Maritimt Innovasjonsprosjekt: Utvikling og Integrering av Digital Tvilling for Tunglabbens Maskinromslaboratorium

Jarl Benjamin Andersen Fredrik Nielsen Anders Bjerkengen

November 29, 2023

Innholdsfortegnelse

1 Introduksjon med formål

Tunglabben ved USN Campus Vestfold er i prosessen med å bli digitalisert. Denne labben er konstruert som et virkelighetsnært skipsmaskinrom utstyrt med hovedmotor, hjelpemotorer og diverse hjelpesystemer med et eget kontrollrom utstyrt med hovedtavle og maskinromkontrollkonsoll som har blitt brukt i undervisnings sammenheng. I de siste årene har USN, i samarbeid med Kongsberg Maritime, integrert automasjonssystemer som K-Chief600 og Ship@Web for å forbedre overvåkning og kontroll mulighetene for å forberede studenten på digitaliseringen som finner sted.

I samarbeid med faglærer vil vi forme en interaktiv HMI-løsning med sanntidsdata fra en database som simulerer data fra laboratoriets hjelpemotorer. Denne oppgaven vil også involvere videreutvikling av Tunglabbens 3D-modell i Unity3D med data fra databasen.

Målet med oppgaven er å videreutvikle den digitale tvillingen samt lage et brukergrensesnitt for å sortere data, og hente ut informasjon som skal presenteres på en gunstig og brukervennlig måte.

2 Kommunikasjonsoversikt

Kommunikasjonsoversikten under gir en fremstilling av nettverksstrukturen og dataflyten i Tunglabben. Sentralt i dette systemet er K-Chief 600 automasjonssystemet som tjener som hjernen i operasjonen, og behandler data fra ulike hjelpemotorer og sensorer. Med et nettverk bestående av LAN (Local Area Network) og CAN (Controller Area Network), er systemet utstyrt til å håndtere den den mengden informasjon som genereres av de tilkoblede enhetene.

To hovedroutere fungerer som bindeledd mellom skipets interne nettverk og den utvidede infrastrukturen, som tillater sikker datakommunikasjon både internt og eksternt. Dette inkluderer tilkobling til Ship@Web-serveren, som gir et ekstra lag med dataadministrasjon og tilgangskontroll gjennom en dedikert brannmur. Dette sikrer at sensitiv informasjon forblir beskyttet samtidig som den er tilgjengelig for autorisert personell via brukergrensesnittet for logging, trendanalyse og rapportering. Ship@Web-serveren er der dataen som vises i brukergrensesnittet er hentet fra.

Auxiliary Sensors, som en del av IoT (Internet of Things)-nettverket, overvåking av temperaturer i kjølevannsystemet og motorrommet samt luftfuktighet. Sanntidsdata til både AUX Engine 1 og AUX Engine 2 blir også hentet inn til Ship@Web-serveren. Denne integrerte tilnærmingen til nettverkskommunikasjon sørger for at operatører har umiddelbar tilgang til operasjonelle data, og gir dem muligheten til å monitorere flere fartøy. Oversikten under er viktig for å forstå hvordan data og kontrollsignaler navigerer gjennom systemets mange lag.

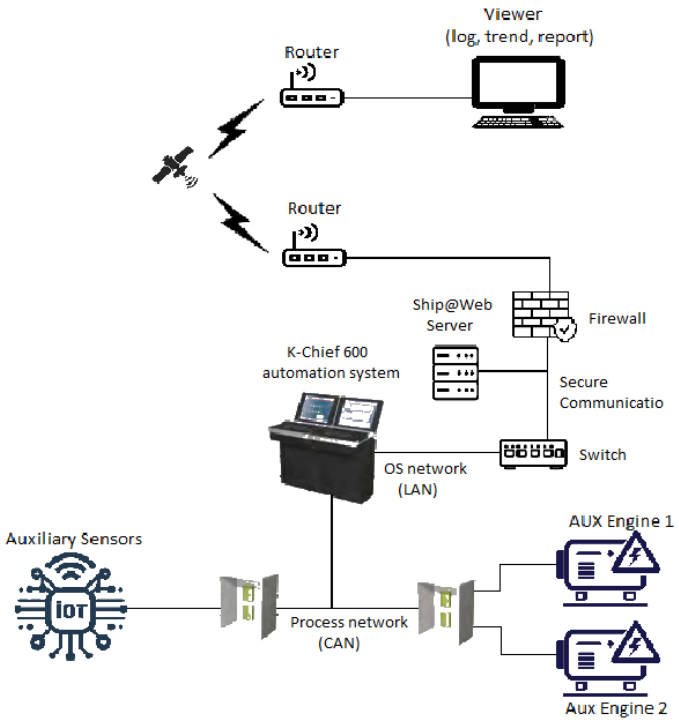

Figure 1: Kommunikasjonsoversikten

3 HMI Design

I utviklingen av brukervennlige og effektive systemer, er det avgjørende å ha en dyp forståelse av brukerens behov og opplevelser[1]. Dette har vi prøvd å oppnå med en metodisk tilnærming som omfatter bruk av personas, use case, use case slices og narrativ. Personas er fiktive representasjoner av de brukerne, gir et konkret bilde av målgruppen, deres behov og hvordan de sannsynligvis vil samhandle med systemet. Dette hjelper utviklere og designere å holde brukernes interesser og behov i sentrum gjennom hele design- og utviklingsprosessen.

