sjette og siste steg. Når produktet er ferdig kan det slippes ut til markedet, eller til brukergruppen produktet er beregnet på. Det vil som regel alltid oppstå problemer eller feil som må rettes. Dette utføres i denne fasen (Liu, Xia og Zhang 2014).

3 Teknologi og metode

I denne seksjonen vil jeg presentere alt som trengs for å reprodusere arbeidet som er gjort i denne oppgaven. Jeg vil først beskrive arbeidsmetoden jeg har brukt, deretter vil jeg redegjøre for valg av rammeverk og til slutt vise til kontakten jeg har hatt med eksterne aktører for å kunne beholde framdriften i prosjektet.

3.1 Arbeidsmetode

I begynnelsen av prosjektet var tanken å benytte vannfallsmetoden som utviklingsmetode. Denne metoden ble byttet ut med CBFSD (se 2.4.1). Grunnen til at vannfallmetoden ble byttet ut med CBFSD er hovedsakelig på grunn av fleksibiliteten. En annen grunn for byttet er at mye av planleggingen allerede ble gjort i høstprosjektet. Høstprosjektet la blant annet føringer for hvordan utseende skulle være. CBFSD er utviklet for å håndtere endringer, mens vannfall er mindre fleksibelt. I et prosjekt hvor hovedformålet er å lage noe som skal være dynamisk og fleksibelt, er det vanskelig å vite alle krav på forhånd. Dette er en forutsetning for at vannfall skal fungere optimalt. Samtidig var det en rekke nye teknologier og rammeverk å sette seg inn i som ga usikkerhet i hvordan enkelte krav skulle løses.

CBFSD-metoden beskrev bedre hvordan jeg ønsket å håndtere prosjektet. Task Level fikk en utvidet beskrivelse som det administrative rundt hele prosjektet. Jeg måtte holde kontakt med eksterne selskaper for å bestille grafiske elementer og spesialtilpasset programvare, samtidig som jeg måtte følge fremdriftsplan og drive med dokumenthåndtering. Som nevnt i kapittel 2.4.1 er chaos-model beregnet for å løse det viktigste problemet først. Som den eneste studenten på prosjektet, falt det naturlig å gjennomføre det nettopp slik. De viktigste problemene er ofte de problemene som må løses for å kunne gå videre. Code level, som beskrevet i kapittel 2.4.1, omhandler hvilke rammeverk som skal benyttes. De benyttede rammeverkene som blir omtalt i kapittel 3.2, blir tatt i bruk etter hvert som behovet har oppstått i utviklingsprosessen. Component level, som er det ustabile nivået, var derfor stadig under endring. For hvert nye rammeverk ble det justeringer på kodens komponenter. Som nevnt tidligere har denne utviklingsmetoden større fokus på selve utviklingen, og mindre på prosjekthåndteringen. Likevel fungerte dette utmerket da mye av grunnlaget for planleggingen allerede var gjort i høstprosjektet, samtidig som NEP var tilgjengelig ved spørsmål.

Jeg hadde nær kontakt med NEP gjennom prosessen, ettersom de stilte en egen kontorpult til disposisjon for meg i deres lokaler. Selv om de ikke har aktive utviklere i Trondheim, har oppdragsgiver lettere erfaring med utvikling og høy teknisk

innsikt. De var derfor tilgjengelige for bransjerelaterte spørsmål, og for sparring på problemer som oppsto. Siden jeg hadde nær kontakt med NEP, vurderte jeg det som ikke nødvendig med møter. Problemer og spørsmål som har oppstått ble tatt fortløpende. Etter 12. mars, var koronaviruset også en medvirkende årsak til at møtene ble vanskelig å gjennomføre. Mer om dette i kapittel 1.3.

Timelister ble ført i et program som heter TrackingTime. Med dette programmet ble timer ført på riktig underkategori for prosjektet, som programmering og dokumentasjonsarbeid for å holde kontroll på timebruken på de forskjellige punktene. Hver dag ble det også loggført hva som ble gjort og hvilke problemer som hadde oppstått under økten.

3.2 Valg av rammeverk

Valg av teknologier er delvis basert på ønsker fra NEP selv, mens andre er basert på vurderinger gjort etter å ha undersøkt fordeler og ulemper ved diverse rammeverk.

3.2.1 ReteJS

ReteJS er et modulært rammeverk for å visualisere programmering. Rammeverket har modulær struktur og støtte for selvutviklede komponenter og plug-ins. ReteJS er bygget for å bli implementert i andre applikasjoner.

Oppdragsgiver kom opprinnelig med et ønske om å kunne visualisere dataflyten med Node-RED. Node-RED er en programvare for å knytte sammen maskinvare og API-er i et event-drevet miljø, gjennom å knytte noder sammen i en web editor. Som nevnt tidligere, har nodediagram en visuell egenskap (Wybrow mfl. 2014). Gjennom grundig undersøkelse og en liten periode med testing ble Node-RED byttet ut med ReteJS.

Grunnen til at ReteJS ble valgt over Node-RED var hovedsakelig arbeidsmengden med å implementere rammeverkene. Node-RED var et selvstendig program, som skulle kjøres som en frittstående applikasjon. Dersom ønske om en alt-i-en løsning skulle innfris, ble arbeidsmengden urealistisk for å kunne oppfylle kravene til NEP:GraphicsRouter som skulle testes i Norgesmesterskapet i 10m luftpistol i mars 2020. Tid var derfor hovedfaktoren for at ReteJS ble valgt over Node-RED. Hadde ikke tid vært en faktor, ville Node-RED blitt valgt over ReteJS av den grunn at utviklermiljøet rundt Node-RED er større. Det ville vært lettere å få hjelp ved feilsøking.

En av bakdelene med ReteJS er mangelfull dokumentasjon. Enkelte forklaringer og eksempler stemmer ikke overens med kodens faktiske funksjoner. Dette tyder på at rammeverket har blir oppdatert uten at dokumentasjon har blitt prioritert. Deler av den engelske dokumentasjon er heller ikke oversatt fra originalspråket russisk. Dette gjorde det vanskelig å feilsøke på mindre grunnleggende funksjoner, og krevde derfor mye tid.

3.2.2 ElectronJS

ElectronJS er et rammeverk for å lage plattformuavhenging programvare. Rammeverket kombinerer Chromium som nettleser og Node Runtime for å skape applikasjoner basert på HTML, CSS og JavaScript. Kjente applikasjoner som Slack, Discord og Visual Studio Code er bygget med ElectronJS (*ElectronJS.org* udatert).

ElectronJS er ikke det eneste rammeverket for å lage plattformuavhenginige skrivebordsapplikasjoner. NWJS og EnyoJS har tilsvarende egenskaper. Siden jeg hadde valgt ElectronJS som rammeverk allerede til høstprosjektet, var det naturlig å videreføre dette i bacheloroppgaven.

I tillegg har ElectronJS et stort utviklingsmiljø. ElectronJS har 82354 stjerner, 940 bidragsytere og omlag 2800 følgere på GitHub (*ElectronJS GitHub Page* udatert). NWJS og EnyoJS har henholdsvis 36745 og 1952 stjerner, 102 og 80 bidragsytere, og 1700 og 223 følgere (*NWJS GitHub Page* udatert;*EnyoJS GitHub Page* udatert). Det store utviklingsmiljøet til ElectronJS, gir bedre forutsetninger for å finne svar på problemer som kunne oppstå underveis. Det gir også bedre forutsetninger for at rammeverket i større grad blir vedlikeholdt. Dette var hovedgrunnen til at ElectronJS ble videreført som rammeverk til applikasjonen.

3.2.3 VueJS

VueJS er et JavaScript-rammeverk for utvikling av singlepage sider (*Getting Started - vue.js* udatert). Rammeverket bygger på arkitekturmønsteret MVVM (se kapittel 2.2). VueJS erstatter høstprosjektets oppbygging med Create-React-App, fra nå omtalt som CRA. CRA krever mye manuell konfigurering dersom man utfører CRA sin kommando `eject` gjennom yarn. Yarn er en pakkehåndteringsprogramvare som også fungerer som prosjektadministrasjonsprogramvare. Jeg så meg nødt til å bruke denne kommandoen i høstprosjektet, som i ettertid bare laget problemer med kompliserte konfigurasjonsfiler. VueJS erstattet derfor ReactJS fra høstprosjektet

på grunn av VueJS sin CLI. VueJS sin CLI lar deg konfigurere din egen boilerplate ved å huke av for hva du trenger. Følgende rammeverk ble lagt til boilerplate-en:

- Babel
- TypeScript
- Router (Ruter)
- Vuex (Lagring)

Babel er en JavaScript kompilator. Denne kompilatoren lar deg skrive i ECMAScript 2015+ og likevel få bakoverkompatibel JavaScriptkode (*What is Babel? · Babel* udatert). Dette gjør det mulig å bruke moderne JavaScript og samtidig kunne bruke det i eldre nettlesere (*What is Babel? · Babel* udatert).

TypeScript er JavaScript med integrert typesjekking. Ettersom JavaScript ikke har typesjekking som standard, vil ikke utviklingsverktøy som Visual Studio Code kunne gi tilbakemelding ved bruk av feil type variabler. Dermed blir det vanskelig å unngå feil under kjøring. TypeScript setter fart på utviklingen, fanger feil og tilbyr løsninger før du kjører koden (*TypeScript: Typed JavaScript at Any Scale.* Udatert).

Router gir mulighet til å bygge en singlepage nettside. Vue Router er den offisielle rutermodulen til VueJS (*Introduction — Vue Router* udatert).

Vuex fungerer som et globalt lagringsobjekt. Vuex automatiserer prosessen med oppdatering av variabler slik at alle data som blir hentet fra objektet er riktig i sanntid.

Deretter ble standardoppsettet benyttet med unntak i modus for ruteren. "History mode" ble byttet ut med "hash mode", ettersom history mode krever en webserver som apache, nginx eller lignende. Deretter ble kommandoen
`vue add electron-builder` kjørt for å legge til ElectronJS i prosjektet. Electron-builder er en plug-in til VueJS CLI skrevet av Noah Klayman (*A Vue Cli 3/4 plugin for Electron with no required configuration* udatert). Denne plug-in-en bygger om prosjektstrukturen til å være tilpasset ElectronJS samt legger til kommandoer i prosjektet for å starte utviklingsmiljøet og pakke det sammen til én kjørbar fil.

3.2.4 Singular.Live

Valget av Singular.Live er basert på et ønske fra NEP. Oppdragsgiver hadde hørt at selskapet Singular.Live og deres playout-løsning hadde vunnet priser for funksjonalitet og nyvinning. Tanken fra start var å utvikle et eget playout-system. Oppdragsgiver mente at denne oppgaven burde utkontrakteres til andre aktører for å holde vedlikeholdsmengden for NEP nede. Singular.Live ble derfor et naturlig alternativ for bruk i utviklingsprosessen, samt for brukt i produksjoner på grunn av deres fleksibilitet. Singular.Live har en egen avdeling for integrering. Denne avdelingen har også vært til stor hjelp gjennom utviklingen av NEP:GraphicsRouter.

En klar ulempe ved å velge Singular.Live, er at det er et ungt selskap. Produktet er fremdeles under store endringer, og dokumentasjon kan til tider virke mangelfull. Selv om deres designverktøy er intuitivt og enkelt, kan det også ved mer komplekse grafiske uttrykk være mangelfullt. Dette er fordi designverktøyet enda er under utvikling.

Fordelene med Singular.Live er mange, under presenterer jeg de viktigste for mitt prosjekt.

Composition Script er Singular.Live sin egen kodeeditorløsning for å integrere JavaScript i grafikken. Ved å benytte seg av composition script kan man utvikle sofistikert grafikk som forandrer seg basert på hva den skal vise.

Bodymovin er et rammeverk utviklet av Airbnb som lar deg animere vektorgrafikk i det populære programmet Adobe Aftereffects (*Lottie for Web GitHub Page* udatert). Den eksporterte filen fra Adobe Aftereffects er en JSON-fil som inneholder alle vektorpunkter, samt animasjonsdata. Denne filen lastes inn i grafikken og blir lest av Lottie-Web (*Lottie for Web GitHub Page* udatert). Elementer og animasjoner blir da lik som i Adobe Aftereffects. Dette sparer tid og penger, fordi det eliminerer ut flere steg i prosessen som før har vært nødvendig for å få de grafiske elementene og animasjonene til å samsvare i TV-grafikken.

3.2.5 Companion

Programmet Companion er et Open-Source prosjekt utviklet av Bitfocus og har fått godt fotfeste i eventbransjen. Programmet har en brukergruppe på mer enn 7000 unike brukere (*Companion User Group* udatert), og er en plug-in basert programvare for å kontrollere over 200 andre programvarer eller enheter. Det gir

brukeren mulighet til å kontrollere videomiksere, projektorer og en hel rekke andre kategorier. Ettersom dette er et mye brukt program var det et ønske om å kunne benytte dette til å kontrollere grafikken. Companion hadde ingen støtte for Singular.Live, og dette måtte jeg dermed utvikle.

3.3 Kontakt med eksterne aktører

NEP er en av mange aktører som er involvert i arrangementer. For at Norgesmesterskapet i 10m luftpistol på Ørlandet skulle ha grafikk som ønsket var jeg nødt til å holde kontakt med flere aktører.

3.3.1 Megalink AS

I dette tilfellet var Megalink leverandør av eksterne data. Megalink leverer digitale blinker for bruk i sportsskyting. De har en digital plattform for live resultatvisning. Denne løsningen var ikke tilfredstillende nok til bruk i sendingen, fordi den ikke var TV-vennlig. De var imøtekommende og utviklet en egen modul i sine systemer som skulle sende data over nettverk til NEP:GraphicsRouter. Dataen ble sendt som JSON som svar på en HTTP GET forespørsel. De ga meg også tilgang til demodata til deres systemer, slik at jeg kunne emulere et live arrangement.

3.3.2 Produsenten og de grafiske elementene

Produsenten til Norgesmesterskapet i 10m luftpistol på Ørlandet, Kim Jørgen Vang, er selvstendig næringsdrivende. Han har produsert blant annet fotballkamper i Eliteserien, og landskytterstevner. Jeg hadde et møte med Kim for å planlegge hvilke grafiske elementer han mente var nødvendige. Etter sammenstilling av ønskede grafiske elementer, ble dette videreformidlet til et kommunikasjonsbyrå. Dette gjorde NEP for å redusere arbeidsmengden min, da disse grafiske elementene var nødvendige for å teste NEP:GraphicsRouter. Slik fikk jeg bedre tid til å fokusere på det tekniske.

4 Resultater

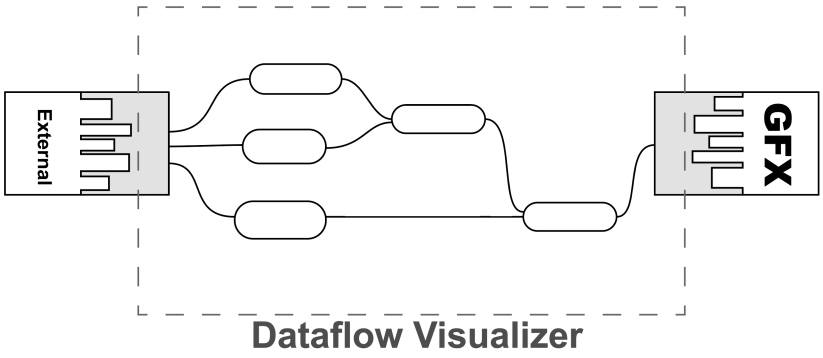
I denne seksjonen vil jeg presentere de vitenskapelige, ingeniørfaglige og administrative resultatene mine. Jeg vil først beskrive hvordan jeg har brukt teorier og prinsipper fra studiet for å løse problemstillingen i de vitenskapelige resultatene. Så vil jeg presentere hvordan de ikke-funksjonelle kravene fra visjonsdokumentet ble ivaretatt i de ingeniørfaglige resultatene. Til slutt vil jeg beskrive de administrative resultatene rundt hvordan jeg har håndtert prosjektet.

