Bachelorprosjekt

Våren 2022

av Odd Martin Kveseth

Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR.
68
TILGJENGELIGHET
Offentlig

Telefon: 22 45 32 00

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
Ytterligere funksjonalitet til ForzaSys' pmSys	25.05.2022
	ANTALL SIDER / BILAG
	89
PROSJEKTDELTAKERE	INTERN VEILEDER
Odd Martin Kveseth (s305057)	Pål Halvorsen palh@oslomet.no

OPPDRAGSGIVER	KONTAKTPERSON
ForzaSys	Pål Halvorsen palh@oslomet.no Tomas Kupka tomas@forzasys.com

SAMMENDRAG

Det skal lages en webapplikasjon der det skal fremstilles funksjonalitet som kan brukes i ForzaSys sitt statistikk monitoreringssystem, pmSys. Dette innebærer å lage graf diagrammer og tabeller av statistikk om manglende rapportering som skjer av deres brukere.

3 STIKKORD
Webapplikasjon
Grafisk plotting av data
Brukerundersøkelse

Forord

Jeg er en dataingeniørstudent ved OsloMet som har fått muligheten til å jobbe sammen med firmaet ForzaSys for å kunne lage ytterligere funksjonaliteter til deres system pmSys.

Funksjonaliteten jeg har jobbet med er forskjellige måter å kunne visualisere statistikk om brukernes evne til å rapportere inn all dataen de skal. Når man jobber med statistikk er det viktig å ha et komplett datasett, derfor kan en oversikt som viser om det er en mangel av data være nyttig.

Det var derfor utviklet en webapplikasjon som skulle bli brukt for å fremstille løsningen min, og for å kunne ha den testet av arbeidsgiver og deres brukere.

Jeg vil også her takke min veileder og arbeidsgiver i dette prosjektet, Pål Halvorsen. Han har hjulpet meg gjennom hele denne prosessen, vært vennlig og imøtekommende uansett hva som skjedde. Det gikk mye opp og ned, og mest ned, men han var alltid der for å hjelpe meg opp igjen.

Leserveiledning

Dette er en rapport for bacheloroppgaven ved OsloMet som en del av dataingeniørstudiet, skrevet våren 2022, i samarbeid med ForzaSys.

Rapporten har som mål å beskrive hele prosessen rundt samarbeidet med ForzaSys. Den er delt opp i 8 deler:

Prosjekt introduksjon

Denne delen er en introduksjon av oppgaven som skal løses, målet for oppgaven, og hvem som har vært en del av dette prosjektet.

Prosessdokumentasjon

Her forklares prosjektets prosess. Det vil bli gjort rede for hvilke arbeidsmetoder som ble tatt i bruk og reflektert over disse valgene.

Teknologier brukt i løsning og utvikling

Her vil det vises en oversikt over alle språk, rammeverk og verktøy som var utnyttet i utviklingen av dette prosjekt og hvorfor disse ble brukt.

Produktdokumentasjon

Her blir det gått i detalj hvordan løsningen som er utviklet er bygd opp og en gjennomgang av all funksjonaliteten som er bygd.

Testing

I denne delen blir det gått over måten løsningen er testet. Dette innebærer testing utført av utvikleren, arbeidsgiver og sluttbrukere.

Oppdragsgivers vurdering

Dette er en kort del som beskriver oppdragsgiver sin vurdering av løsningen og om det er noe de kan bruke videre.

Refleksjoner

Dette kapittelet diskuterer sluttresultatet og hva som er blitt lært gjennom prosjektet.

Oppsummering

Denne delen er en kort helhetsvurdering av prosjektet, hva som ble gjort, og hva som er blitt lært i løpet av prosessen.

Innholdsfortegnelse

1 Prosjekt introduksjon

1.1 Gruppemedlemmer

Det er kun 1 person i denne gruppen og det er meg, Odd Martin Kveseth. Jeg er en student i studiet dataingeniør på OsloMet. Jeg fikk tillatelse til å jobbe alene på grunn av omstendigheter rundt tidspunkter jeg har endt opp med å ta diverse fag som førte til at jeg ikke hadde mulighet til å gå i gruppe med noen andre ved prosjektets oppstart.

1.2 Veileder

Min veileder er Pål Halvorsen. Han er en professor ved institutt for informatikk ved OsloMet, og også administrerende direktør for ForzaSys, firmaet jeg skal jobbe med.