Use case gir en strukturert beskrivelse av hvordan et system vil bli brukt i praksis. Det setter fokus på brukerens interaksjoner med systemet, og gir et klart bilde av systemets funksjonalitet og begrensninger. Ved å bryte ned komplekse systemer i mindre, håndterbare deler gjennom use case slices, kan utviklere fokusere på å levere verdifulle, brukervennlige funksjoner i trinnvise iterasjoner. Dette forenkler både utviklingen og testingen, og sikrer at hvert element av systemet er relevant og verdifullt for brukeren[3].

Til sist er narrativ – historier – et kraftfullt verktøy for å konkretisere brukstilfellene og personasene. Historier skaper en dypere forståelse av brukerens opplevelser og utfordringer ved å fortelle om deres interaksjoner med systemet i realistiske scenarier. Dette hjelper utviklerne med å visualisere brukerens reise, identifisere potensielle problemer og utvikle mer intuitive og effektive løsninger.[2]

Samlet sett gir integreringen av personas, use case, use case slices og narrativ et robust rammeverk for brukersentrert design og utvikling. Denne tilnærmingen fører til utvikling av systemer som ikke bare møter brukerens forventninger, men også beriker deres interaksjon med teknologien.

3.1 Personas

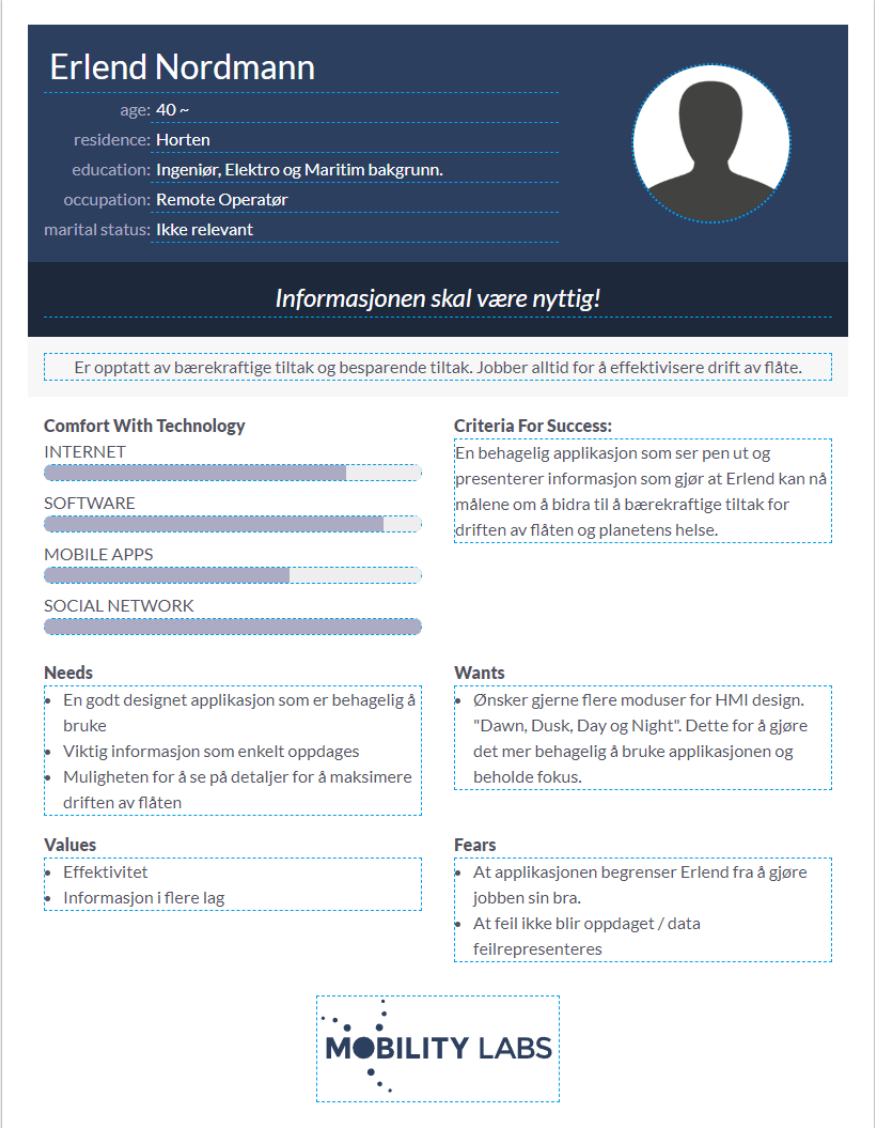

Figure 2: Persona

3.2 Use case

Denne use casen beskriver prosessen med å overvåke maskinromsdata ved hjelp av Ignition prespective HMI i Tunglabben. Casen starter når en ingeniør logger seg inn på systemsiden og slutter når de har fullført sine overvåknings- eller kontrolloppgaver.