4.1 Vitenskapelige resultater

Dynamisk håndtering og tilpassing av data fra eksterne leverandører kan være en vanskelig oppgave for applikasjoner som krever et fast format eller oppsett. En løsning på dette problemet skal jeg nå presentere i de neste underkapittelene.

4.1.1 I/O prinsippet

Første punkt i problemstillingen er å kunne håndtere data fra eksterne leverandører og videreformidle den til Singular.Live. Produktets sammensetning bygger på samme tankegang som enhetsdrivere for operativsystemer. Drivere flater ut forskjeller mellom I/O-enheter og operativsystemet. På samme måte flater Megalink-modulene ut forskjellen på Megalink sine systemer mot ruteren, og Singular.Live-modulene flater ut forskjellene på ruteren til deres API som illustrert under.



Figur 2: Illustrasjon av utflating mellom eksterne til ruteren og fra ruteren til grafikken (GFX).

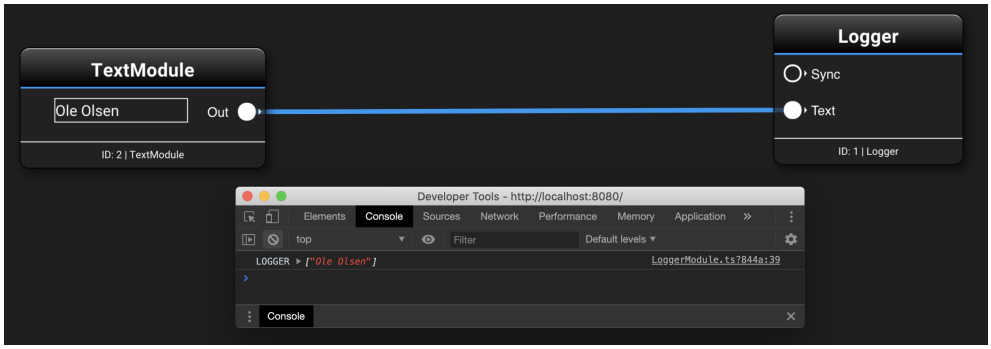
Resultatet av et slikt system er at dersom NEP skifter leverandør av data, kan de programmere nye moduler som representerer driverne. På lik linje som at det

er mulig å skifte grafikkmotor fra Singular.Live til noe annet, kan man lage en ny driver for å kommunisere med den nye grafikkmotoren.

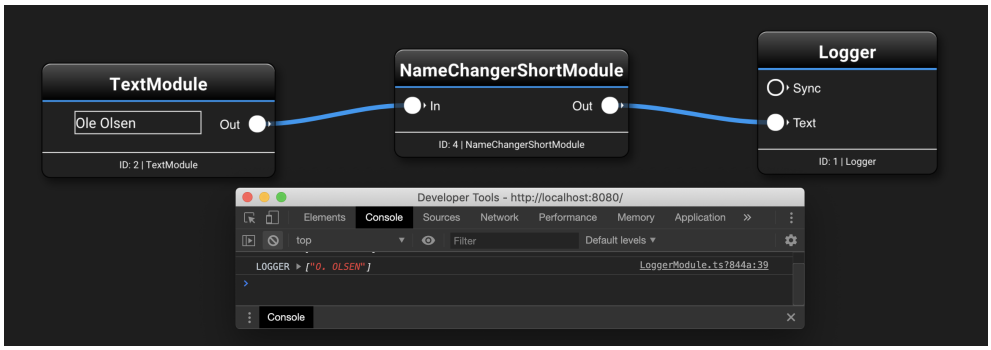
4.1.2 Modulær oppbygging (omformere)

Er det én ting TV-bransjen har kjennskap til, så er det omformere. Videosignalet HDMI må ofte konverteres til et annet videosignal som SDI, eller at HD-videooppløsning må konverteres til et format for å passe en vegg av LED-skjermer. Den modulære strukturen i ruteren hvor man kan trekke koblinger mellom forskjellige omformere, løser den ene delen av problemstillingen – tilpassing av data. For å kunne tilpasse data som skal til de forskjellige delene av grafikken må man kunne gjøre endringer. Navn skal forkortes, variabler skal tolkes og tabeller skal splittes.

Under er det illustrasjoner som tar for seg forkorting av navn. Figur 3 viser resultatet uten forkorting av navn og gir oss `Ole Olsen` som opprinnelig i TextModulen. Figur 4 viser resultatet med forkorting, altså `O. Olsen` som ble forandret av NameChangerShortModulen.



Figur 3: Utdrag fra applikasjonen som viser resultatet av en tekstmodul som er koblet direkte på loggermodulen.



Figur 4: Utdrag fra applikasjonen som viser resultatet av en namechangershort-modul som sitter mellom tekstmodulen og loggermodulen.

Ved å bruke denne teknikken kan det enkelt utvikles nye omformere etter behov. Dette gir mulighet til å tilpasse produktet til enhver ekstern leverandør, enhver tilpassing av data og til enhver grafikkmotor.

4.1.3 Visualisering av dataflyt

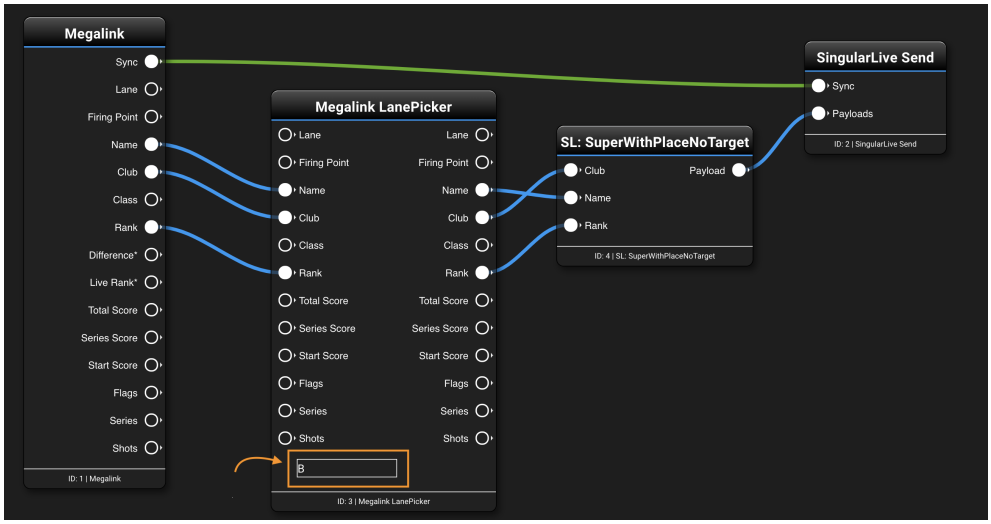
For å korte ned opplæringstiden til applikasjonen har jeg implementert et visualiseringsverktøy. Visualiseringsverktøyet bevarer likheter mellom den eksisterende arbeidsmåten i TV-bransjen, og implementerer det i applikasjonen. Det trekkes mye kabler fra A til B i TV-bransjen. I applikasjonen kan grafikkoperatøren nå trekke virtuelle kabler mellom moduler som tilsvarer omformere. Samtidig som dataflyten blir visualisert, blir også prinsippet om skeuomorfisme beholdt.

4.1.4 Hva med nøkkelordet dynamisk?

Dynamisk håndtering og tilpassing handler ikke bare om å kunne gjøre endringer i forkant. For å håndtere og tilpasse data i en live-setting, må systemet takle endringer i oppsettet. Visualiseringen gir grafikkoperatøren mulighet til å endre dataflyten i sanntid. Trekker man en ny kobling vil altså dataene flyte den veien.

Et annet eksempel på hvorfor det er viktig at applikasjonen er dynamisk, er muligheten til å velge hvilken skytebane det skal hentes informasjon fra. I 10m luftpistol er det åtte baner markert med A til H. Figur 5 viser hvordan en tilpasset modul for Megalink gir grafikkoperatøren mulighet til å velge hvilken bane og utøver som er aktuell å vise data fra i et grafisk element kalt en super. Dette tilfellet ville vist informasjon om utøver i bane B. Ved å endre bokstav mellom A og H i tekstfeltet

til modul Megalink LanePicker kan grafikkoperatøren selv velge utøver. Endringer skjer i sanntid og vil endre det grafiske elementet til å vise den valgte informasjon.



Figur 5: Utdrag fra applikasjonen som viser hvordan man dynamisk velger utøverdata. Her velges utøver i bane B

4.1.5 Companion som kontrollpanel

Et krav fra NEP, var at programvaren Companion skulle benyttes til å kontrollere grafikken. Løsningen ble å utvikle en modul som kommuniserte direkte med Singular.Live. Å animere inn og ut grafikk kan nå gjøres ved å trykke på programmerbare LCD knapper på en Elgato Stream Deck (se figur 6). Modulen er nå integrert som en del av den offisielle Companion-applikasjonen.

Figur 6: Bilde av Elgato Stream Deck XL. Bilde er hentet fra elgato.com

4.2 Ingeniørfaglige resultater

Jeg vil ut fra målene jeg satte meg i visjonsdokumentet, her redegjøre for statusen på disse.

4.2.1 Uavhengig av operativsystem

Som mål under utviklingen skulle produktet være plattformuavhengig. Hensikten med plattformuavhenginghet er å kunne benytte programmet på hele datamaskinflåten som NEP har internt i selskapet. Applikasjonen ble bygget og testet på MacOS. Mer om Windows i kapittel 5.1.1. Med ElectronJS som rammeverk for applikasjonen er muligheten for støtte også med Windows som operativsystem tilrettelagt, men ikke utført.

4.2.2 Lukket kildekode

Et offentlig GitHub-depot kunne ikke brukes, siden NEP hadde et krav om at produktet bare skulle tilhøre dem. Det ble derfor opprettet et privat depot, som har blitt brukt til å oppbevare koden som sikkerhet mot egen maskinvarefeil. Koden er derfor ikke tilgjengelig for offentligheten.

4.2.3 Intuitivt brukergrensesnitt

Selv om koronasituasjonen ikke har latt meg fullføre brukertester på de ansatte, er brukergrensesnittet blitt vurdert tidligere i prosjektet. Da jeg hadde kontorpult i NEP sine lokaler, var de ansatte med jevne mellomrom involvert i utformingen av brukergrensesnittet. En frilanser for NEP kikket på brukergrensesnittet da han var innom kontoret, og fikk med engang assosiasjoner til OB-bussene (produksjonsbussene til NEP) og deres lignende system. Han tipset meg også om at synkroniseringen mellom start- og endenodene burde hete `sync`, som det heter i TV-bransjen.

4.3 Administrative resultater

Det første visjonsdokument omhandlet et system som skulle ta for seg langrenn. Dette ble byttet ut med 10m luftpistol. For NEP var et Norgesmesterskap en mer attraktivt test for løsningen. Dette førte til at visjonsdokumentet allerede i tidlig fase ble endret. NEP:GraphicsRouter skal likevel være en applikasjon som håndterer endringer. Med utgangspunkt i et modulært system skulle denne endringen ikke være et hinder. Det som skulle vise seg å være et hinder var tidsfristen. Langrennsarrangementet skulle finne sted i august 2020, som er flere måneder

etter denne bacheloren skulle leveres. Det ville ha gitt noen måneder med ekstra tid dersom man skulle finne problemer med produktet. Norgesmesterskapet i 10m luftpistol skulle arrangeres allerede i slutten av mars 2020, som var en drastisk framskynding av første prototype sammenlignet med langrennsarangementet. Målet med å ha en fungerende prototype til 10m luftpistol krevde derfor en del raske løsninger som er omtalt i kapittel 3.

4.3.1 Brukertesting

Regjeringen satte smittevernsrestriksjoner som ga konsekvenser for brukertestingen på ansatte (se kapittel 1.3). I tillegg var alle ansatte permittert, og kunne derfor ikke møte meg til testing. I følge framdriftsplanen skulle brukertestingen foregå i uke 15. Dette kunne ikke gjennomføres, slik situasjonen utartet seg.

4.3.2 Kontakten med Megalink AS

Megalink AS hadde som nevnt tidligere ikke en løsning for å kunne levere informasjon til eksterne systemer. De utviklet derfor et endepunkt for å gi fra seg ønsket data. Jeg ga dem en beskrivelse av hva jeg trengte, og etter noen dager mottok jeg et dokument med informasjon om endepunktet. Jeg kunne dermed starte utviklingen fra min side før endepunktet deres var ferdig. Jeg fikk derfor laget en kjapp skisse til Megalink-modulen før planlagt tid i fremdriftsplanen. Selve løsningen til Megalink ble ikke tilgjengelig for meg før etter den planlagte perioden, samtidig som den inneholdte feil som jeg ikke kunne jobbe meg rundt. Megalink-modulen ble derfor forsinket til uken etter planlagt ferdigstilling. Dette skapte heldigvis ikke noen konsekvenser som først antatt. Løsningen var et snapshot av data som ble sendt inn i programmet gjennom å emulere et svar fra Megalink-modulen.

4.3.3 Kontakt med produsent

Møtet med produsenten Kim var ikke opprinnelig i planen da dette kom som en ekstra oppgave fra NEP. Møtet ga likevel den informasjonen som var nødvendig for å fortsette. Resultatet av møtet var at grafiske elementer som var ønsket av produsent ble bestilt hos et kommunikasjonsbyrå.

4.3.4 Kontakt med BERRE AS

BERRE AS var leverandør av grafiske elementer. Disse elementene skulle først leveres i slutten av uke 12, men ble utsatt da arrangementet ble utsatt på grunn av COVID-19. Elementene ble derfor levert uken etter. Det ble samtidig testet for muligheten å bruke Bodymovin i Singular.Live, noe som fungerte utmerket.

5 Diskusjon

I denne seksjonen vil jeg diskutere rundt resultatene presentert i forrige kapittel. Jeg tar først for meg kravene som ikke ble tilfredsstilt. Deretter tar jeg for meg krav som ble innfridd, og hva som er bra med disse. Videre vil jeg diskutere brukertester, hvorfor de ikke ble gjennomført, og hvordan de opprinnelig skulle gjennomføres. Deretter drøfter jeg applikasjonens innvirkning på samfunnet med perspektiver som økonomi og personvern. Til slutt vil jeg drøfte rundt min egeninnsats i denne oppgaven.

5.1 Krav som ikke ble tilfredsstilt

Mange av kravene i visjonsdokumentet ble oppfylt, men til tross for en god utviklingsprosess var det ikke mulig å oppfylle alle.

5.1.1 Støtte for Windows

Som nevnt i resultatkapittelet, var et ønske fra NEP at programvaren skulle fungere både på MacOS og Windows. I utviklingen har jeg lagt til rette for at programvaren også skulle kunne kjøre på Windows. Det som gjenstår for å fullføre kravet er å bygge den kjørbare exe-filen. Dette er en tidkrevende oppgave for en utvikler på MacOS som ikke har gjort det før, da konfigurasjonsprosessen er relativt innviklet. Det ble derfor bortprioritert, men det er likevel klargjort for å kunne benytte flere plattformer.

5.1.2 Egendefinert kontrollpanel

Kontrollpanelet ble ikke fullført i den grad at det er tilfredstillende for grafikkopretøren. Grafikkoperatøren må derfor benytte de redigerbare feltene i moduler i ruteren for å kontrollere flyten, som vist tidligere i figur 5. Dette er for tungvint i en live-setting og tilfredsstiller derfor ikke kravene satt i visjonsdokumentet.