1.3 Om oppdragsgiver

ForzaSys er en del av Simula Research Laboratory, som er et selskap som driver med forskning innen programvare- og kommunikasjonsteknologi, og ble opprettet i 2014. ForzaSys jobber med utvikling og forskning innen video streaming og data analysering innen idrett, som i produktene deres “Forzify”, et direkte og on-demand video streaming system, og “pmSys”, et dataanalysering system.

1.3.1 Om pmSys

pmSys står for “Player Monitoring System” og er et monitoreringssystem for idrettsutøvere og er allerede i bruk av mange idrettslag i blant annet Toppserien for kvinner. Brukerne skal gjennom en mobilapp registrere informasjon rundt helsen sin på daglig basis, etter trening, og etter kamp.

All denne informasjonen vil så bli mulig for treneren å kunne se på nettsiden til pmSys også kalt treningsportalen. Her vil trenerne kunne se oversikt av dataen sendt inn av lagene de har tilgang til. Man kan velge å se snitt tallene for hele laget, eller gå i detalj på en spiller. Trenerne vil også kunne sende ut varsler til laget eller enkeltspillere om for eksempel påminnelser eller oppfølging av dataen de har sendt.

1.4 Oppgavebeskrivelse

For at trenere skal kunne få mest mulig ut av slik monitorering av spillere, er det viktig å få en full oversikt over HELE laget. Alle bør derfor rapportere inn i systemet, og en viktig funksjonalitet kan derfor være å kunne få en god og enkel oversikt over hvem som til enhver tid har rapportert, og evt. kunne sende påminnelser.

ForzaSys ønsket derfor mer funksjonalitet til treningsportalen for å gi mer statistikk enn det allerede er tilgjengelig. PmSys foreløpig viser mange forskjellige statistikker hentet inn fra dataen sendt inn fra spillerne, men en av statistikken som mangler er for eksempel statistikk om statistikken, som hvor mange sender inn rapporter daglig.

1.5 Mål

Målet for oppgaven er å kunne lage en web applikasjon hvor jeg skal kunne presentere løsninger til de problemene jeg får fra ForzaSys. Denne løsningen skal kunne legges ut på nett for å dele den med brukerne deres og deretter bli bedømt av dem for å kunne fastslå hvilken løsning som er mest hensiktsmessig. Og ut i fra dette kunne dele noe med ForzaSys som de så kan legge til på plattformen deres.

2 Prosessdokumentasjon

I dette kapittelet skal jeg gå over hvordan prosjektet ble utført. Om prosjekt metodikkene som ble tatt i bruk, hvilke utfordringer som jeg støtte på, og hvordan det gikk. Hvordan jeg har forholdt meg til arbeidsgiver, og møtene som er holdt. Jeg har jobbet alene om denne oppgaven, i motsetning til vanlige bachelorer som er i grupper, så jeg vil også diskutere litt hvordan det har påvirket prosessen.

Det har gått mye galt når det kommer til denne delen av prosjektet. Å jobbe alene har vært en utfordring på den måten at det er jeg som skal gjøre alt, og bare meg selv til å følge opp at det blir gjort. På grunn av dette var det uklart for meg i starten av prosjektet hvor konkret og detaljert en prosessmodell måtte være for å jobbe alene, og dette endte opp med å være en ganske avgjørende feil for prosjektet.

Dette prosjektet startet den 1 september 2021 og varte til 25 mai. Det var meningen at dette prosjektet skulle vært ferdig i januar, men på grunn av dårlig planlegging fra min side førte det til at jeg ikke ble ferdig til januar og prosjektet ble forlenget til mai.

2.1 Oppstart

Dette prosjektet ble satt igang ganske brått. Jeg er en student som har tatt opp noen fag i slutten av studiet og fikk en anbefaling av studieadministrasjonen om å ta bachelor oppgaven på høstsemesteret og de satte opp en plan for mine 3 siste semestre. Når det tredje semesteret kom prøvde jeg å melde meg opp til bacheloren, men det funket ikke. Jeg får så beskjed at jeg skulle ha meldt meg på et halvt år i forveien, men på grunn av min litt spesielle situasjon var ikke dette informasjon som hadde nådd meg eller tydeliggjort under semesterregistrering, selv om jeg hadde vært i kommunikasjon med studieadministrasjonen hvert semester. De ga meg så tillatelse om å prøve å starte likevel så lenge jeg kunne finne meg et prosjekt å jobbe med. Jeg fikk så tips om å kontakte Pål Halvorsen. Han hørte ut hva jeg hadde kunnskap om og ga meg et prosjekt jeg kunne jobbe med på kort tid og tok vare på meg hele veien etter det.