Grunnleggende Flyt

1. Operatør logger seg inn på HMI-systemet.
2. Operatøren navigerer til den tekniske tabben for overvåking av motorprestasjon.
3. Sanntidsdata fra motorene vises på tabben.
4. Operatøren gjennomgår dataene for å se etter uregelmessigheter eller problemer.
5. Hvis et problem oppdages, varsler operatøren maskinisten via egen kommunikasjon på det gjeldene fartøyet og maskinisten gjør tiltak for å fikse problemet.
6. Operatøren logger alle betydelige observasjoner eller tiltak som er tatt.
7. Operatøren logger ut av HMI-systemet etter fullførte oppgaver.

Alternativ/Unntaksflyter

- Hvis sanntidsdata ikke er tilgjengelig:
  - Systemet viser en feilmelding eller varsling.
  - Operatøren sjekker systemtilkoblingen eller rapporterer problemet til det tekniske teamet.
  - Brukstilfellet avsluttes.

Forutsetninger

- HMI-systemet er operativt.
- Operatøren har gyldige innloggingsdata.
- Motorovervåkningssystemet er tilkoblet og overfører data.

Etterbetingelser

- Motorens parametere har blitt overvåket og meldt fra om etter behov.
- Alle viktige brukerhandlinger og observasjoner er logget.

Supplerende Krav

- HMI-systemet bør oppdatere sanntidsdata med jevne mellomrom.
- Systemet bør ha sikkerhetsmekanismer for nødssituasjoner.
- HMI-grensesnittet bør være brukervennlig og tillate rask tilgang til viktige funksjoner.

3.3 Use case slice: Brukerens mulighet til a logge inn

Denne slicen tar for seg brukerens mulighet til å logge inn sikkert og få tilgang til systemet, og sikrer riktig autentisering. Slicen starter når brukeren forsøker å logge inn og slutter når de enten har fått tilgang til systemet eller blir nektet tilgang.

Aktører

- Bruker (Mariningeniør, Tekniker, Instruktør)
- Autentiseringssystem

Forutsetninger

- Brukeren må ha gyldige påloggingsdata.
- Systemet er operativt.

Grunnleggende Flyt

1. Brukeren navigerer til innloggingssiden.
2. Brukeren taster inn sitt brukernavn og passord.
3. Systemet verifiserer påloggingsdataene.
4. Brukeren får tilgang til riktig nivå av systemet basert på deres rolle.

Alternativ/Unntaksflyter

- Ugyldige brukernavn eller passord:
  - Systemet viser en feilmelding.
  - Brukeren kan prøve å taste inn påloggingsdataene på nytt.
  - Etter flere mislykkede forsøk, låser systemet kontoen.
  - Brukstilfellet slutter.

Etterbetingelser

- Brukeren er enten logget inn og ser på dashbordet, eller er nektet tilgang.

3.4 Use case slice: Visning av sanntidsdata fra motoren

Denne slicen fokuserer på visning av sanntidsdata fra motorene i et forståelig format. Den begynner når brukeren får tilgang til datavisning og slutter når brukeren forlater datavisningen.

Aktører

- Bruker (Mariningeniør, Tekniker)
- Systemdatabase

Forutsetninger

- Brukeren er logget inn.
- Sanntidsdata er tilgjengelig fra motorene.

Grunnleggende Flyt

1. Brukeren velger alternativet for å vise sanntids motordata.
2. Systemet henter og viser dataene.
3. Brukeren ser på og tolker dataene.

Alternativ/Unntaksflyter

- Data ikke tilgjengelig:
  - Systemet viser en 'data ikke tilgjengelig' melding.
  - Brukeren returnerer til den forrige menyen.
  - Brukstilfellet slutter.

Etterbetingelser

- Brukeren har sett på sanntids motordata eller har blitt informert om at data ikke er tilgjengelig.

3.5 Use case slice: Varsling om unormale motordata

Denne slicen involverer systemets evne til å oppdage unormale motordata og varsle brukeren. Den starter når en anomali oppdages og slutter når brukeren anerkjenner varslingen.

Aktører

- Systemovervåkningsverktøy
- Bruker (Mariningeniør, Tekniker)

Forutsetninger

- Systemet overvåker aktivt motordata.

Grunnleggende Flyt

1. Systemet oppdager en unormal motordata.
2. Systemet genererer og viser en varsling.
3. Brukeren anerkjenner varslingen og utfører passende tiltak for å fikse problemet.

Alternativ/Unntaksflyter

- Brukeren klarer ikke å anerkjenne varslingen:
  - Systemet varsler en overvåkningssentral.
  - Brukstilfellet slutter.
- Brukeren navigerer til en egen alarmtab for å få bedre oversikt dersom det er flere alarmer pågående

Etterbetingelser

- Brukeren har blitt varslet om unormale motordata og har tatt passende handling for å løse problemet.
- Brukerens handlinger på alarm blir lagret i en egen journal.

3.6 Use case slice: Dataanalyse

Denne slicen involverer systemets evne til å gjøre analyse på data av motorparametere. Den starter når det ses behov for å se historikk på data eller sammenligne generatorer.

Aktører

- Systemovervåkningsverktøy
- Bruker (Mariningeniør, Tekniker)

Forutsetninger

- Systemet overvåker aktivt motordata og lagrer historikk på innsamlet data.