Likeså er heller ikke eksportering og importering av kontrollpanelet fullført. Dette er funksjonalitet som naturligvis ville blitt implementert etter at kravet om et kontrollpanel var innfridd.

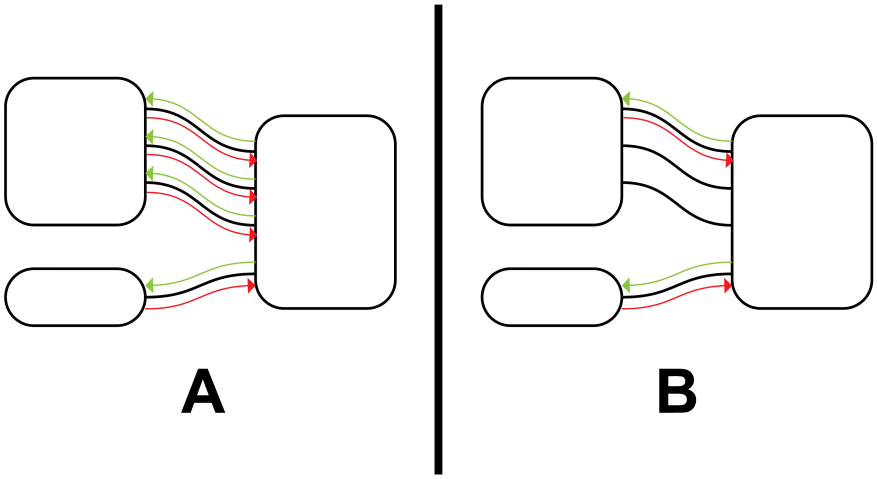
5.2 Krav som ble innfridd

Til tross for problemene jeg nevnte ovenfor, har applikasjonen også mye god funksjonalitet.

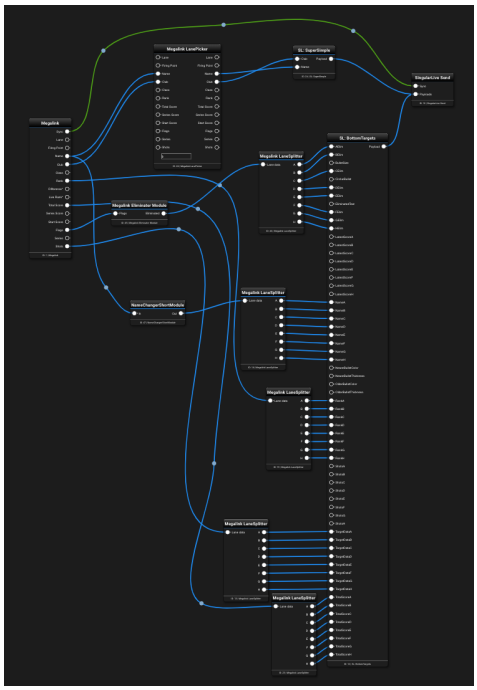
5.2.1 Store flytdiagrammer i ruteren

Fungerer store flytdiagrammer i ruteren? Det korte svaret er ja. Figur 8 inneholder 53 koblinger som binder sammen 12 moduler. Nodene er ansvarlig for å hente nødvendig data, og for å omforme dataen til et riktig formatert svar. Hver node må derfor kalkulere data ved hver gjennomkjøring av flytdiagrammet. Ruteren består av ReteJS og flere plug-ins som er tilpasset ReteJS. Blant plug-inene er Task plug-in. Task plug-in muliggjør injeksjoner av data inn i ruteren. Eksempel på dette er Megalink-modulen som kjører hver gang det blir hentet data fra Megalink. Denne plug-in var opprinnelig for mangelfull for at den skulle fungere som den var. Plug-in måtte derfor skrives om til å passe behovet i ruteren.

Effektivisering av Task plug-in var nødvendig. Det avanserte oppsettet vist i figur 8 brukte opprinnelig 212 nodekalkulasjoner for å fullføre en gjennomkjøring. Den brukte altså flyten beskrevet som A i figur 7. Dette førte til at antall kalkulasjoner ville økt betraktelig ved et større og enda mer avansert oppsett. Task plug-in ble derfor omskrevet til å benytte flyten som er beskrevet som B i figur 7. Dette reduserte antall kalkulasjoner fra 212 til 16. Dette er en betydelig effektivisering av Task plug-in.



Figur 7: Task plug-in illustrasjon. (A) henter data for hver tilkobling mellom hver node. (B) henter data én gang for hver tilkoblet node.



Figur 8: Utdrag fra applikasjonen som viser et avansert oppsett av dataflyt

5.3 Brukertester

Som nevnt i kapittel 1.3 satte regjeringen restriksjoner som fikk følger for NEP og denne oppgaven. Siden alle ansatte ble permittert og utilgjengelig, er det heller ikke gjennomført brukertester på de ansatte. Dette burde likevel bli gjort i fremtiden for å avdekke svakheter eller uklarheter.

5.3.1 Hvordan ville brukertesten blitt gjennomført?

Planen for brukertesten var å lage en oppgave til brukeren for så å ta et kvalitativt dybdeintervju. Brukeren ville her vært en av de ansatte i NEP, som vil ha ansvaret for grafikk på arrangementer i fremtiden og som har erfaring i dette fra før av. Før brukeren får oppgaven ville en kort introduksjon av programvaren bli gjennomført. Å vise brukeren hvordan han eller hun kobler seg til Singular.Live, og hvordan de legger til eksterne leverandører, samt en kort introduksjon til hvordan ruteren fungerer, uten å røpe hvordan oppgaven skulle løses.

Skjerm- og lydopptak av intervjuet ville blitt gjort ved samtykke. Det ville også blitt presisert at dersom brukeren står fast er det på grunn av et ikke-intuitivt

brukergrensesnitt, for å berolige en eventuelt nervøs testperson.

5.3.2 Brukertestoppgaven

Oppgaven ville så gå ut på å fylle en super i grafikken med utøverens navn, plassering og skytterklubb. Det brukeren må gjøre da, er å legge inn Megalink-modulen, modulen som representerer en super i grafikken og SingularLiveSend-modulen. Når brukeren kobler sammen de riktige koblingene vil brukeren kunne emulere data fra Megalink ved å benytte `Emulate` knappen på Inputsiden.

5.3.3 Intervjuet

Dersom brukeren ikke klarer oppgaven er det viktig å finne ut hva som er uklarhetene hos brukeren. Hva var det som ikke var logisk? Hva var vanskelig å forstå? Hvordan oppfatter brukeren koblingen mellom modulene? Uklarhetene ville blitt benyttet som et grunnlag for å lage spørsmål som er relevante i intervjuet.

Hvis brukeren har koblet riktig, vil grafikken fungere som planlagt. Da ville spørsmål om hvordan brukeren opplevde programmet være relevant for å kartlegge en fremtidig utvikling. Det ville også vært naturlig å legge inn spørsmål som hva brukeren mener mangler i programvaren.

Ved å gjennomgå skjermopptakene i kombinasjon med lydopptakene, vil dette kunne bidra til å utbedre uklarheter, forvirringer og feil i programvaren.

5.4 Innvirkning på samfunnet

Å vurdere utviklingsprosjekter med et helhetlig systemperspektiv gjør det lettere for utviklere å unngå uetiske og ikkeøkonomiske beslutninger.

5.4.1 Økonomi

I dette prosjektet er de økonomiske beslutningene tatt av NEP. Prosjektet vil likevel gi en økonomisk gevinst for NEP, fordi de nå kan gjennomføre arrangementer med grafikk, uten at det blir for dyrt, og vil dermed gi dem et økonomisk fortrinn i bransjen. Å kunne tilby profesjonell grafikk til lavbudssjettsprosjekter vil i fremtiden kunne generere flere kunder for NEP. De vil også kunne få større avanse på mellomstore og store prosjekter hvor utvikling av tilpasset programvare ville vært et tema.

5.4.2 Personvern

Å ha tilgang til data betyr å ha ansvar. Som utvikler i slike prosjekter kan informasjon som ikke skal være offentlig komme fram. Høstprosjektet ga meg innblikk i et slikt tilfelle. Når utviklingen mot en database fullt av utøverdata inneholder telefonnummer og adresse, er dette personlig informasjon som ikke skal på avveie.

Når utviklere senere skal integrere kommunikasjon med andre eksterne enn Megalink AS, er det derfor viktig å vite hvilken informasjon som skal holdes tilbake. Det er derfor ikke nødvendig å gi grafikkoperatøren tilgang til all informasjon som finnes i databasen, og som igjen vil redusere sjansen for at feil informasjon går ut på direkte-tv.

For meg har ikke dette hatt betydning for denne oppgaven. Dette er likevel et viktig tema som må tas hensyn til når applikasjonen skal utvides til å håndtere flere sportsgrener, eller andre eksterne leverandører.

5.5 Egeninnsats

Å vurdere egeninnsats objektivt kan være vanskelig. I dette prosjektet har jeg støtt på mye nytt og gått mange nye veier. Området prosjektet handler om blir sett på som en nisje i TV-bransjen, som igjen gir lite veiledning på best practice. Det igjen gjør at det blir vanskelig å vite hva som er lurt å gjøre, før man har prøvd. Prøving og feiling har vært essensielt i denne utviklingsprosessen. Det har krevd at jeg har hatt selvdisiplin og motivasjon for å fullføre. Selvsagt skulle jeg ønske at produktet ble helt ferdig, men slik ble det dessverre ikke. Jeg er uansett fornøyd med min egen innsats, og stolt over produktet jeg nå overleverer til NEP Norway AS.

Det er også positivt at Companion-modulen jeg skrev, har blitt godt mottatt av Singular.Live sin utviklingsavdeling. De var godt kjent med Companion fra før, men hadde ikke ressurser til å lage en slik modul selv. Samfunnet rundt Singular.Live hadde ønsket seg denne lenge, og de skulle publisere nyheten om modulen i et av nyhetsbrevene sine. De kom også med en rekke ønsker til utvidelser som vi sammen skal implementere de kommende månedene. Dette viser at til tross for at prosjektet handler om en nisje i TV-bransjen, er resultatene relevante for andre enn bare oppdragsgiver.

6 Konklusjon og videre arbeid

Gjennom denne rapporten har jeg forsøkt å knytte sammen prinsippet om I/O-systemer og User Experience Design for å gjør oppgaven til en grafikkoperatør så sømløs som mulig. Denne innfallsvinkelen er bare en av flere mulige vinkler å angripe problemstillingen på. Problemstillingen min er:

> Hvordan kan en grafikkoperatør dynamisk håndtere og tilpasse data fra eksterne leverandører til bruk i TV-grafikk?

For å svare på dette, har jeg flatet ut forskjeller mellom eksterne leverandører, applikasjonen og Singular.Live, ved hjelp av I/O-systemprinsippet. Samtidig har jeg brukt prinsippet om skeuomorfisme til å vedlikeholde grafikkoperatørens brukervennlighet.

NEP:GraphicsRouter er en programvare som i fremtiden vil bidra til at NEP leverer profesjonell grafikk på lavbudsjettsoppdrag. Ruteren lar grafikkoperatøren bestemme dataflyten og endring av data i kullisene til grafikken. Ved å bygge ruteren rundt prinsippet om skeuomorfisme fremhever produket gjennkjennelsesfaktoren for grafikkoperatøren. Sammenligningen med kabler og omformere gjør det enkelt å forstå hvordan ruteren skal brukes. Det er kun grafikkoperatørens egen evne til å løse logiske problemer om hvilke moduler som skal i hvilken rekkefølge, som bestemmer hvor godt grafikkoperatøren kan bruke applikasjonen.

Siden ruteren også bygger på prinsipper rundt I/O-enheter, gir det utviklere en enkel inngang i systemet. Å lage nye moduler som kommuniserer med andre eksterne leverandører av data blir relativt enkle og kostnadseffektive oppgaver. Det er også enkelt å lage nye moduler som kan tilpasse data.

6.1 Mine anbefalinger

Jeg vil anbefale personer som skal ta opp denne utviklingen å vurdere overgang til Node-Red som rammeverk. Dette rammeverket har en godt dokumentert oppbygging, et større samfunn, og aktive utviklere som holder rammeverket oppdatert, sammenlignet med ReteJS.

6.2 Videre arbeid

Programvaren har en del mangler som gjør den tungvint å bruke som den er nå. Utbedringene som er beskrevet under vil bidra til å øke brukervennligheten.

6.2.1 Svakheter som må utbedres

Programvaren lagrer ikke endringer når den avsluttes. Dette er en svakhet som må utbedres for at programvaren skal kunne brukes i det daglige. Det er veldig ineffektivt at grafikkoperatøren må lage samme oppsett hver gang programvaren avsluttes. Siste oppsett burde derfor lagres i applikasjonen til neste gang.

ReteJS har muligheter for å lage kategorier på koblingene. Eksempelvis kan strenger, tall og tabeller være kategorier. Det kan også lages underkategorier som tabeller med tall og tabeller med strenger. Ved å utnytte denne funksjonaliteten er det mulig å lage en solid ruter som ikke tillater at brukeren kobler tall inn i en tekstomformer.

6.2.2 Fallgruve av evige løkker i ReteJS

Evige løkker vil føre til at applikasjonen fryser, noe som er uheldig for en grafikkoperatør. Det er ikke tatt høyde for at ruteren skal oppdage evige løkker og avslutte disse selv. Likevel er ikke dette å anse som et betydelig problem ettersom man sjeldent looper signaler i TV-bransjen. Grafikkoperatører vil derfor neppe konfigurere evige løkker. Det er fremdeles helt klart en utbedring som burde gjøres.

6.2.3 Krav som må tilfredsstilles

For at programvaren skal bli behagelig å bruke for en grafikkoperatør, må et kontrollpanel være på plass. Uten dette er programvaren vanskelig å håndtere og gir ikke den korte responstiden som en grafikkoperatør burde ha på beskjeder fra produsent.

Eksportering og importering av oppsett fra tidligere gjennomkjøringer må implementeres. Det er viktig at oppsett kan lagres og oppbevares utenfor programmet av hensyn til redundans.

Brukertester på ansatte må gjennomføres for å forsikre at programvaren er intuitiv nok. Erfaringer fra brukertestene vil kunne generere nye krav.

Referanser

A Vue Cli 3/4 plugin for Electron with no required configuration (udatert). https://github.com/nklayman/vue-cli-plugin-electron-builder. (Accessed on 05/02/2020).

About Vizrt (2020). https://www.vizrt.com/vizrt. (Accessed on 04/29/2020).

Archbold, Steve (des. 2006). „TV GRAPHICS". English. I: *Broadcast Engineering* 48.12. Copyright - (Copyright 2006 by PRIMEDIA Business Magazines Media Inc. All rights reserved.; Last updated - 2017-10-31, s. 24. URL: https://search.proquest.com/docview/204169566.

Bakke, Sturla (2017). „Å planlegge for gode brukeropplevelser". I: *Metodebok for kreative fag*. Red. av Hans Erik Næss og Lene Pettersen. Universitetsforlaget, s. 220–221. ISBN: 978-82-15-027000-5.

Companion User Group (udatert). https://www.facebook.com/groups/companion/. (Accessed on 05/03/2020).

Dictionary, Oxford English (apr. 2020). *skeuomorph, n. : Oxford English Dictionary*. https://www.oed.com/view/Entry/180780?. (Accessed on 04/24/2020).

ElectronJS GitHub Page (udatert). https://github.com/electron/electron. (Accessed on 05/02/2020).