2.2 Kommunikasjon

Siden det var kun meg og mine resultater som ville være kommunikasjonen mellom oss avgjorde vi at det ikke var noen stor grunn til å starte en Slack/Discord server for dette, men at vi tar det over mail. I en større gruppe ville det nok vært fornuftig med bruk av slike

programmer for å kunne også diskutere mellom seg i gruppen om det skulle være noen utfordringer man ville diskutere med en gang. For meg som jobber alene var mail godt nok for å kunne rapportere fremgangen min, eller sende en mail til Pål og Tomas om jeg hadde spørsmål rundt krav eller oppklaringer av oppgaven. Disse hendelsene er ikke veldig hastesaker som ville være der live-forums/chats som Slack kan være mer nyttig.

Utenom mail ble det også holdt møter gjennom Google Meet. I starten ble det avtalt på mail etter at en milepæl var møtt om å ta et nytt møte. Mot slutten av prosjektet spurte jeg om å kunne ha litt mer regelmessige møter for å prøve å øke min egen effektivitet ved å måtte vise fremgangen min oftere.

2.3 Prosjektorganisering

Jeg jobber alene, så ikke mye å nevne om dette, men jeg er min egen prosjektleder. Det er jeg som er ansvarlig for at prosjektet skal kunne ha den fremgangen som trengs for å gjennomføre prosjektet på en effektiv og god måte (Vaagaasar & Skyttermoen, 2017, s.49). Det er jeg som skal utføre alt som trengs for å få gjennomført dette prosjektet Jeg skal forholde meg til Pål Halvorsen hos ForzaSys som arbeidsgiveren og prosjekteier. Det er han som formidler interessene til bedriften, følger opp fremgangen til prosjektet og kan gi tilgang til nødvendige ressurser (Vaagaasar & Skyttermoen, 2017, s.18).

2.4 Arbeidsmetodikk

Her skal jeg gå over hva slags arbeidsmetodikk som ble vurdert og brukt i dette prosjektet. Siden jeg er alene var det vanskelig å komme opp med noe som vil kunne bli utnyttet på en effektiv måte. Jeg har kunnskap om de forskjellige metodikkene siden dette ble undervist om i fag som Systemutvikling i løpet av studiet, men jeg har kun lært dem for bruk i sammenheng med større grupper. Jeg slet derfor litt med å finne ut hva som kunne passe for meg i dette prosjektet. En ting var ganske åpenbart og det var at dette prosjektet måtte utføres på en agil måte. Det måtte kunne startes opp fort og endringer ville skje underveis på grunn av uklarheten som var rundt et prosjekt som ble satt opp på kun et par dager.

2.4.1 Agil arbeidsmetode

Agil arbeidsmetodikk har fokus på å kunne utvikle og presentere flere dellevereanser av produktet til oppdragsgiver over en rekke utviklingssløyfer (også kalt sprinter) for å kunne regelmessige gi tilbakemeldinger om styrker og svakheter (Vaagaasar & Skyttermoen, 2017, s.120). I dette prosjektet var ikke alle oppgavene som skulle utføres fastsatt med en gang og

det var i stedet viktig å starte der man kunne og iterere på det mens flere ting blir oppklart. Så ved å jobbe agilt ville jeg kunne starte med de oppgavene som var avgjort og kunne få tilbakemelding fortløpende på disse. For hver sprint ville jeg kunne vise fremgangen min og arbeidsgiver gi tilbakemelding. Arbeidsgiver vil da kunne foreslå hva som kan endres, ikke bare ut i fra valgene som er gjort under utviklingen så langt, men også ettersom oppgaven blir mer klar og utfyllende over tid.

2.4.1.1 Refleksjoner rundt andre gennomføringsstragegier

Det er flere måter å styre et prosjekt på, men for dette prosjektet var agil det mest hensiktsmessige. Det er to andre vanlige brukte strategier: Sekvensiell og Milepælebasert.

I en sekvensiell strategi vil prosjektets oppstartsfase innebære å fastsette de mest vesentlige rammebetingelsene og spesifikasjonene knyttet til produktet. Gjennom disse premissene skal det avgjøres hvilke aktiviteter som skal gjennomføres og rekkefølgen dems. Prosjektet forventes å kunne utføres i samsvar med planen som blir avgjort her (Vaagaasar & Skyttermoen, 2017, s.40). På grunn av all uforutsigbarhet i dette prosjektet, med tanke på både tidsbruken for hver oppgave og hvilke oppgaver som vil bli relevante, vil det ikke være mulig å kunne lage noen plan i oppstartsfasen om hva som skal skje over de neste månedene.