Grunnleggende Flyt

1. Bruker er innlogget på systemet og navigerer til table tabben.
2. Bruker velger flåte og filtrerer data etter ønskede generatorsett som ønskes å analyseres.
3. Bruker ser live-verdier og vurderer at ett generatorsett har dårligere verdier.
4. Bruker navigerer til trends tabben for å se på historikk til valgte verdier gjennom filtrering-funksjon.
5. Bruker har fått samlet nødvending informasjon til å ta nødvendig handling.

Alternativ/Unntaksflyter

- Brukeren mangler historisk data etter brutt kontakt med database.
  - Systemet viser ikke verdier i ett gitt tidsintervall.
- Alarmer har trigget, og bruker benytter seg av trender til å analysere hva som har skjedd i forkant av utfallet.

Etterbetingelser

- Bruker har samlet nødvendig informasjon av enten unormale data, eller sett forbedringspotensialer etter analyse, og tar passende handling.

3.7 Narrativ – Historier

I dette avsnittet er det laget historier. Bruk av historier i systemutvikling en god tilnærming som bidrar til en dypere forståelse og bedre kommunikasjon mellom alle involverte parter i et prosjekt. Disse historiene gir liv til tekniske spesifikasjoner ved å illustrere hvordan systemfunksjoner vil fungere i praktiske, dagligdagse situasjoner. Dette brukersentrerte perspektivet hjelper utviklerteamet å forstå og prioritere brukernes behov, samtidig som det fremmer empati og forståelse for sluttbrukerens opplevelse. Ved å konkretisere abstrakte konsepter, gjør historiene det enklere for både tekniske og ikke-tekniske teammedlemmer å identifisere potensielle problemer og muligheter, noe som fører til mer effektiv planlegging, utvikling og testing av systemet. Samlet sett bidrar bruken av historier til å skape mer brukervennlige og vellykkede systemer som er tilpasset de virkelige behovene til de som skal bruke dem.

Unormale temperaturer i kjølevannet

Operatøren Mari overvåker motorytelsen om bord på et skip. Plutselig oppdager systemet unormale verdier i kjølevannstemperaturen, temperaturen betydelig høyere enn det normalt skulle vært. En alarm kommer opp i alarmtabellen. Mari mottar varslingen og vurderer situasjonen. Hun melder fra til maskinisten som identifiserer en feil i kjølesystemet og setter i gang en prosedyre for å forhindre motorskade. Takket være den tidsriktige varslingen klarer Mari å håndtere problemet før det fører til en kritisk feil.

Rutinemessig Innlogging og Systemtilgang

Jonas, en ny operatør, skal starte sin første dag med å overvåke motorytelsen på systemet. Han taster inn sitt tildelte brukernavn og passord for å få tilgang til systemet. Først skriver han feil passord, og systemet nekter tilgang med en feilmelding. Jonas taster inn legitimasjonen sin riktig og får tilgang til systemet. Han navigerer til motordata-delen og begynner sine rutinemessige overvåkningsoppgaver, og gjennomgår sanntidsdataene som vises på det tekniske dashbordet.

Håndtering av Utilgjengelige Data

Operatøren Hans logger seg inn på systemet for å sjekke motordataen til Aux Engine 2. Han får meldingen 'data ikke tilgjengelig' der hvor han ellers ville funnet dataen. Hans prøver å oppdatere siden, men meldingen vedvarer. Hans vurdere det slik hen at det kan være et tilkoblingsproblem med motorsensorene. Han rapporterer problemet til maskinisten som bytter sensor.

4 Beskrivelse av den digitale tvillingen

I dette avsnittet presenteres en detaljert beskrivelse av utviklingsprosessen og de tekniske aspektene ved den digitale tvillingen av Tunglabben, utviklet ved bruk av Unity. Den digitale tvillingen er utformet for å gi studenter og ansatte muligheten til å overvåke, feilsøke og familisere seg uten å være fysisk til stede, med en sterk vektlegging på brukervennlighet og gjenkjennbarhet. Prosjektet bygger på tidligere erfaringer med Unity, og har ført til utviklingen av en rekke nøkkelfunksjoner som et interaktivt kontrollrom, visning av livedata, og tilpasningsmuligheter for brukeren.

Arbeidet med den digitale tvillingen involverte importering av nødvendige assets og utvikling av tilpassede løsninger, spesielt i forhold til modellering og scripting. Det ble lagt stor vekt på struktur og organisering av prosjektet, spesielt når det gjelder asset-håndtering og kodelesbarhet. En rekke tekniske utfordringer ble adressert underveis, blant annet relatert til systeminnstillinger og håndtering av live data fra dieselgeneratorer.

Avslutningsvis reflekteres det over bruk av Unity i prosjektet, med ettertanker om hva som kunne vært gjort annerledes og en oppsummering av den digitale tvillingens nåværende status. Den fremstår som en detaljert og funksjonell første versjon, med alle ønskede funksjoner implementert og en virtuell gjengivelse av den virkelige labben.

4.1 Planlegging av den digitale tvillingen

Vår digitale tvilling av tunglabben skal gi de ansatte mulighet til å ha oversikt og feilsøke problemer uten å nødvendigvis være tilstede, av den grunn har vi satt et fokus på at vår digitale simulator skal være gjenkjennelig replika av tunglabben som skal presentere relevant data og informasjon på en brukervennlig måte.