ElectronJS.org (udatert). https://www.electronjs.org/. (Accessed on 05/01/2020).

EnyoJS GitHub Page (udatert). https://github.com/enyojs/enyo. (Accessed on 05/02/2020).

Folkehelseinstituttet (mar. 2020). *Koronavirus – fakta og håndtering i Norge - helsenorge.no*. https://helsenorge.no/koronavirus/fakta-og-handtering-i-norge. (Accessed on 05/06/2020).

Getting Started - vue.js (udatert). https://012.vuejs.org/guide/. (Accessed on 04/30/2020).

Hofseth, Anders (mai 2018). *Bergensselskapet VizRT kritiseres hardt i epost sendt ut fra deres egen konto*. https://nrkbeta.no/2018/05/27/bergensselskapet-vizrt-kritiseres-hardt-i-epost-sendt-ut-fra-deres-egen-konto/. (Accessed on 04/29/2020).

Introduction — Vue Router (udatert). https://router.vuejs.org/. (Accessed on 05/03/2020).

Liu, Jin Hang, Hong Xia Xia og Heng Bo Zhang (2014). „A Research into the UML Legend in the Waterfall Model Development". eng. I: *Applied Mechanics and Materials* 519-520.Computer and Information Technology, s. 322–328. ISSN: 1660-9336.

Lottie for Web GitHub Page (udatert). https://github.com/airbnb/lottie-web. (Accessed on 05/05/2020).

Maribu, Geir (udatert). „I/O-administrasjon". I: *Kompendium i operativsystemer*. Utdraget ligger vedlagt, s. 1–3.

NEP Group - Behind Powerful Production (2020). https://www.nepgroup.com/. (Accessed on 04/29/2020).

NEP Norway - Projects (mar. 2020). http://nepnorway.com/#prosjekter. (Accessed on 04/29/2020).

NWJS GitHub Page (udatert). https://github.com/nwjs/nw.js. (Accessed on 05/02/2020).

Omari, Mouad El, Mohammed Erramdani og Abdelkader Rhouati (2020). „Getting Model of MVVM Pattern from UML Profile". eng. I: *International Journal of Recent Contributions from Engineering, Science IT* 8.1, s. 36–47. ISSN: 2197-8581. URL: https://doaj.org/article/5f1f86d5870346c7aaa1e82e41cad03e.

Pratas, António (2014). *Creating Flat Design Websites.* Packt Publishing. ISBN: 9781783980048. URL: http://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=771465&site=ehost-live.

Proff.no (apr. 2020). *Nep Norway AS - Oslo - Regnskap.* https://proff.no/selskap/nep-norway-as/oslo/tv-selskaper/IFJFDML01SK/. (Accessed on 04/29/2020).

TypeScript: Typed JavaScript at Any Scale. (Udatert). https://www.typescriptlang.org/. (Accessed on 05/03/2020).

What is Babel? · Babel (udatert). https://babeljs.io/docs/en/. (Accessed on 05/03/2020).

Wu, Huifeng mfl. (2005). „Chaos-Model Based Framework for Embedded Software Development". I: *Embedded Software and Systems.* Red. av Zhaohui Wu mfl. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 582–588. ISBN: 978-3-540-31823-1.

Wybrow, Michael mfl. (2014). „Interaction in the Visualization of Multivariate Networks". I: *Multivariate Network Visualization: Dagstuhl Seminar #13201, Dagstuhl Castle, Germany, May 12-17, 2013, Revised Discussions.* Red. av Andreas Kerren, Helen C. Purchase og Matthew O. Ward. Cham: Springer International Publishing, s. 97–125. ISBN: 978-3-319-06793-3. DOI: 10.1007/978-3-319-06793-3_6. URL: https://doi.org/10.1007/978-3-319-06793-3_6.

NTNU
Avdeling for informatikk og e-læring
Geir Maribu

1. I/O-administrasjon

Datamaskinen har to hovedoppgaver. Det er *prosessering* (kjøring av programmer) og *i/o-operasjoner* (lesing og skriving av data). Operativsystemets rolle når det gjelder i/o er å administrere og styre i/o-operasjonene og i/o-utstyret. Siden utstyret varierer mye både i funksjonalitet og hastighet, er det behov for mange forskjellige metoder for å styre dette utstyret. Disse metodene finnes i det som kalles for i/o-subsystemet til kjernen.

Det er spesielt to trender som kan være vanskelig å forholde seg til når det gjelder i/o-utstyr. For det første er det er behovet for standarder for å gjøre det enklere å inkludere nytt utstyr i eksisterende datamaskiner og operativsystemer. For det andre er det behovet for å utvikle nye i/o-enheter med nye og forbedrede egenskaper.

Denne utfordringen møtes med en kombinasjon av maskinvare- og programvareteknikker. Felles for disse teknikkene er maskinvarekomponenter som porter, busser og utstyrs-kontrollere som kan tilpasses til et vidt spekter av i/o-enheter. Variasjonen i de forskjellige utstyrsenhetene fanges opp av driverprogrammer som operer direkte mot maskinvaren og tilpasser seg den variasjonen som finnes der.

Mot brukerprogrammene derimot, dvs på motsatt side, presenterer driverne et standard grensesnitt med et standard sett av funksjoner, og kan på denne måten skjule alle de detaljerte og maskinspesifikke egenskapene som finnes i utstyret.

1.1. Litt om i/o-maskinvare

Det er en svært stor variasjon i hva slags i/o-enheter som kan kobles til en datamaskinen, fra mus og tastatur koblet til hjemmemaskinen din, til f.eks joystick-en som brukes for å styre et jagerfly.

Men tross denne variasjonen er det et begrenset antall begreper vi trenger for å forstå hvordan utstyrsenheter er koblet til maskinen, og hvordan de kan styres av operativsystemet.

En ustyrsenhet kommuniserer med datamaskinen ved å sende signaler via en kabel eller trådlåst via radiosignaler. Kommunikasjonen foregår via en *port*, for eksempel en serieport eller en USB-port. Dersom flere enheter deler på et sett av ledere, kalles det en *buss*, f.eks en PCI-buss.

En *kontroller* er en maskinvare-enhet som står i tilknytting til en port, en buss eller direkte mot en utstyrsenhet. Det finnes f.eks en kontroller på serie-porten, ja man kan også si at serieporten er en kontroller. Denne kontrolleren styrer trafikken av signaler på ledningene i serieporten. Tilsvarende finner en også kontrollere for harddisker, nettverkskort etc.

Operativsystemet styrer operasjonene til i/o-enheten via registre som finnes i kontrolleren. Dette er registre som operativsystemet bruker for å legge inn en kode for aktuell i/o-operasjon. Videre er det registre for avlesing av status til i/o-utstyret. I tillegg finnes det dataregistre og/eller buffere der selve dataoverføringen skjer.

Dataoverføringen mellom cpu og kontroller-registrene kan skje på to måter:

- Ved å bruke spesielle i/o-instruksjoner som opererer direkte mot kontrollerregistrene. Hvert register har sine egne i/o-port-adresser.
- Alternativt kan utstyrskontrolleren støtte et system med memory-mapped i/o. Det betyr at kontrollerregistrene er "mappet" til minneadresser i cpu-en sitt adresseområde. Det betyr at alle kontrollerregistrene kan nås via ei vanlig minneadresse, og alle instruksjonene for lagring og henting av data fra minnet, kan også benyttes mot kontrollerne.

Mer om memory-mapped i/o i et seinere kapittel.

1.2. Hvordan gjøres i/o?

Nå handler det om tilkobling av utstyr til datamaskinen, og hvordan disse administreres via operativsystemet. Det handler med andre ord om i/o (input/output). Husk at utstyr er forskjellig, og derfor trenger operativsystemet egne programmer for å kunne bruker utstyret. Disse programmene kalles for *utstyrs-drivere*. Via driverne kan f.eks brukerprogrammer skrive til en enhet eller lese fra en enhet. Operativsystemet gir oss, og dermed brukerprogrammene et idealisert grensesnitt mot utstyrsenhetene.

Operativsystemet har også som oppgave å administrere utstyrsenhetene. Det er for eksempel nødvendig når flere prosesser ønsker å bruke samme ustyrsenhet på samme tid. For å få til dette må utstyrsenhetene representeres via såkalte deskriptorer, og prosessene må stilles i en kø til den aktuelle utstyrsenheten.

Det finnes mange metoder og systemer rundt dette med utføring av i/o:

- *Polling*: Den første krever at cpu, og dermed operativsystemet, er sterkt involvert med overføring av data til og fra ustyrets kontrollerregistre. Operativsystemet sjekker jevnlig et statusregister i kontrolleren om i/o-overføringen er ferdig. Denne sjekkingen av statusregister kalles *polling*.
- *Avbruddstyrt i/o*: Utstyret varsler sjøl operativsystemet når i/o-operasjonen er ferdig. Denne varslingen skjer via et signal direkte til cpu på en avbruddslinje.
- *Memory*-mapped i/o: Dette er en mye brukt metode for å adressere i/o-enhetene, dvs adressere registrene i kontrolleren. Med memory-mapped i/o skjer dette via adresser i minnet, og ikke via spesielle i/o-adresser.
- *Direkte minneaksess (DMA)*: Med denne metoden trenger ikke cpu delta i all data-overføringen. Nå skjer overføringen direkte fra i/o-enhet (dvs. fra kontrollerens registre) til minnet (RAM). Cpu er kun involvert i starten av overføringen. Deretter utføres overføringen direkte. Metoden er spesielt anvendelig ved store dataoverføringer, og moderne metoder for minneadministrasjon er helt avhengige av denne metoden.

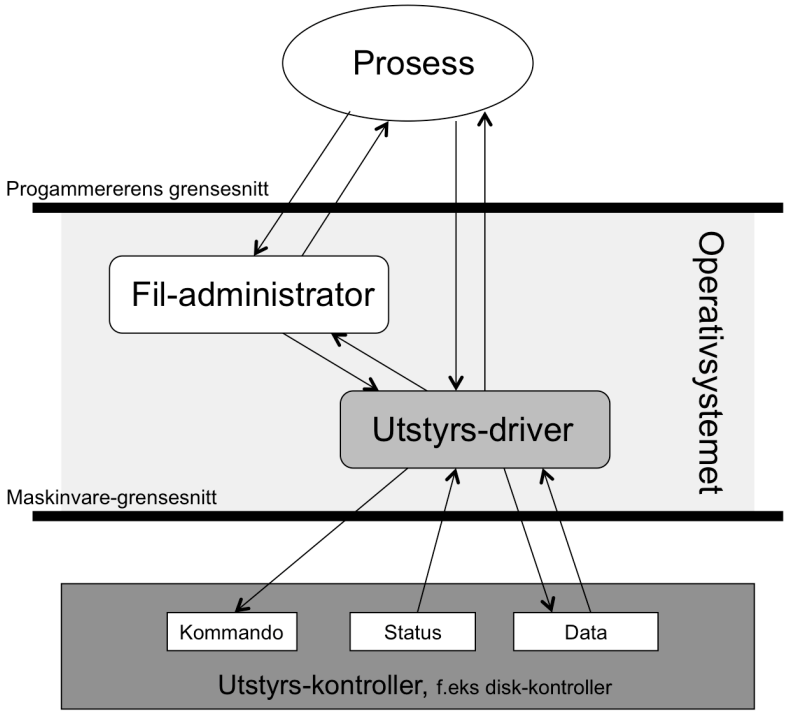
1.2.1. Organiseringen av i/o-systemet

Vi er interessert i i/o-enheter og administrasjon av disse fordi brukerprogrammene våre nødvendigvis må utføre i/o for å kunne hente inn data, og for at brukerprogrammene våre skal kunne sende fra seg data. Denne utvekslingen av data skjer via driverprogrammet til utstyret. Driveren har ansvaret for å tilby programmereren (dvs. den som lager brukerprogrammene) et grensesnitt mot funksjonaliteten som finnes i utstyret. Dette kalles også for et API (application programmers interface). Via API-et kan programmereren nyttiggjøre seg utstyret ved å kalle opp de funksjonene som finnes der.

Man tilstreber seg hele tiden å få disse API'ene så like som mulig for forskjellige utstyrs-drivere. Det gjør det enklere for programmererne som kan forholde seg til utstyret via et abstrahert og enhetlig grensesnitt som gjelder for alle driverne.

Forskjellig utstyr har forskjellige kontrollere. Driveren henvender seg til kontrolleren når utstyret skal brukes. Det innebærer at driveren bruker kontrollerens kommandoregister, statusregister og dataregister, og må forholde seg til hvilke kommandoer som skal brukes, hvordan innholdet i statusregistre skal tolkes, og hvilke krav utstyret setter til selve over-føringen av dataene.

Det er driveren sin oppgave å utligne disse forskjellene. API'ene er like og utstyret er forskjellig. Det er driverprogrammet sin oppgave å utligne disse forskjellene.



1.2.2. I/o med polling

Dette er en enkel, men langsom metode som ikke er så anvendelig i dagens superraske og effektive datamaskiner. Grunnen er at den krever mye oppmerksomhet fra cpu. Denne metoden overlater nemlig til cpu å overføre dataene mellom datamaskinens minne og data-registrene i utstyrskontrolleren.

Etter at cpu har satt i gang selve i/o-operasjonen må cpu jevnlig sjekke om i/o-operasjonen er ferdig. Dette gjør cpu-en ved at kontrollerens statusregister sjekker med jevne mellomrom.

039

NEP:GraphicsRouter
Visjonsdokument

Versjon 3.2

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
2. Oktober 2019	1.0	Oppstart og utfylling av mal	Fredrik Mediå
4. Oktober 2019	1.1	Ferdig utfylling av mal, venter på oppstartsmøtet	Fredrik Mediå
14. Oktober 2019	2.0	Redefinisjon av prosjekt	Fredrik Mediå
29. Januar 2020	3.0	Redefinisjon for bachelor	Fredrik Mediå
07. Februar 2020	3.1	Temaskifte fra Skisport til Skytesport	Fredrik Mediå
04. Mars 2020	3.2	Oppdatering av funksjonelle krav	Fredrik Mediå

Innholdsfortegnelse

1. Innledning

Hensikten med dette dokumentet er å holde riktig kurs. Ønsket om et fungerende produkt er stort, men veien din kan til tider virke vanskelig. Dette dokumentet skal hjelpe med å huske hvilke valg vi har tatt. For å kunne vise poengene til en utøver som registreres av dommere eller digitale løsninger i grafikk trenger man å konvertere informasjon fra scoring-systemet til korrekt format. Produktet skal til slutt kunne rute og håndtere data fra diverse leverandører til grafiske elementer i Singular.Live. Produktet skal også være så fleksibelt at det skal kunne rute og manipulere data i en live TV-produksjon dersom det er nødvendig.

2. Sammendrag problem og produkt

Under skirenn, fotballkamper og andre idretter som sendes på TV er grafikken et vesentlig element. Sportskyting er intet unntak. For at en TV-seer skal kunne følge med på åtte skyttere som skyter samtidig er det vesentlig at blinker er synlige for å vise treff. Alternativt kan bare tall vises. Dette bestemmes av hvor sofistikert grafikken er. Derfor trenger NEP et grensesnitt som kan rute og manipulere data fra eksterne system, til den grafiske motoren. Grensesnittet skal fungere som en ruter hvor data som kommer inn enkelt kan vises vei til sin plass i grafikken.

Dersom dette grensesnittet skal være attraktivt for lavbudsjettsjobber kan ikke oppsett og håndtering være tidkrevende eller komplisert. Det skal aller helst være et grensesnitt som er selvforklarende, samtidig som det er fleksibelt og enkelt. Dersom grensesnittet fyller kravene vil det kunne bli operert av en person som har flere oppgaver. Da sparer man arbeidstimer i form av bemanning, samtidig som man opprettholder TV-seerens behov for grafisk fremstilling av tider, poeng og annen nyttig tilleggsinformasjon.