På grunn av et tidligere prosjekt med Unity fikk vi raskt noen ideer om hva vi ønsker fra vår digitale tvilling. Våre ønsker for tunglabben, fra før vi begynner å utvilke vårt avsluttende case i Unity er.

- Et kontrollrom som viser kritiske alarmer
- Muligheten til å se livedata og databladene til komponenter
- Gjenkjennbar modell av tunglabben
- Mulighet for brukeren å selv justere innstillinger i simuleringen, skru av og på popups osv..
- Praktiske Hot-Keys
- Start/hjem skjerm med mulighet for å endre innstillinger og informasjon/bilde

4.2 Utvikling av digitale tvillingen

Til å begynne med, fikk vi inn vår spillermodell som skal kunne navigere seg gjennom labben, for dette importerte vi asseten Starter Assets – First Person Character Controller — URP fra Unity asset store. Dette gir oss en bane og en spiller Kapsel med innebygd kamera, som kan hoppe, løpe og låser musen til kamera noe som gir vår spiller gjenkjennbart kontroll-oppsett. Elementer som ikke er relevante ble fjernet fra banen. I deres plass importerte vi en Unity pakke av tunglabben gitt til oss av faglærer OSJ. Etter av vi satt opp basisen for simulatoren la vi den ut på Github for at alle medlemmer skal kunne arbeide med simulatoren. I hovedsak så har det vært gruppemedlemmene Anders og Fredrik som har vært ansvarlige for utvikling av simulatoren, hvor Anders har hovedsakelig fokusert på designet og importering av modeller, mens Fredrik jobbet med design av modeller i tillegg til scripting av funksjoner.

I designfasen oppdaget vi at asset store hadde sine bergrensinger. Vi har blandt annet brukt Grabcad, kontaktet leverandører for modeller og funnet 3d modeller hos f.eks Wärtsilä. Filene har vært i ulike formater som ikke alltid har vært støttet av Unity. Da har to metoder vært nøkkel for å kunne implementere disse modellene. Enten ved bruk at Blender, som er et 3D modelleringsprogram. Dette gir litt frihet til å endre på mesh eller å få en modell til å bli kompitabel med unity. Den andre metoden har vært å teste ulike filformat-konverteringer for å få kompitable modeller til unity. F.eks noen filer fungerer helt fint i FBX-format, mens andre filer i dette formatet resulterte i å bli selve "Autocad" tegningen. Da var løsningen å teste andre kompitable fil-formater.

Prosessen med å lage en gjenkjennbar modell av Tunglabben begynte naturligvis med å ta bilder. Vi tok en blanding av oversiktsbilder, nærbilder og bilder av datablad som vi sorterte i teams. Oversiktsbildene ga oss referanser når vi valgte og plasserte komponenter vi importerte fra blender. Når det gjaldt databladene ønsket vi en standard som vi kunne følge, som gjorde at databildene ble enkle og lese og enkle og endre på. For det lagde vi to program i html (DatabladEditor.html og DatabladLiten.html) som er tilpasset ulike størrelser på datablad. Når det var programmert skrev vi den samme dataen fra de faktiske databladene inn, og importerte skjermbilde av databladet til Unity. Nå er databladene satt opp for at en triggerkollider på en komponent lager en popup med tilhørende datablad. Dette gjorde vi for alle komponenter vi vurderte at det var praktisk.

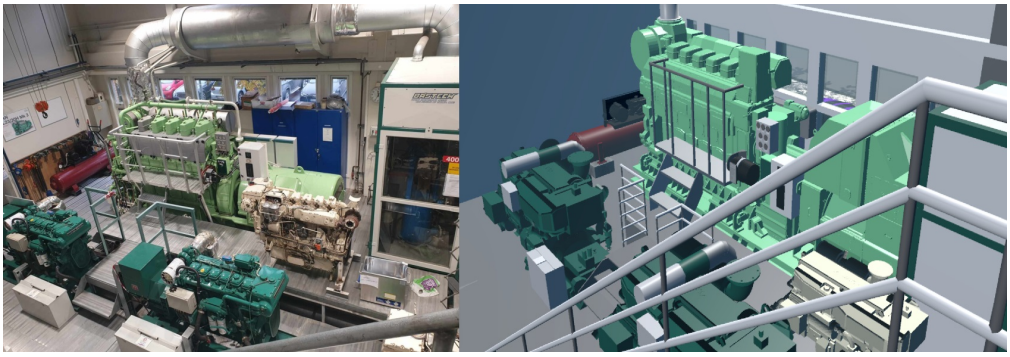

Figure 3: Sammenligner mellom virkelige og digitale labben

4.3 Struktur

I større prosjekter er det viktig med struktur, derfor begynte vi tidlig med å organisere mapper og komponenter, dette viste seg å være ekstremt praktisk etter hvert som vi importerte flere assets. Under Assets ble det dannet mapper for materiale, script, textures, komponenter og scenes. Importerte assets ble sortert under etablerte mapper med noen unntak som Starter assets, der hvor vi ble vurdert praktisk. Med jevne mellom ble mappene kontrollert for å fjerne mest mulig unødvendig materiale.

Inne i selve skriptene har vi også forsøkt å holde en standard for lesbarhet og struktur. Vi har jobbet med en filosofi hvor alle funksjoner og variabler skal beskrive hva de er eller gjør i selve navnet sitt, variabler som henger sammen skal være bygd opp på en lik måte og være cirka like i lengden. Det har oppstått tilfeller hvor vi har endt opp med å bruke variabler for noe annet enn det den opprinnelig var tenkt til, variabelnavnet blir da endret på i en senere gjennomgang. Ved de områdene hvor det kan dukke opp misforståelse om hva som skjer/skal skje, er det lagt til korte kommentarer. Dette er tiltak vi har gjort for å styrke lesbarhet og samarbeidet mellom oss på gruppen.

4.4 Tekniske utfordringer og løsninger

Våre ønsker om å ha settings og navigerbar start og pause meny har ledet til tekniske utfordringer etter hvert som vi har utviklet simulatoren videre. Hovedsakelig har utfordringene handlet om å få samarbeid mellom de tre ulike metodene brukeren kan bruke for å endre settings (fra startmenyen, pausemenyen og hotkeys). I tillegg, har det vert noen mindre utfordringer med å implantere funksjonene til tilhørende setting.

4.5 Håndtering av live data

Dataen tilhørende dieselgeneratorene er strukturelt helt like og benytter derfor samme skript, som er tenkt tilpasset alle generatorer / motorer. Skriptet ligger på et under objekt med navn lik generatorens ID, skriptet henter da underobjektets navn og bruker det for å velge hvilken liste med data vi henter. Dette er lagt opp til å skje hver gang vi går inntil diesel generatorene, det var vurdert å kjøre denne hentingen av data i update for at verdiene skal kunne oppdatere seg live, men dette ble vurdert imot fordi Unity bruker for lang tid på å koble opp.

I skriptet har vi valg å legge verdiene vi trenger i forskjellige lister, fordelen med dette er at vi raskt kan kjøre en for loop for å både hente og plassere dataen riktig, sannsynligheten for skrivefeil er også redusert ved å anvende denne metoden.

Hentingen av livedata begynner etter at scriptene kommuniserer via gamemanager og blir enige om at de hente data. Dette starter en lokal funksjon som blir gitt lister basert på komponent navnet for å skille de ulike generatorene, og kun gi relevant data om komponenten vi velger å inspisere. For hvert objekt i listen vil skripte lage en url og returner informasjon fra databasen, deretter kjøres en ny funksjon som splitter dataen og returnerer den delen vi er ute etter (ikke dato eller tid siden vi alltid har de nyeste målingene). Til slutt legger funksjonen dataen inn i vår popup med riktig variabel, for verdier som brukeren kanskje ikke forstår (eks Run = 1 og Rev-pwr = 1), kobler vi de bolske verdiene med en mer beskrivende tekst (som Running-Reverse).

4.6 Ettertanker om Unity delen av prosjektet

Når jeg skriver dette er vi nesten ferdige med prosjektet, det betyr at vi har rukket å bli litt etterpåkloke. Om vi skulle gjort Unity delen av prosjektet ville vi gjort noen ting annerledes, men så langt er vi ganske fornøyde med det vi har klart å utarbeide. En ting vi burde gjort annerledes var å velge mer seriøse eller beskrivende navn på endringene vi pushet til GitHub, dette gjelder særlig starten av prosjektet siden det er få commits fra den perioden vi faktisk husker eller forstår hva er.

En av de større tingene vi skulle ønske vi hadde gjort bedre hadde vært å sette opp Gamemanger på en annen måte. Per nå styrer Gamemanager settings slik som den burde men, den styrer også menyen og oppdaterer tekst og sånt, dette har ført til at vår Gamemanger har vert litt uryddig og trengt noen bug-fixes. Til et senere prosjekt ville vi nok antageligvis satt Gamemanger til å kun endre verdier. Deretter ville vi fått alle andre skript til å forholde seg til dens verdier, inkludert pausemeny skriptet PauseMenu.cs som ville sett radikalt annerledes ut.

En funksjon vi likte, som vi fant ut av ganske sent var OnMouseDown. Om vi hadde oppdaget denne funksjonen tidligere i prosjektet, eller om vi hadde hatt bedre tid, så kunne brukt OnMouseDown til at brukeren må trykke på komponenten for å se databladet eller livedataen. Fordelen med dette hadde vært at brukeren kan lettere unngå å hente livedata når brukeren nærmer seg komponenten, selv om livedata er aktivert i settings. Per nå fungerer komponentene slik at, hvis du går innenfor triggerområdet og du har aktivert livedata, vil skript forsøke å koble til databasen uavhengig om du er koblet til riktig nett eller ikke. Dette tilkoblingsforsøket tar alltid en del tid så det hadde vært svært praktisk å bruke OnMouseDown for å fjerne u-ønskede forsøk på tilkobling.