2.1 Problemsammendrag

Problem med	dagens løsning er mangel på profesjonalitet. TV-seeren ønsker ikke grafikk som ikke er tilpasset sporten, eller uegnet på TV. En nettside som «scrolles» gjennom er ikke tilstrekkelig. Eksisterende løsning som er utviklet er for dyr å leie da det har tilnærmet monopol på markedet.
berører	TV-seeren som stiller krav, arrangørs renomme og NEP som teknisk leverandør av sendingen.
som resultatet av dette	kan ikke NEP tilby grafikk til sportssendinger med lave budsjetter.
en vellykket løsning vil	gi sportssendinger med lave budsjetter et profesjonelt utseende. TV-seerne vil få den informasjon de er vant med fra lignende sendinger, og ikke minst gi NEP ett konkurransefortrinn som leverandør av grafikk til lavbudsjettssendinger.

2.2 Produktsammendrag

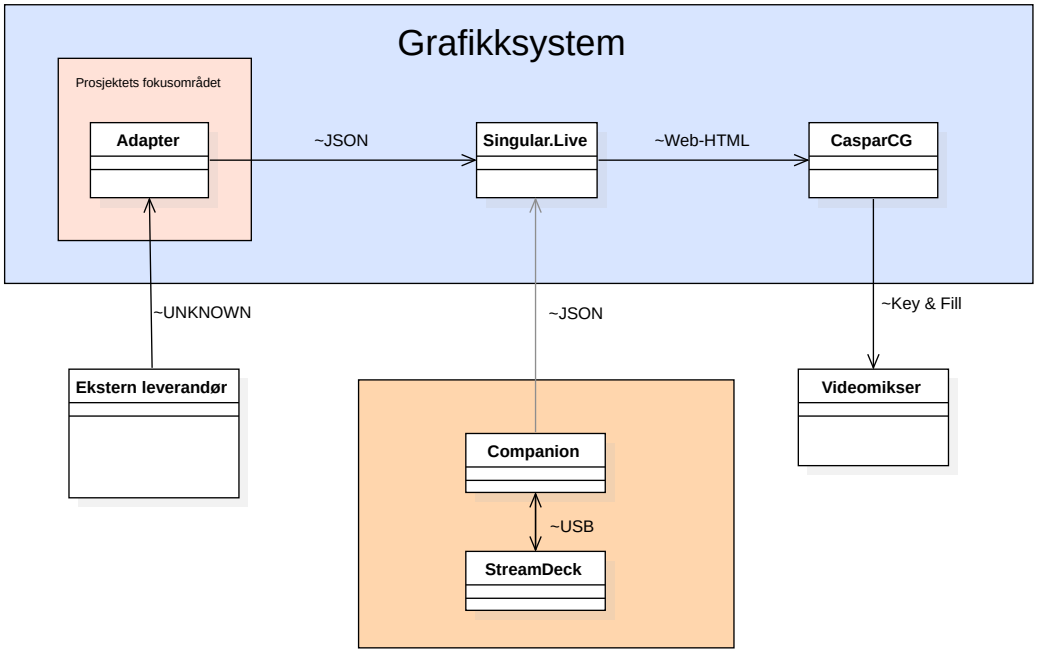
For	NEP
som	har behov for et grensesnitt som håndterer data fra eksterne kilder, og formaterer det til riktig format før det videreformidles til grafikkmotoren.
produktet navngitt	NEP:GraphicsRouter
skal	være er at det er allsidig grensesnitt som takler flere typer ekstern data, samtidig som det er **stabilt**. Det skal også være budsjettvennlig.
I motsetning til	andre dyre løsninger som for eksempel VizRT
har vårt produkt	til formål å kunne være et alternativ for lavbudsjetts arrangementer som ønsker å ta sine sendinger til et profesjonelt nivå uten budsjettspreng.

3. Overordnet beskrivelse av interessenter og brukere

3.1 Oppsummering interessenter

Navn	Utdypende beskrivelse	Rolle under utviklingen
NEP v/ Stig-Roar Aftret	Er bedriften som trenger dette systemet for å kunne tilby et profesjonelt uttrykk selv på lavbudsjettsproduksjoner	NEP kommer til å være oppdragsgiver, og komme med innspill til hvilke problemer som må løses før det tas i bruk.
Arrangør	Er kunden til NEP og arrangerer for eksempel skirenn, fotballkamp eller lignende	Arrangør har ikke en spesifikk rolle under utvikling, men kan bidra med egne grafiske elementer som kan benyttes av systemet.
TV-seer	Er bedømmer av sluttproduktet	TV-seeren er en taus part som ikke bidrar i utviklingen. Man hører ikke fra TV-seeren før det er noe som er feil. Dersom man ikke hører noe fra de, kan grafikken se ut til å gi de det de ønsker.
Brukeren	Grafikkoperatøren er den som opererer grafikkprogrammet. Denne personen har ansvar for å følge instrukser fra produsent på ønsket grafikk.	Jeg har selv erfaring innenfor grafikkhåndtering og blir derfor selv ansvarlig for å bedømme brukergrensesnittets funksjonelle egenskaper.
Produsenten	Ansvarlig for sendingen. Produsenten har det fulle ansvaret for at produksjonen følger sendeplan og velger også hvilke kameraer som skal fortelle historien.	Produsentens rolle er å gi informasjon om hvilken grafikk som er ønsket slik at produktet håndterer alle ønsker.
EIC (Engineer in charge)	Teknisk ansvarlig i forkant og under produksjon. EIC bestemmer hvilket utstyr som skal brukes og hvor. Er det feil med komponenter er det EIC sitt ansvar å finne en erstattende løsning.	EIC stiller med teknisk innsikt i hvilket utstyr som er disponibelt i OB-bussene (Outside Broadcasting busser er produksjonsbussene).

3.2 Brukermiljøet



Figur 1 – Miljømodell som viser sammenhengen mellom de forskjellige delene i kjeden fra ekstern leverandør av data til grafikken når videomikseren. «Adapter» er grensesnittet som er omtalt som NEP:GraphicsRouter i dokumentet.

Eksterne leverandører leverer data i forskjellige formater. Dette er området hvor grensesnittet trenger å være fleksibelt. Det må kunne håndtere forskjellige formater, rute og manipulerer dette før det pakker det inn til et ønsket format for Singular.Live.

Singular.Live håndterer koblingen mellom dere REST-api og en såkalt web-output. En web-output er en webside som fungerer som en grafisk motor. Websiden tegner grafikk som designet i deres designverktøy, og animerer inn og ut grafikken. Dette gjør websiden basert på kommandoer.

CasparCG er en mye brukt play-out software for å håndtere flere lag med grafikk og konvertere dette til et signal som videomiksere forstår. CasparCG er et open-sourceprogram som er tilgjengelig i de fleste OB-busser.

Companion er bransjens nye tilskudd i programvare for å håndtere omlag 200 forskjellige enheter. Dette innebærer prosjektorer, videomiksere og lignende. Denne programvaren er godt etablert i bransjen og er foretrukket av mange. Det er derfor naturlig å lage en modul for Companion for å kunne gjøre enkle oppdateringer i grafikkens datanoder, samt animere inn og ut grafiske elementer. Companion er ikke en del av produktet, men fungerer som et tillegg til produktet. Siden programmet er så etablert i bransjen vil det å kunne samkjøre kommandoer for andre enheter med grafikken være en fordel. Kanskje skal lysene på et konferansesenter slokkes samtidig som en video skal spilles av på sendingen fra konferansen. Dette kan samkjøres på en trigger i Companion.

3.3 Alternativer til vårt produkt

Så vidt vi vet eksisterer det ingen alternativer til vårt produkt som er så fleksibelt. En produsent på et arrangement jeg var grafikkoperatør på hadde skaffet seg en løsning som fungerte bra. Dessverre har denne produsenten enerett på programmet, og leier ikke dette ut uten at han selv er tilstede. Programmet var spesialtilpasset EQTiming som har tidtakingssystemet eTiming. Programmet er ikke så fleksibelt som vårt produkt.

4. Produktoversikt

4.1 Produktets rolle i brukermiljøet

Produktets rolle er i hovedsak å fungere som en ruter mellom eksterne leverandører av data, og Singular.Live. Det vil si at NEP:GraphicsRouter ikke nødvendigvis er grafikksystemet, men sammen med leddene beskrevet i 3.2 Brukermiljøet utgjør det et grafikksystem.

Grensesnittet skal fungere som en plattform hvor grafikkoperatøren kan visuelt endre hvor informasjon skal rutes til. Det skal også være mulighet for grafikkoperatøren å endre data i for eksempel grafikk som viser navn og lignende.

4.2 Forutsetninger og avhengigheter

Scoringsystemer som arrangørene rundt om i Norge bruker varierer stort. Det vil si at data fra eksterne leverandører også vil varierer stort. Derfor er det viktig at grensesnittet kan håndtere forskjellige typer data. En modulær tilnærming hvor hver modul er tilpasset det eksterne systemet vil derfor være en naturlig løsning. Hver enkelt modul vil kunne kommunisere med ruteren og finne veien til riktig plass i grafikken.

Grensesnittet må også formatere dataen til etter konvensjon som Singular.Live krever til grafikk motoren. Dette må følge Singular.Lives REST-api sine regler. Enkelte kommandoer er universelle, som for eksempel hvordan man animerer inn og ut grafikk. Grafikken laget i Singular.Live kan også variere. Det er derfor viktig at det lages moduler som kommuniserer med Singular.Live slik at hvilke datapunkter som skal fylles med informasjon blir fylt med riktig data.

Singular.Live er en state-based grafikkmotor som håndterer visning og kontrollering av grafikk. Singular.Live henter ingen informasjon gjennom sine widgets. Singular.Live krever at man sender data gjennom deres REST-api.

Dersom grensesnittet ikke håndterer data fra eksterne leverandører og/eller kan kommunisere med Singular.Live sitt REST-api vil ikke grensesnittet være til nytte.

Det forutsettes altså at grensesnittet har moduler som er tilpasset for håndtering av data fra den spesifikke leverandøren og dens formater. En annen viktig avhengighet og forutsetning er internett-tilkobling. Singular.Live er skybasert og krever derfor en internettlinje for å fungere. Skal systemet funke uten internett må en annen grafikkmotor benyttes.

5. Produktets funksjonelle egenskaper

Ikke i prioritert rekkefølge

- Kunne hente data fra flere leverandører som ikke er avhengig av hverandre.
- Kunne manipulere data – som forkorting av navn.
- Kunne samle data som skal sendes til Singular.Live i en PUT-request for å spare datatrafikk over nettverket.
- Ha en oversiktlig og visuell ruter som brukeren kan betjene gjennom å dra streker mellom moduler.
- Brukeren må kunne logge på Singular.Live.
- Brukeren må kunne velge hvilken Show (i Singular.Live) som skal benyttes.
- Brukeren må kunne legge til nye moduler i ruteren under kjøring.
- Henting av data skal ikke effektere brukeren. GUI skal fremdeles være responderende.
- Ruting av data skal ikke effektere brukeren. GUI skal fremdeles være responderende.
- Manipulering av data skal ikke effektere brukeren. GUI skal fremdeles være responderende.
- Produktet skal alltid holde Singular.Live oppdatert med samme informasjon som kommer fra eksterne leverandører. Det skal alltid foreta nye kalkulasjoner og ikke stole på sist brukte informasjon.
- Brukeren skal ha mulighet til å navn sette modulene i ruteren for å lettere beskrive og kunne få oversikt over hva de gjør.
- Brukeren skal kunne få tilgang til Singular.Live-moduler som gjenspeiler de grafiske elementenes datanoder.
- Brukeren skal kunne definere sitt eget kontrollpanel.
- Ruteren skal kunne eksportere dens nåværende oppsett.
- Ruteren skal kunne importere et oppsett som tidligere er eksportert.
- Brukeren skal kunne eksportere sitt oppsett i kontrollpanelet
    - Avhenger av ruteren og må derfor også inneholde den eksporterte ruteren
- Brukeren skal kunne importere sitt oppsett til kontrollpanelet

5.1 Companions funksjonelle egenskaper

- Brukeren skal kunne logge seg på Singular.Live fra kontrollpanelet.
- Brukeren skal kunne oppdatere datanoder i grafikken hos Singular.Live.
- Brukeren skal kunne velge hvilket grafisk element som:
    - Brukeren skal kunne animere inn grafikk.
    - Brukeren skal kunne animere ut grafikk.
- Brukeren skal få oppgitt ShowID (tilhørende Singular.Live) som tilhører Show som er tilgjengelig for brukeren.

6. Produktets ikke-funksjonelle egenskaper

- Produktet skal kunne benyttes uavhengig av operativsystem. Det skal hovedsakelig fungere på MacOS og Windows.
- Kildekode skal ikke være offentlig tilgjengelig. Privat GitHub depot kan benyttes.
- Produktet skal være intuitivt for en teknikker som har erfaring innenfor tv-produksjoner.
    - Det innebærer kort opplæringstid. Skal testes på ansatte i NEP.
    - Gjenkjennelsesfaktor fra programvare og arbeidsmetoder som kjent for NEP Norway.

7. Referanser

- CasparCG: https://casparcg.com/
- Companion: https://bitfocus.io/
- Singular.Live: http://www.singular.live/
- Vizrt: https://www.vizrt.com/

039

NEP:GraphicsRouter
Kravdokumentasjon

Versjon 3.2

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
07. Februar 2020	3.1	Skiftet tema fra skisport til skyting og opprettelse av kravdokumentasjon som følger visjonsdokumentets versjon	Fredrik Mediå
04. Mars 2020	3.2	Oppdaterte krav i visjonsdokumentet gir flere user stories	Fredrik Mediå

Innholdsfortegnelse

1. Introduksjon

Dette dokumentet tilhører utviklingsprosjektet som omfavner NEP:GraphicsRouter. Prosjektet er en del av bacheloroppgaven til Fredrik Mediå. Hensikten er å bidra under utviklingen av programvaren med å holde brukeren i fokus. Dokumentet inneholder user stories og wireframes som er referanser til høstprosjektets applikasjon.