4.7 Oppsummering av digital tvilling

Vår Digitale tvilling er nå et godt detaljert og funksjonelt førsteukast. Alt av funksjoner vi kunne tenke oss når vi begynte på prosjektet er nå, i en eller annen form lagt til og fungerer. Tvillingen er en detaljert og gjenkjennbar replika av den virkelige labben med tydelige og riktige datablad på relevante komponenter. All tilgjengelig livedata er blir nå vist i Unity på riktig tilhørende komponent når brukeren ønsker å se, ved å aktivere det i settings. Startsiden har en navigerbar meny som inneholder informasjon, justerbare innstillinger og muligheten å starte simulatoren via scenemanager. Vi har så langt fått til det vi ønsker og er fornøyde med vår egeninnsats i Unity-delen av prosjektoppgaven.

5 Brukertilbakemeldingsdrøfting for HMI

I prosessen med kontinuerlig forbedring av Tunglabben eksterne brukergrensesnitt, er brukertilbakemelding en viktig komponent. Det overordnede inntrykket fra sluttbrukere har vært positivt, spesielt med tanke på det intuitive designet og den klare visualiseringen av individuelle skip i det tekniske oversiktsbildet. En tilbakemelding vi har fått, og som vil forbedres, er muligheten for å kunne sammenligne to skip direkte i den tekniske oversikten. Dette forslaget underbygger behovet for en dypere analytisk tilnærming hvor operatører kan utføre side-ved-side analyser for å identifisere ytelsesforskjeller, optimalisere vedlikeholdsrutiner og forbedre beslutningsprosesser. Som en respons på denne tilbakemeldingen, vil vi se på muligheter med å integrere en sammenligningsmulighet som lar brukerne se informasjon om to skip mot hverandre. Dette vil øke verdien av den tekniske innsikten applikasjonen gir, og styrke systemets brukervennlighet, driftsoptimaliseringevne og gi bedre grunnlag for langsiktig flåtestyring.

6 Utbedringsforslag

I dette avsnittet utforskes arbeidsmetoden og de tekniske aspektene ved utviklingen av et prosjekt som involverer Ignition Perspective HMI i Tunglabben. Vi har tilnærmet oss prosjektet med stor grad av selvstendighet, hvor oppgaver er blitt fordelt og fulgt opp individuelt, drevet av en klar visjon om prosjektets mål. Videre diskuteres potensielle ideer for fremtidige forbedringer og integrasjoner i arbeidsmetodikk og i det tekniske. Det vektlegges hvordan en dypere samhandling mellom disse systemene kunne realisere nye funksjoner som inspeksjon av fartøy via en digital tvilling. Gruppen reflekterer også over utfordringer knyttet til databasen og live data, samt mulige forbedringer for å forsterke programmenes effektivitet og brukervennlighet. Denne seksjonen gir en innsikt i prosjektets nåværende status, de lærdommene vi har trukket, og mulige retninger for videre utvikling.

6.1 Arbeidsmetode

Arbeidsmetoden vår har så langy vært veldig selvstendig, vi har fordelt oppgaver og gjennom prosjektet, hovedsakelig forholdt oss til disse oppgavene. Dette har vi klart å gjennomføre ved å vi har hatt en klar visjon om hva vi ønsker å lage/bidra med. Før vi begynner på utbedring må vi bli enige om en videre visjon til hva vi ønsker å gjennomføre i prosjektet fremover, noen av de ideene er beskrevet i seksjon 6.2. I tillegg til å legge en ny visjon burde vi involvere hverandre mer i hva vi har laget og kommer til å lage fremover, dette vil kunne gjøre at oss på gruppen vil få en bedre forståelse for hva som burde gjøres i tillegg til at vi kan kontinuerlig kvalitetssjekke hverandres arbeid.

6.2 Ideer til design

Om vi hadde fortsatt med prosjektet kunne det vært praktisk å utvikle et større samarbeid mellom ignition og Unity delen av oppgaven, en tenkt funksjon hadde vært å kunne ha tilgang fra det ene programmet. Hvis man kunne gått inn i ignition og velger en vessel, så hadde det vært ide å kunne trykke på noe sånt som en inspect knapp som starter en Unity tvilling av den valgte båten. En annen måte å gjennomføre noe lignende hadde vært å kunne klikket på en dropdown boks, mens man fortsatt er på startiden for å inspisere tvillingen til den valgte båten. På den måten kunne man lest data fra ignition samtidig som man navigerer seg gjennom båten i Unity.

Per nå har vi fuelscore kalkulert ved at vi ser på live verdier sitt avvik fra de nominelle verdier til utstyret. Deretter er kun en metode vi bruker for som et konseptbevis, hvis vi hadde hatt drivstofforbruksdata hadde vi benyttet disse isteden. Dette er kun en placeholder metode, ideen bak fuelscore er å bergene effektiviteten til fartøyets bensinforbruk, ved å hente verdier som blant annet båtens tyngde og bensinforbruk per km (informasjon vi ikke har tilgang til). Ved å beregne fuel score kan vi gi kapteinen forslag om optimal hastighet, eller bruke det til å lete etter slitne/ødelagte komponenter ombord på båten som kan bidra til lavere fuelscore enn forventet.

Time since maintenance tenker vi oss er koblet opp mot en klokke som teller antall dager gått. Vi ser for oss at når båten har gjennomført en vedlikeholdskontroll så skal klokken resettes. Time to dock maintenance viser tiden til det skal gjennomføres en vedlikeholdskontroll. Time to dock maintenance skal være satt opp ved at vi gir båten en passende tid mellom hver kontroll, så subtraherer tallet fra time since maintenance for å gi en tid (i dager) til neste kontroll. Vi ønsker også å varsle kapteinen noen dager før tiden går ut (melding eller popup) i tillegg vil kapteinen også bli varslet når Time to dock maintenance tiden har gått ut.

Ved feilsøking av generatorer så hadde det vært praktisk å vise temperaturen i en popup og koblet den verdien opp mot AH/AL logikk, slik at brukeren raskere kan identifisere mulige feil med generatoren. Om vi hadde hatt et veldig sterkt ønske om å vise høy og lav alarm så kunne vi ha brukt dataen fra den forrige databasen slik som var gjort i den forrige oppgaven. Dette har vi vurdert imot grunnlaget, at det tar tid å implementere, at det tilhører en database som ikke er del av denne prosjektoppgaven og at de ikke viser varmen, men kun AH og AHH (som vi kunne gjort om til AL og AH med noen endringer).

6.3 Database

En fiks som vil styrke programmene vårt er å ha en database med mer sikker tilkobling, hittil i prosjektet har vi hatt en del problemer med tilkoblingen til databasen hovedsakelig i Ignition. Blant annet inne på Technical siden så har vi hatt et problem at vi plutselig mister all live-dataen som ble vist og vi må vente til programmet finner all dataen på nytt. Dette svekker flyten i programmet og samtidig senker brukervennligheten.

Å kunne hente mer data fra databasen hadde også vært et pluss, verdier som trip, voltage-under/over level og temperatur. Er bidrag som kan hjelpe brukeren identifisere eventuelle problemer med utsyret sitt. Om vi får mer data som vi kan hente, og ønsker at den dataen skal vises i Unity burde vi utvilke en funksjon som tillater brukeren å se all dataen, fortsatt på en ryddig måte. En tenkt løsning på det å ha den mest sentrale dataen på en side, tre sider for data om L1 L2 og L3 og den siste informasjonen på en femte popup side.

Enda ett ønske vi har til Databasen er en raskere måte å koble opp til i unity. Per dags dato så finner vi livedataen ved at vi kjører en url som vi bruker til å finne dataen, dette er en operasjon som programmet foretar seg flere ganger, via en for-loop for å fylle ut tabellene sine. Resultatet av dette er at hele Unity fryser en liten stund før den har hentet all dataen sin. En mulig løsning på dette er å utvikle et view i databasen som viser alle verdier vi ønsker, det betyr at vi kun trenger å søke med url en gang, som vil redusere tiden skjermen er fryst i Unity.

7 Konklusjon

Dette har vært en lærerikt og morsomt avsluttende caseoppgave. Vi har fått muligheten til å bygge på det vi har lært i tidligere caseoppgaver og fag, og brukt den kunnskapen til å utvikle program som er detaljerte og foretar seg nyttige funksjoner. Ved å ha tatt i bruk av use-caser, use-case-slices og historier har vi fått en innsikt i hva det er praktisk å vise frem og hvordan vi burde vise det. Vi har brukt vår kunnskap og evne for å komme på kreative og intuitive løsninger som styrke brukervennligheten og har satt et fokus på ha et ryddig og pent design.

I Ignition har vi utviklet et program som gir god oversikt over en rekke båter med den tilgjengelige dataen vi har fått tildelt. HMI designet har vært presentert for kunden og vi har fått positive tilbakemeldinger. Forslag som de har kommet med har vi tatt til vurdering og gjennomført i Ignition etter beste evne. Forslag til videre utvikling av HMI ville vært å gjøre applikasjonen mer vennlig for å sende informasjon mellom views. Vi har implementert noen slike funksjoner, men ved å opprette kommunikasjon via message handlers tidlig i arbeidet, kunne vi utviklet et design som kan tilpasses brukeren i større grad. Dersom dette prosjektet skulle videreutvikles ville fokuset vært mer fleksibilitet for bruker til å benytte seg av. En slik applikasjon kunne også vært tilpasset f.eks mobiltelefoner også.

Å bygge en simulator i Unity har vert en morsom erfaring som inspirerer oss til å utvikle flere prosjekter i Unity til en senere anledning. Vi har utfordret oss selv med å bygge en detaljert labb og avanserte funksjoner, vi også har fått til alt vi ønsket oss i starten av prosjektet. Underveis så har vi oppdaget nye elementer, funksjoner og løsninger som gjerne også skulle ha gjennomført, som vi dessverre ikke har fått tid til. Vi har lært av feil som vi har oppdaget underveis i prosessen, og gjort oss noen tanker om hvordan vi hadde jobbet og gjort annerledes, om vi hadde fått et nytt prosjekt.

Til slutt har vi reflektert og kommet på forslag for utbedring av prosjektet. Vi har gjort oss noen tanker om funksjoner og endringer som ville gjort prosjektet bedre. I tillegg har vi reflektert over vår arbeidsmetodikk, og hvilke endringer vi burde gjøre for å oppnå et sterkere samarbeid fremover.

8 Litteraturliste

[1] Alan Cooper. *The Inmates Are Running the Asylum*, volume 2004. Paul Boger, 2004.

[2] Jesse James Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond*, volume 2011. New Riders, 2010.

[3] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, volume 2009. John Wait, 2005.