2. User Stories

```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker jeg å velge hvilken kilde data skal komme fra
Slik at jeg kan bruke det i rutersystemet.


Scenario: Begynne oppsett av dataflyt for grafikk
Gitt at jeg vet hvem som er kilde til data
Når jeg velger den riktige kilden
Så skal jeg legge inn kilderelatert informasjon som IP og port,
brukernavn og passord dersom det trengs.
Og modulen til denne kilden skal så være tilgjengelig i ruteren.
```

```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker jeg å velge hvilket show i Singular.Live som skal benyttes
Slik at jeg sender ferdig formatert data til riktig grafiske element.


Scenario: Benytte et gitt grafisk oppsett
Gitt at jeg er logget på Singular.Live
Når jeg velger riktig show
Så skal jeg få tilgang til de grafiske elementene i ruteren
Og muligheten til å dra koblinger til disse.
```

```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker jeg å endre data
Slik at dataen i tv-grafikken får riktig format


Scenario: Endre formatet på navn til utøver
Når jeg legger til en modul som korter ned navn
Så skal jeg kunne dra koblinger gjennom denne modulen
Og modulen vil da forkorte navnet.
```

```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker jeg å sende data til Singular.Live
Slik at tv-grafikken viser data fra eksterne leverandører
```

```
Scenario: Data er ferdig formatert og skal sendes til Singular.Live
Når jeg legger til moduler som representerer grafiske elementer
Så skal jeg kunne dra koblinger til disse for å fylle grafikken med
data
Og modulen vi pakke inn dataen slik Singular.Live ønsker.
```

```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker jeg å trekke tråder mellom eksterne og singular
Slik at jeg visualiserer flyten


Scenario: Det skal lages et oppsett for bruk i en tv-produksjon
Når jeg legger til moduler i ruteren
Så skal jeg kunne dra koblinger mellom disse slik at data blir
håndtert på riktig måte
Og dataflyten blir visualisert.
```

```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker jeg å kunne endre data mens man er live
Slik at eventuelle feil i formatering kan rettes uten at systemet må
startes på nytt.


Scenario: Navnet er ikke forkortet, og tar for stor plass i grafikken
Når jeg legger inn en navneforkortermodul
Så skal jeg kunne dra koblinger gjennom denne uten at systemet feiler
Og resten av grafikken fungerer som den skal.
```

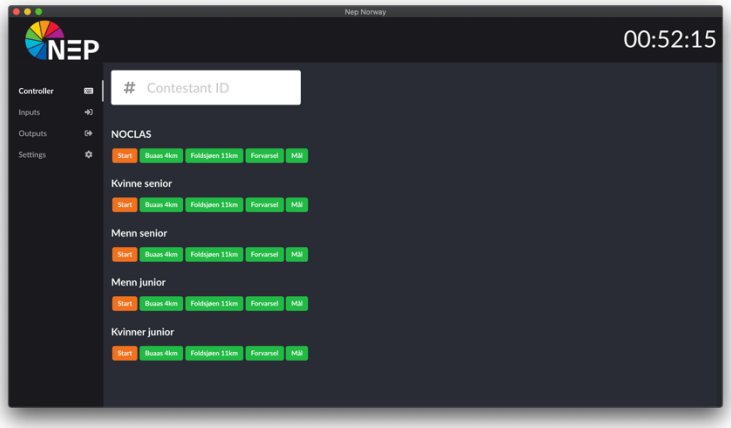
```
Som grafikkoperatør eller teknisk ansvarlig
Ønsker opprette mitt eget kontrollpanel
Slik at jeg vet hvor jeg har knapper og raskt kan respondere


Scenario: Brukeren trenger en knapp for å endre navn til kommentator
Når oppretter en knapp i kontrollpanelet
Så skal jeg kunne knytte denne til endringer i kontrollpunkt i
ruteren.
Og ved å trykke på knappen skal endringen skje.
```

3. Prototype

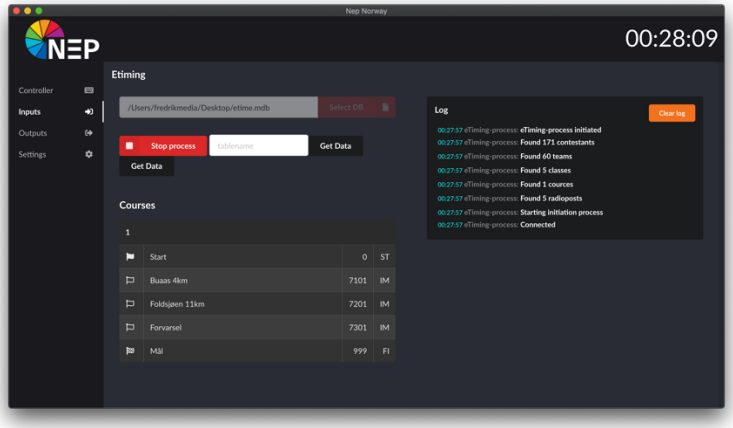
Brukergrensesnittet bygger på den samme oppbyggingen som høstprosjektet. Det er derfor ikke laget wireframes til de fleste brukersidene, men brukt screenshots. Wireframes er laget i Adobe XD.

3.1 Kontrollpanelet



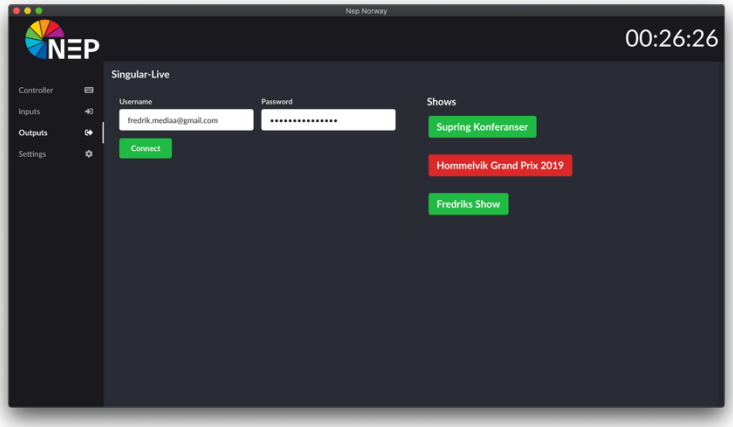
Skjermbildet er fra høstprosjektet og representerer hvordan et kontrollpanel kan bli bygget opp.

3.2 Inputs



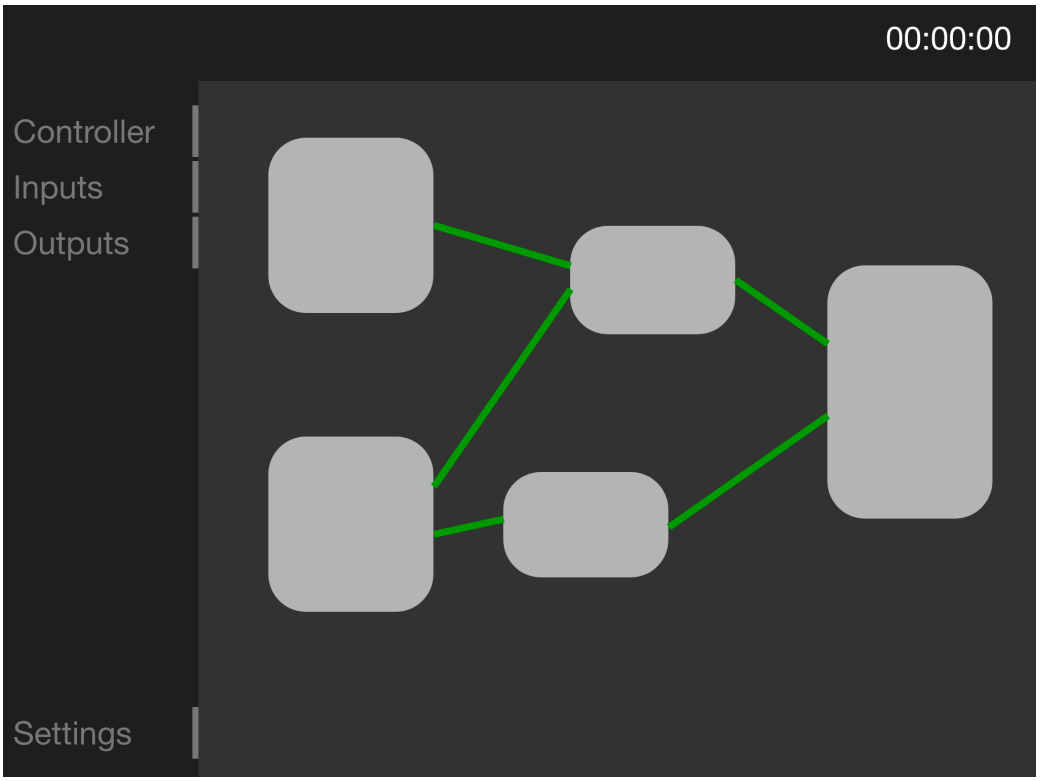
Skjermbildet er fra høstprosjektet og representerer hvordan det kan se ut å koble på eTiming.

3.3 Outputs (Singular.Live)



Skjermbildet er fra høstprosjeket og representerer hvordan det kan se ut når brukeren logger på Singular.Live

3.4 Mapper (Ruter)



Wireframe av hvordan mapper (ruteren) kan se ut. De grå feltene representerer noder som forandrer og formaterer data. De grønne linjene er koblingene mellom.

4. Referanser

- Singular.Live: http://www.singular.live/

039

Grafikkprosjekt
Systemdokumentasjon

Versjon <3.0>

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
28/12/2019	2.0	Første utkast etter omstrukturering	Fredrik Mediå
29/12/2019	2.1	Fjerning av kapitler, med begrunnelse	Fredrik Mediå
03/01/2020	2.2	Ferdigstilling av dokument	Fredrik Mediå
14/05/2020	3.0	Omskrevet til bachelor	Fredrik Mediå

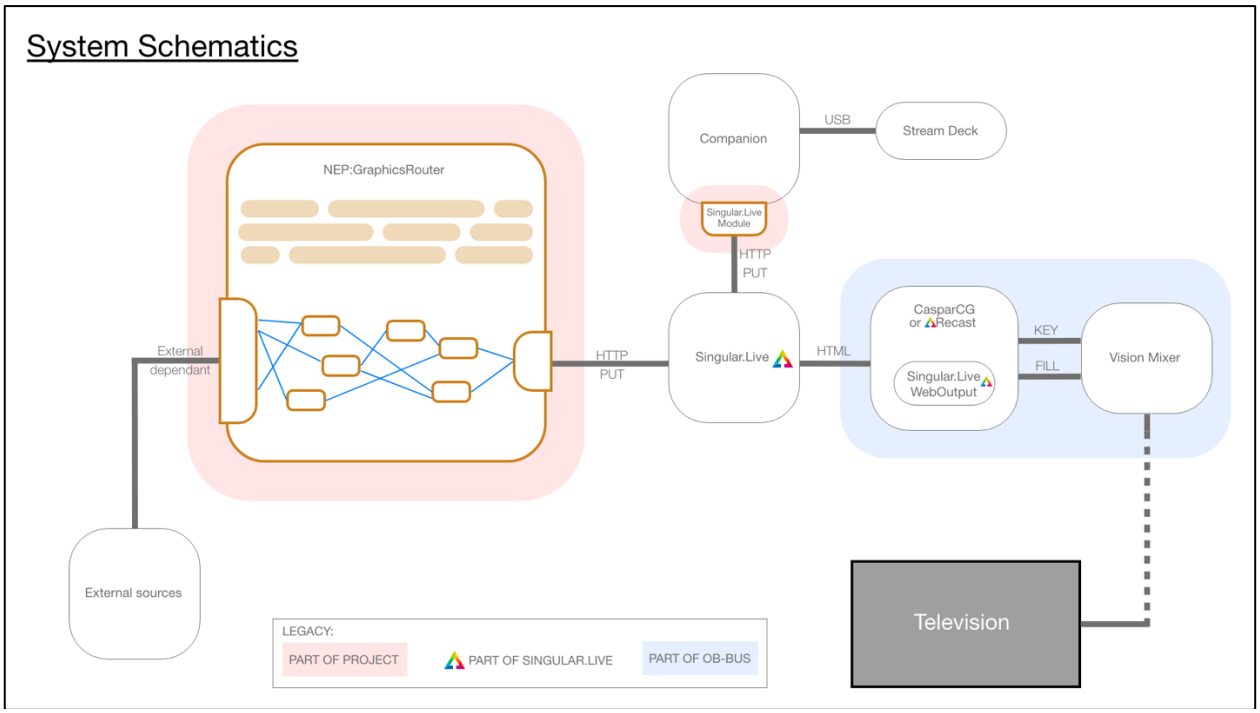
Innholdsfortegnelse

1. Introduksjon

Dette dokumentet skrevet i forbindelse med prosjektet NEP:GraphicsRouter, en bacheloroppgave av Fredrik Mediå for NEP Norway AS. Hensikten med dokumentet er å skape et oversiktlig bilde av prosjektet oppbygging og struktur. Dokumentet inneholder arkitekturskisser og helhetlige systemtegninger som forklaringer sammenhengen mellom kode og maskinvare, samt en guide i oppsett av utviklingsmiljø.

Tegninger er på engelsk av den grunn at NEP Norway er en del av et verdensomspennende konsern, hvor dette produktet i senere tid mulig skal brukes.

2. Arkitektur

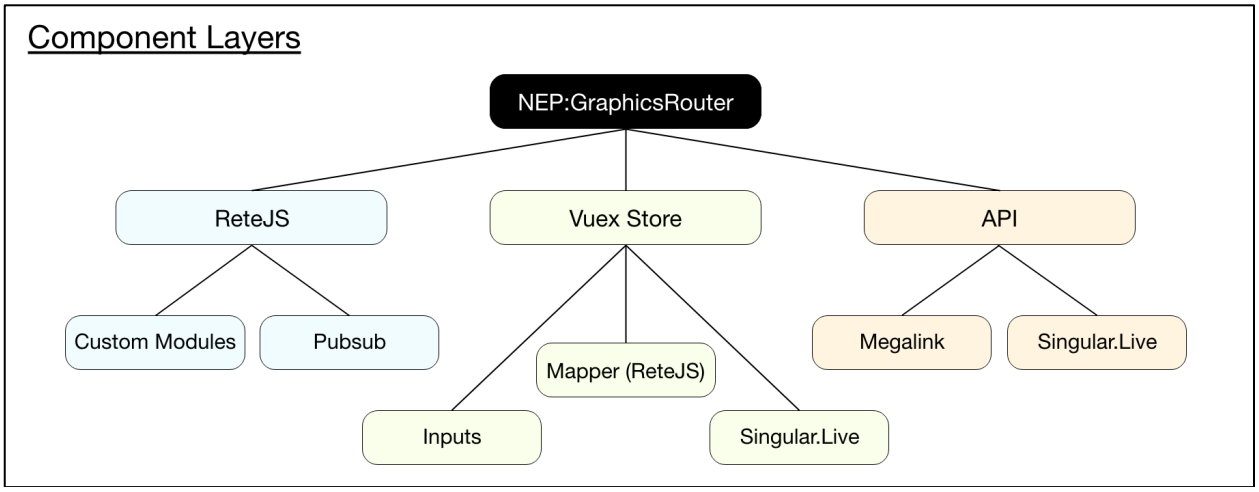


Figur 1 – Skisse over det komplette systemet, hvordan de forskjellige komponentene kommuniserer og hvor i produksjonen de hører hjemme. Markeringer med rød bakgrunn tilhører dette dokumentet. (OB-Bussen med blå markering, er produksjonsbussen og står for Outside Broadcasting bus).

Systemet i sin helhet består av flere ledd. Skissen over er en forenklet versjon som tar for seg de viktigste delene for å forstå hvordan det totale systemet skal fungere. NEP:GraphicsRouter og Singular.Live Module er bare en liten del av den totale pakken. Likevel er disse leddene vell så viktige for at systemet skal fungere.

Eksterne kilder gir fra seg data til NEP:GraphicsRouter gjennom kommunikasjonsform valgt av den eksterne. NEP:GraphicsRouter sender denne dataen gjennom sitt interne flytdiagrammet, før den sender dette til Singular.Live med HTTP-PUT spørringer. Companion med Singular.Live-modulen sender informasjon som animasjon in og ut for de grafiske elementene. Singular.Live overfører data til sine web-outputs, som er htmlsider med websockets. Disse sidene kan lastes inn i programmer som CasparCG og Recast, som igjen omformer signal til key og fill. Key og fill er to signaler som hjelper videomiksere å skille mellom hva som er gjennomsiktig og ikke i grafikken. Ut av videomikseren kommer et signal som er klart for TV-er.

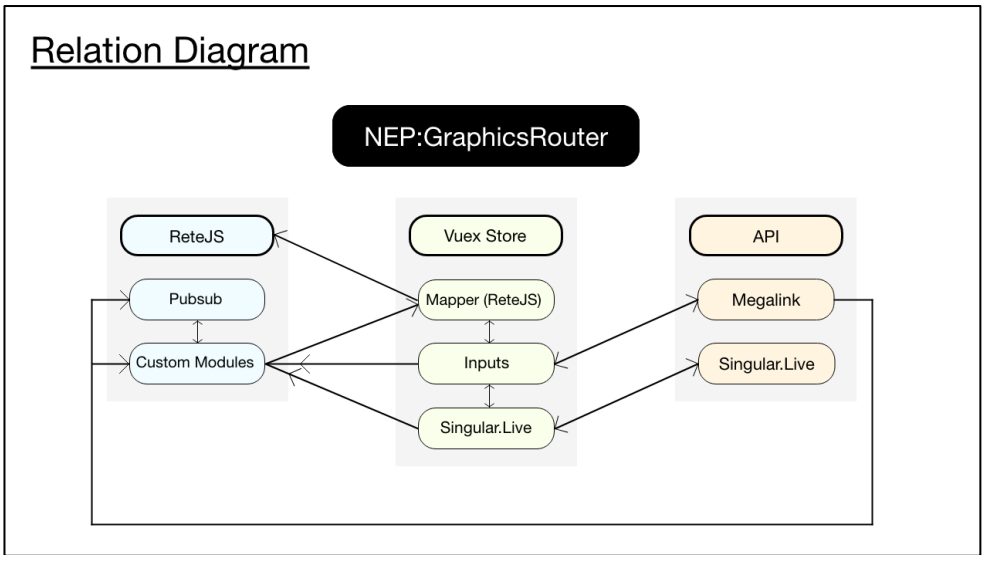
2.1 Applikasjonens hovedkomponenter



Figur 2 – Skisse over hovedkomponentene i applikasjonen.

ReteJS er den desidert største komponenten i systemet. Denne komponenten håndterer visualisering og logikken bak flytdiagrammet. Flytdiagrammet bruker ReteJS sin motor for kalkulasjoner og eventhåndtering. Vuex Store er Vue sin versjon av Redux. Vuex er altså et samlepunkt for data som automatiserer watchers. Watchers ser etter endringer i variablene og sender ut endringene til alle som bruker variablene. API er en samling av klasser som kommuniserer med gitte eksterne endepunkter.

2.2 Relasjonsdiagram



Figur 3 – Skisse over relasjoner mellom komponentene.

Figuren viser relasjoner mellom komponentene og hvilken vei relasjonene går. Det er også relasjoner innad mellom de forskjellige modulene i samme kolonne.

3. Prosjektstruktur

Prosjektet består av to separate deler. NEP:GraphicsRouter følger standard oppbygging av filstruktur for VueJS med ElectronJS applikasjoner. Mappen src inneholder all kildekode som relaterer seg til applikasjonen. Companion-module-singularlive-studio er modulen som er skrevet for å kontrollere grafikk fra applikasjonen Companion av Bitfocus. Filstrukturen til denne modulen er bygget på anbefalt oppbygging av Bitfocus.

Filstruktur:

NEP:GraphicsRouter
- public
  - index.html
- src
  - api
    - fetchWrapper.ts
    - megalinkAPI.ts
    - singularLiveAPI.ts
  - assets
    - *bilder*
  - plugins (Vue plugins)
    - vuetify.ts
  - retejs
    - modules
      - Controls
        - TextControl.ts
      - customization
        - Control.vue
        - Node.vue
        - Socket.vue
        - TextControl.vue
      - Generic
        - LoggerModule.ts
        - NameChangerShortModule.ts
        - TexttModule.ts
      - Megalink
        - MegalinkFlagModule.ts
        - MegalinkLanePickerModule.ts
        - MegalinkLaneSplitterModule.ts
        - MegalinkModule.ts
      - SingularLive
        - SingularLiveSendModule.ts
        - SingularLiveTemplateModule.ts
      - index.ts
    - plugins
      - taskpluginsv2
    - pubsub.ts
    - rete.ts
    - savedEditor.ts *(only for development)*
    - Sockets.ts
  - router
    - index.ts
  - store
    - modules
      - inputStore.ts
      - mapperStore.ts
      - singularLiveStore.ts (output)
    - index.ts
  - views
    - About.vue
    - Dashboard.vue
    - Home.vue
    - Inputs.vue
    - Mapper.vue
    - SingularLive.vue
  - **background.ts (Electron Main prosessen)**
  - **main.ts (Electron Renderer prosessen)**
  - shims-tsx.d.ts
  - shims-vue.d.ts
  - yarn.lock
- babel.config.js
- package.json
- tsconfig.json
- README.md

Companion-module-singularlive-studio
- lib
  - actions.js
  - actionsUI.js
  - api.js
- yarn.lock
- index.js (Module entry point)
- package.json
- HELP.md
- README.md
- LICENSE (Bitfocus)

3.1 NEP:GraphicsRouter

Filstrukturen til NEP:GraphicsRouter er som nevnt basert på VueJS og ElectronJS sine preferanser. Innenfor mappen src er mapper fordelt etter komponenter. Views inneholder alle html-templates samt logikken som er representert i brukergrensesnittet.

ReteJS er den største komponenten og har en egen mappe, hvor hver node som er tilgjengelig i flytdiagrammet eksisterer i respektive mapper. Generic er standardnoder som ikke er avhengig av en spesifikk forgjenger og kan brukes generelt i flytdiagrammet. Megalinkmappen inneholder noder som er spesifikk til håndtering av Megalink sine data, på lik linje som Singular.Live-mappen inneholder moduler som er tilpasset Singular.Live. SingularLiveTemplateModule.ts er en fil som genererer noder basert på datapunktene hos Singular.Live. Hvert grafiske element har sin egen representerte modul i flytdiagrammet, som er bygget av templatemodulen.

3.2 Companion-Module-Singularlive-Studio

Companionmodulen er bygget på et standard oppsett fra Bitfocus. Dette oppsettet gir oversikt over hvilke funksjoner knappene kan utføre ved å trekke de ut i actions.js, samt hvordan setter innstillingene til funksjonene i actionsUI.js. Api.js håndterer kommunikasjon mellom Companion og Singular.Live.

4. Installasjon og kjøring

4.1 Utviklingsmiljø

For å sette opp utviklingsmiljøet kreves det NodeJS og Yarn installert på maskinvaren.

Gå inn i rotmappen til NEP:GraphicsRouter og bruk følgende kommandoer:

1. *yarn install*
2. *yarn run electron:serve*

4.2 Bygge applikasjonen

For å bygge applikasjonen til en kjørbar fil brukes følgende kommando:

1. *yarn run electron:build*

4.3 Kjøre applikasjon

Siden applikasjonen er bygget med ElectronJS er det ingen installasjonwizard. Man starter programmet ved å åpne den kjørbare filen som ble laget i kapittel 4.2.

5. Referanser

- CasparCG: https://casparcg.com/
- Companion: https://bitfocus.io/
- Singular.Live: http://www.singular.live/

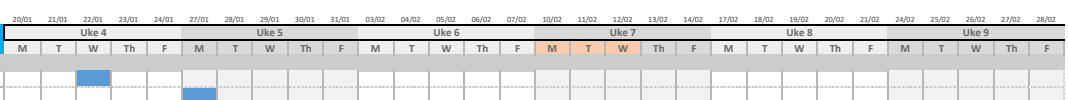
039

NEP:GraphicsRouter
Prosjekthåndbok

Innholdsfortegnelse

GANTT Diagram

TASK NAME	START DATE	END DATE	DURATION (WORK DAYS)	TEAM MEMBER
Administrativ / Undervisning				
Oppstartsmøtet	22/01	22/01	1	Fredrik
VT1	27/01	27/01	1	Fredrik
VT2	29/01	29/01	1	Fredrik
Dokumentasjon				
Research og prototyping for videre planlegging	03/02	14/02	10	Fredrik
Ferdigstilling av første halvdel av GANTT	07/02	07/02	1	Fredrik
Systemdokumentasjon	17/02	21/02	5	Fredrik
Programmering				
Bygge modul mot Megalinks endepunkt	24/02	03/03	7	Fredrik
Bygge grafikkelementer i JavaScript	02/03	11/03	8	Fredrik
			0	
Testing av programvare				
NM 10m luftpistol på Ørlandet (også helg)	25/03	30/03	5	Fredrik / NEP
Brukertester på ansatte i NEP	31/03	03/04	4	Fredrik / NEP
			0	
Utbedring etter testing				
Implementere brukerønsker	06/04	10/04	5	Fredrik
Effektivisering av programvare	13/04	17/04	5	Fredrik
			0	
Ferdigstilling (Dokumentasjon)				
Rapportarbeid	20/04	15/05	20	Fredrik
Korrekturlesing	18/05	19/05	2	Fredrik
Innlevering	20/05	20/05	1	Fredrik



GANTT Diagram

Eksamen

Ørlandet

TASK NAME	START DATE	END DATE	DURATION (WORK DAYS)	TEAM MEMBER
Administrativ / Undervisning				
Oppstartsmøtet	22/01	22/01	1	Fredrik
VT1	27/01	27/01	1	Fredrik
VT2	29/01	29/01	1	Fredrik
Dokumentasjon				
Research og prototyping for videre planlegging	03/02	14/02	10	Fredrik
Ferdigstilling av første halvdel av GANTT	07/02	07/02	1	Fredrik
Systemdokumentasjon	17/02	21/02	5	Fredrik
Programmering				
Bygge modul mot Megalinks endepunkt	24/02	03/03	7	Fredrik
Bygge grafikkelementer i JavaScript	02/03	11/03	8	Fredrik
			0	
Testing av programvare				
NM 10m luftpistol på Ørlandet (også helg)	25/03	30/03	5	Fredrik / NEP
Brukertester på ansatte i NEP	31/03	03/04	4	Fredrik / NEP
			0	
Utbedring etter testing				
Implementere brukerønsker	06/04	10/04	5	Fredrik
Effektivisering av programvare	13/04	17/04	5	Fredrik
			0	
Ferdigstilling (Dokumentasjon)				
Rapportarbeid	20/04	15/05	20	Fredrik
Korrekturlesing	18/05	19/05	2	Fredrik
Innlevering	20/05	20/05	1	Fredrik

	02/03	03/03	04/03	05/03	06/03	09/03	10/03	11/03	12/03	13/03	16/03	17/03	18/03	19/03	20/03	23/03	24/03	25/03	26/03	27/03	30/03	31/03	01/04	02/04	03/04	06/04	07/04	08/04	09/04	10/04
	Uke 10					Uke 11					Uke 12					Uke 13					Uke 14					Uke 15				
	M	T	W	Th	F	M	T	W	Th	F	M	T	W	Th	F	M	T	W	Th	F	M	T	W	Th	F	M	T	W	Th	F

GANTT Diagram

TASK NAME	START DATE	END DATE	DURATION (WORK DAYS)	TEAM MEMBER
Administrativ / Undervisning				
Oppstartsmøtet	22/01	22/01	1	Fredrik
VT1	27/01	27/01	1	Fredrik
VT2	29/01	29/01	1	Fredrik
Dokumentasjon				
Research og prototyping for videre planlegging	03/02	14/02	10	Fredrik
Ferdigstilling av første halvdel av GANTT	07/02	07/02	1	Fredrik
Systemdokumentasjon	17/02	21/02	5	Fredrik
Programmering				
Bygge modul mot Megalinks endepunkt	24/02	03/03	7	Fredrik
Bygge grafikkelementer i JavaScript	02/03	11/03	8	Fredrik
			0	
Testing av programvare				
NM 10m luftpistol på Ørlandet (også helg)	25/03	30/03	5	Fredrik / NEP
Brukertester på ansatte i NEP	31/03	03/04	4	Fredrik / NEP
			0	
Utbedring etter testing				
Implementere brukerønsker	06/04	10/04	5	Fredrik
Effektivisering av programvare	13/04	17/04	5	Fredrik
			0	
Ferdigstilling (Dokumentasjon)				
Rapportarbeid	20/04	15/05	20	Fredrik
Korrekturlesing	18/05	19/05	2	Fredrik
Innlevering	20/05	20/05	1	Fredrik

Innkalling til oppstartsmøtet bachelor

Med avsluttende del for systemutviklingsprosjektet (TDAT3022)

Dato: 22. januar 2020
Tid: 15:00 - 16:30
Sted: NEP Norway AS avd. Trondheim, Haakon VIIs gate 17c

Saker:

Sak 1: Presentasjon
Presentasjon av systemutviklingsprosjektets resultat. Dette for at Sander skal kunne fullføre vurderingen av TDAT3022, og for at Helge skal få et innblikk i hva som er gjort. NEP kan gjerne komme med tilbakemeldinger om hva tenker så langt.

Sak 2: Drøfting
Drøfting av oppgaven. Det er viktig at vi setter klare grenser for hva som er innenfor oppgaven. Dersom oppgaven blir for stor vil den være vanskelig å fullføre. Vi må bli enige om hvilke deler som har høyest prioritet. Samtidig må vi finne en vinkling som kan gi en problemstilling i løpet av bacheloren.

Sak 3: Fremdriftsplan og møter
Fremdriftsplanen for prosjektet avhenger av hvilken del av prosjektet vi satser på. Møteplan med veileder foreslås til hver tredje uke. Det blir innkalling for hver av disse. Fremdriften i prosjektet er studentens ansvar.

Sak 4: Utviklingsprosess
Forslag til utviklingsprosess er fossefall. Produktet som skal utvikles er primært en adapter for å håndtere data fra et system og inn i et annet. Dersom brukergrensesnittet faller mer i fokus er kanskje smidig utvikling veien å gå.

Sak 5: Krav til dokumentasjon
Det er visse krav til dokumentasjon som er satt for studiet. Disse innebærer: visjonsdokument, systemdokumentasjon, kravdokumentasjon og hovedrapport. Disse eksisterer i form dokumentasjon brukt i TDAT3022-faget. Forslaget er å fortsette og videre utforme denne dokumentasjon etterhvert som produktet endrer seg.

Sak 6: Retningslinjer for vurdering
Kanskje Helge kan gi en kort oppsummering av hvilke retningslinjer som gjelder og hva som er fokuset for veileder og sensor.

Sak 7: Kontrakt
Underskriving av kontrakt. Denne kontrakten er standardkontrakten til NTNU. Den skal underskrives i tre eksemplarer. En til hver av partene.

Sak 8: Eventuelt

Referat oppstartsmøtet bachelor

Med avsluttende del for systemutviklingsprosjektet (TDAT3022)

Dato: 22. januar 2020
Tid: 15:00 - 16:00
Sted: NEP Norway AS avd. Trondheim, Haakon VIIs gate 17c
Tilstede: Stig Roar Aftret (NEP), Pål Arne Berg (NEP), Helge Hafting (NTNU), Daniel Hamstad (NEP), Alexander Holt (NTNU), Fredrik Mediå (Student)

Saker:

Sak 1: Presentasjon
Presentasjon av systemutviklingsprosjektets resultat. Dette for at Sander skal kunne fullføre vurderingen av TDAT3022, og for at Helge skal få et innblikk i hva som er gjort. NEP kan gjerne komme med tilbakemeldinger om hva tenker så langt.

Presentasjon ble gjennomført. Stig Roar var imponert over det som var blitt gjort så langt. Det var mer enn opprinnelig forespeilet. Sander dro etter presentasjonen.

Sak 2: Drøfting
Drøfting av oppgaven. Det er viktig at vi setter klare grenser for hva som er innenfor oppgaven. Dersom oppgaven blir for stor vil den være vanskelig å fullføre. Vi må bli enige om hvilke deler som har høyest prioritet. Samtidig må vi finne en vinkling som kan gi en problemstilling i løpet av bacheloren.

Problemstilling skal studenten og veileder fortsette å diskutere etterhvert som problemer oppstår. Foreløpige eksempel er:

- Personers tillit til klokken i grafikken, og hvordan man teknisk kan gjøre den så nært sanntid som mulig.

Sak 3: Fremdriftsplan og møter
Fremdriftsplanen for prosjektet avhenger av hvilken del av prosjektet vi satser på. Møteplan med veileder foreslås til hver tredje uke. Det blir innkalling for hver av disse. Fremdriften i prosjektet er studentens ansvar.

Ettersom studenten sitter hos bedriften vil møter med veileder hver tredje uke være tilstrekkelig. Studenten har ansvar for å kalle inn til møte. Dersom det trengs flere og hyppigere møter er det mulighet for det.

Sak 4: Utviklingsprosess

Forslag til utviklingsprosess er fossefall. Produktet som skal utvikles er primært en adapter for å håndtere data fra et system og inn i et annet. Dersom brukergrensesnittet faller mer i fokus er kanskje smidig utvikling veien å gå.

Studenten foreslår fossefall til utvikling av klokken. Dette vil være naturlig da man ikke kan lage slutten av systemet før man har en begynnelse. Utviklingen i det forrige prosjektet har vært preget mye prøving og feiling, og man må gjøre modulene steg for seg.

Dersom det blir mer aktuelt å fokusere på brukergrensesnittet kan smidig utvikling med sprinter være aktuelt. Da kan det bli gjort intervju med grafikk-operatører angående deres ønsker til systemet.

Sak 5: Krav til dokumentasjon

Det er visse krav til dokumentasjon som er satt for studiet. Disse innebærer: visjonsdokument, systemdokumentasjon, kravdokumentasjon og hovedrapport. Disse eksisterer i form dokumentasjon brukt i TDAT3022-faget. Forslaget er å fortsette og videre utforme denne dokumentasjon etterhvert som produktet endrer seg.

Veileder er enig i at en oppdatering av eksisterende dokumentasjon er greit når studenten videreutvikler et allerede eksisterende prosjekt. Visjonsdokumentet vil bli oppdatert for å passe mer beskrivelsen av bacheloroppgaven, mens systemdokumentasjon og kravdokumentasjon blir mer utfyllende på de nye delene av prosjektet.

Sak 6: Retningslinjer for vurdering

Kanskje Helge kan gi en kort oppsummering av hvilke retningslinjer som gjelder og hva som er fokuset for veileder og sensor.

Helge informerer om at det som ligger til grunn for vurdering er hovedsakelig dokumentasjon av prosjektet. Dette innebærer hovedrapport, visjonsdokument, kravdokumentasjon og systemdokumentasjon. Det skal også lages en prosjekthåndbok. Denne skal blant annet inneholde timelister.

Studentens kontakt og samarbeid med bedriften rundt prosessen kan også virke positivt på oppgaven dersom den er god.

Timelister skal sendes til veileder en gang i uken.

Sak 7: Kontrakt

Underskriving av kontrakt. Denne kontrakten er standardkontrakten til NTNU. Den skal underskrives i tre eksemplarer. En til hver av partene.

Kontrakt ble underskrevet og rettighetshaverne blir NEP.

Sak 8: Eventuelt

NEP tar kontakt med Nederland om mulighet for å få ferdige grafiske elementer til prosjektet.

Fredrik Mediå

Timelister for uke 4

FROM **20/01/2020** TO **26/01/2020**

User	Project	Task	Notes	Date	Duration
Fredrik Mediå	Bachelor	Oppstartsmøte	skriving saksliste og utsending av møteinnkalling.	20/01/2020	2:00
Fredrik Mediå	Bachelor	Oppstartsmøte	Forberedelse til oppstartsmøtet, møtet i seg selv, og opprydding etter møtet	22/01/2020	5:00
Fredrik Mediå	Bachelor	Oppstartsmøte	Renskriving av møtereferat	23/01/2020	0:30

WORK TIME

7:30

Fredrik Mediå

Timeliste for uke 5

FROM **27/01/2020** TO **02/02/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Undervisning	VT1 undervisning	27/01/2020	4:00
Fredrik Mediå	Undervisning	VT2 Undervisning	29/01/2020	3:00
Fredrik Mediå	Dokumentasjon	Formulering av Visjonsdokument, forberedelse til Workshop	29/01/2020	4:00
Fredrik Mediå	Undervisning	Seminar / Workshop Referat: - Problemstillingen: kan finnes litteratur. - Kan være aktuelt å prate med en person som sitter på gløshaugen. Han er aktiv i orienteringsmiljøet. - Teste med videoer med forskjellige delayer uten å fortelle hva de egentlig skal se etter.	30/01/2020	1:00

WORK TIME

12:00

Fredrik Mediå

Timeliste for uke 6

FROM **03/02/2020** TO **09/02/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Arkitekturdesign	Gjennomgang med NEP om hva de tenker om sluttproduktet. Dette resulterte i et ønske om en mer modulær struktur. Det er laget en kjapp skisse over denne strukturen. Det er også undersøkt hvordan en slik struktur kan gjennomføres, men det trenges fremdeles mer informasjon. Det er også utført en splitt av kode fra TDAT3022 og til TDAT3001 av hensyn til kontraktskifte.	03/02/2020	4:00
Fredrik Mediå	Programmering	Prototyping av blink til 10m Luftpistol og Singular Widgets	05/02/2020	2:30
Fredrik Mediå	Programmering	Fremdeles prototyping av blink 10m luftpistol	06/02/2020	3:00
Fredrik Mediå	Dokumentasjon	Ferdigstilling av første halvdel av gantt-diagrammet.	07/02/2020	3:15
Fredrik Mediå	Programmering	Megalink API Modul	07/02/2020	3:00

WORK TIME

15:45

Fredrik Mediå

Timeliste for uke 7

FROM **10/02/2020** TO **16/02/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Arkitekturdesign	Undersøker muligheten for å bruke PubSub Arkitektur	13/02/2020	2:30
Fredrik Mediå	Programmering	Research til Singular Live Widget utvikling, samt utvikling av en blinkskivegrafikk.	14/02/2020	6:30

WORK TIME

9:00

Fredrik Mediå

Timeliste for uke 8

FROM 17/02/2020 TO 23/02/2020

User	Task	Notes	Date	Duration
Fredrik Mediå	Arkitekturdesign	Undersøking av Threadsmuligheter i JS	19/02/2020	4:00

WORK TIME

4:00

Fredrik Mediå

Timeliste for uke 9

FROM **24/02/2020** TO **01/03/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Arkitekturdesign	Research på VueJS som rammeverk for å erstatte React. React har for komplisert config-setup for at systemet skal være vedlikeholdbart i langt løp.	24/02/2020	7:00
Fredrik Mediå	Programmering	Starte overgang til Vue.js med oppsett med CLI	25/02/2020	8:30
Fredrik Mediå	Programmering	Ombygging av gamle moduler fra React til å pass med Vue.js	26/02/2020	9:00
Fredrik Mediå	Programmering	Fungerende prototype av Output-Modul. Videre research og testing av node-red og reteJS for flow	27/02/2020	9:00
Fredrik Mediå	Programmering	Implementering av ReteJS	29/02/2020	5:00
Fredrik Mediå	Programmering	Videre implementasjon av ReteJS	01/03/2020	6:00

WORK TIME

44:30

Fredrik Mediå

Timeliste for uke 10

FROM 02/03/2020 TO 08/03/2020

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering	ReteJS Cusomization og laget en dynamisk SingularLive komponent til ReteJS	03/03/2020	7:00
Fredrik Mediå	Programmering	Videre programmering av reteJS og oppstart med VueWorkers. Møte med Kim Jørgen Vang (Produsent Ørlandet) om hva som trengs til sendingen. Telefonmøte med Megalink i Oslo for å klarere innhenting av informasjon fra deres server. Det ble også videreutvikling på Widgeten til SingularLive som viser blink.	04/03/2020	10:00
Fredrik Mediå	Programmering	Programmerer API for oppkobling mot Megalink. De sender text/html og ikke application/json. Vi må ta en samtale med Megalink for å få rettet opp dette.	05/03/2020	5:00
Fredrik Mediå	Programmering	Videre utvikling av widget til singularlive	06/03/2020	5:00

WORK TIME

27:00

Fredrik Mediå

Timeliste for uke 11

FROM **09/03/2020** TO **15/03/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering		09/03/2020	7:30
Fredrik Mediå	Programmering		10/03/2020	13:00
Fredrik Mediå	Programmering		11/03/2020	12:30

WORK TIME

33:00

Fredrik Mediå

Timeliste for uke 12

FROM **16/03/2020** TO **22/03/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering	Utvikling av singular live widget for bruk i grafikk hvor man skal ha to nesten like grafiske elementer til å vises basert på en variabel. Research rundt dette temaet.	20/03/2020	6:30

WORK TIME

6:30

Fredrik Mediå

Timeliste for uke 13

FROM **23/03/2020** TO **29/03/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering	Videre forsøk på widget for grafisk fremstilling basert på variabel. Samt undersøking av Composition Script i SingularLive	23/03/2020	7:30
Fredrik Mediå	Programmering	Klargjøring av betaversjon for widget og compscript for variabelwidget. Samt sammensetting av flere grafiske komponenter og deres logikk.	24/03/2020	9:00
Fredrik Mediå	Dokumentasjon	Oppsett av hovedrapport, og lesing av tidligere bacheloroppgaver.	25/03/2020	5:00
Fredrik Mediå	Programmering	Utvikling av logikk i grafikk	26/03/2020	2:00
Fredrik Mediå	Programmering	Utvikling av logikk i grafikk	28/03/2020	3:00
Fredrik Mediå	Programmering	Utvikling av logikk i grafikk	29/03/2020	3:00

WORK TIME

29:30

Fredrik Mediå

Timeliste for uke 14

FROM **30/03/2020** TO **05/04/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering	Utvikling av logikk i grafikk. Oppretting og testing av kontroll-noder mot system.	30/03/2020	5:00
Fredrik Mediå	Programmering	Utvikling av plasseringsprognose for megalinkmodul.	01/04/2020	9:00
Fredrik Mediå	Programmering	Videreutvikling av Task-plugin. Det er noen feil da den bruker 212 gjennomkjøringen fremfor 12. Slik den burde.	03/04/2020	8:30

WORK TIME

22:30

Fredrik Mediå

Timeliste for uke 15

FROM **06/04/2020** TO **12/04/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering	Undersøker løsninger for å gjøre ReteJS-kalkulasjonene mer effektive. Det er 212 kalkuleringer per gjennomkjøring på omlag 10 noder. Informasjon enderer seg ikke mellom hver gjennomkjøring og trenger derfor ikke så mange gjennomkjøringer. Det skal optimalt set kun være en kalkulajson per unike tilkoblet node.	11/04/2020	7:00
Fredrik Mediå	Programmering	Videre utvikling av ReteJS Taskplugin. Bruker utgangspunktet til den opprinnelige taskplugin, og legger inn egne modifikasjoner. Nå søker vi gjennom etter uniker tilkoblede noder før vi leter etter data.	12/04/2020	10:00

WORK TIME

17:00

Fredrik Mediå

Timeliste for uke 16

FROM **13/04/2020** TO **19/04/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Programmering	Oppdaget ved en tilfeldighet at nodeID alltid var lik høyeste nodeID. Dette gir derfor feil i kalkulasjoner med noder som har flere tilkoblede noder av samme type. Måtte derfor utvide TaskPlugin til å håndtere nodeID. Systemet fungerer nå som det skal igjen	13/04/2020	9:00
Fredrik Mediå	Dokumentasjon	Gjennomgang av TypeScript. Dette var en mye mer krevsom prosess en forventet. Estimert timebruk før start var 4 timer. Brukte 12 timer.	14/04/2020	12:00
Fredrik Mediå	Programmering	Research for utvikling av Companion Modul, oppsett av lokal utviklingsmiljø og utvikling av Modul. Skal sendes til produkteierne slik at modulen blir en del av den endelige løsningen.	15/04/2020	15:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel.	17/04/2020	7:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel.	18/04/2020	3:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel.	19/04/2020	3:45

WORK TIME

49:45

Fredrik Mediå

Timeliste for uke 17

FROM **20/04/2020** TO **26/04/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel.	20/04/2020	8:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel. Møte med veileder.	21/04/2020	8:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel	22/04/2020	8:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel.	23/04/2020	7:00
Fredrik Mediå	Dokumentasjon	Jobbing med hovedrapport teorikapittel.	24/04/2020	9:00
Fredrik Mediå	Dokumentasjon	Hovedrapport Teori	25/04/2020	4:00
Fredrik Mediå	Dokumentasjon	Hovedrapport Teori	26/04/2020	6:00

WORK TIME

50:00

Fredrik Mediå

Timeliste for uke 18

FROM **27/04/2020** TO **03/05/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Dokumentasjon	Hovedrapport Teori	27/04/2020	10:00
Fredrik Mediå	Dokumentasjon	Hovedrapport Metode	28/04/2020	10:30
Fredrik Mediå	Dokumentasjon	Hovedrapport Metode	29/04/2020	9:00
Fredrik Mediå	Dokumentasjon	Hovedrapport Metode	30/04/2020	10:30
Fredrik Mediå	Dokumentasjon	Hovedrapport Metode	01/05/2020	9:00
Fredrik Mediå	Dokumentasjon	Hovedrapport Metode	02/05/2020	10:00
Fredrik Mediå	Dokumentasjon	Hovedrapport Teori	03/05/2020	6:00

WORK TIME

65:00

Fredrik Mediå

Timeliste for uke 19

FROM **04/05/2020** TO **10/05/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Dokumentasjon	Resultater	04/05/2020	10:00
Fredrik Mediå	Dokumentasjon	Resultat, oppgavetekst og metode.	05/05/2020	10:00
Fredrik Mediå	Dokumentasjon	Resultater	06/05/2020	9:00
Fredrik Mediå	Dokumentasjon	Resultatkapittel	07/05/2020	9:30
Fredrik Mediå	Dokumentasjon	Resultatkapittel	08/05/2020	10:00
Fredrik Mediå	Dokumentasjon	Diskusjon	09/05/2020	9:00
Fredrik Mediå	Dokumentasjon	Diskusjon	10/05/2020	6:00

WORK TIME

63:30

Fredrik Mediå

Timeliste for uke 20

FROM **11/05/2020** TO **17/05/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Dokumentasjon	Hovedrapport diskusjon	11/05/2020	8:00
Fredrik Mediå	Dokumentasjon	Hovedrapport konklusjon + systemdokumentasjon, illustrasjoner til systemdokumentasjon	12/05/2020	8:00
Fredrik Mediå	Dokumentasjon	Hovedrapport metode og konklusjon + systemdokumentasjon	13/05/2020	9:00
Fredrik Mediå	Dokumentasjon	Hovedrapport + prosjekthåndbok + systemdokumentasjon	14/05/2020	9:00
Fredrik Mediå	Dokumentasjon	Korrekturlesinig av hovedrapport	15/05/2020	8:00
Fredrik Mediå	Dokumentasjon	Hovedrapport finpuss	16/05/2020	7:00

WORK TIME

49:00

Fredrik Mediå

Timeliste for uke 21

FROM **18/05/2020** TO **24/05/2020**

User	Task	Notes	Date	Duration
Fredrik Mediå	Dokumentasjon	Presentasjon og klargjøring til innlevering	18/05/2020	9:00
Fredrik Mediå	Dokumentasjon	Ferdigstilling og samling av alle dokumenter og levering i Inspera	19/05/2020	7:00

WORK TIME

16:00