Milepæle strategi har fellestrekk med sekvensiell, men i stede for å lage en plan for en bestemt rekkefølge utførelsen skal følge, gir den mulighet for å kunne jobbe med forskjellige oppgaver parallelt og heller vite om de viktigste delene som prosjektet må nå før prosjektet starter for å kunne nå ønsket leveranser i løpet av prosjektperioden (Vaagaasar & Skyttermoen, 2017, s.41). Av samme grunn som sekvensiell er det ikke veldig hensiktsmessig siden ikke alle kravene er oppklart i begynnelsen av prosjektet. I tillegg er en av fordelene med milepæle at rekkefølgen ikke er viktig, og man kan jobbe med ting i parallell, men siden jeg er bare en på denne gruppen er det ikke stor grunn til å jobbe på diverse deler av prosjektet parallelt, men heller agilt hvor en del kan fort bli gjennomført med tilbakemeldinger underveis før man trenger å starte på neste.

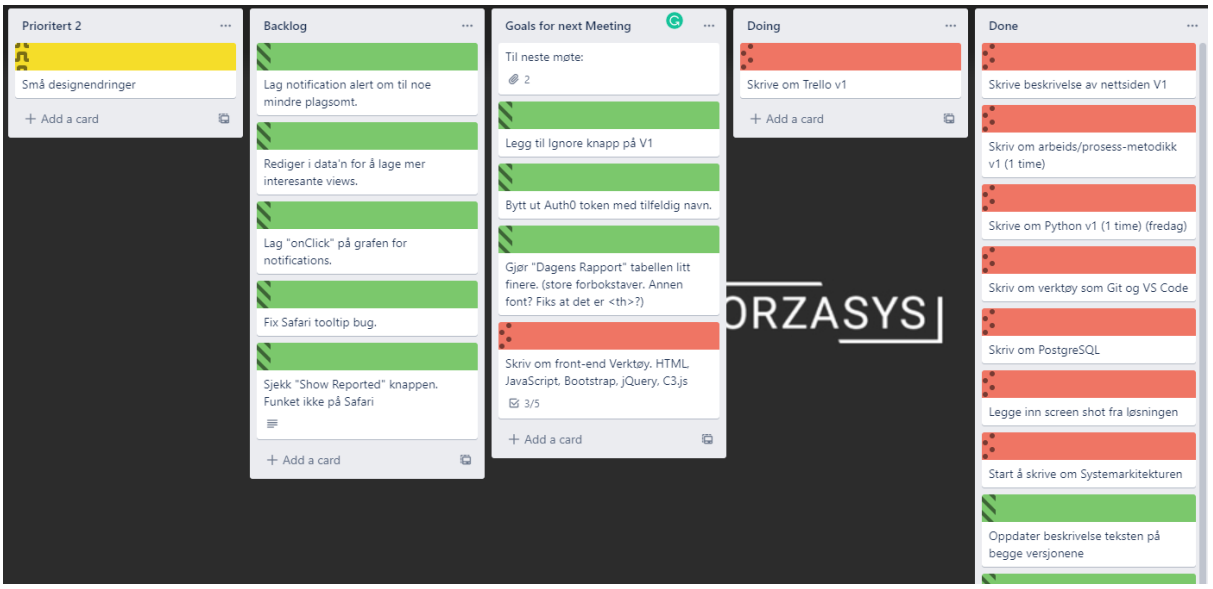
2.4.2 Kanban

Kanban var et metodikk verktøy som jeg ble utnytte i dette prosjektet som en måte å skape en oversikt over alle oppgavene som skal utføres. “Hovedtanken er at man setter i gang bare det arbeidet innenfor et område som er nødvendig for å komme videre. Igangsatt

arbeid innenfor et område skal altså begrenses. Det handler om å balansere behov og kapasitet…" (Vaagaasar & Skyttermoen, 2017, s.123). Dette verktøyet kan være nyttig selv med bare en bruker for den vil fortsatt gi en visualisering av alt arbeidet som må gjøres, de oppgavene som er viktigst å få gjennomført, og hva som kan være realistisk å få gjennomført.

2.4.2.1 Trello

Trello er en webapplikasjon som hjelper å sette opp en Kanban-tavle. Her kan man lage lapper og kolonner, dra lappene mellom kolonner og gi de forskjellige lappene distinkte forskjeller for å kunne identifisere hva slags jobb som skal bli gjort. Figur 1 viser et bilde av Kanban-tavlen brukt i dette prosjektet.



Figur 1: Kanban tavle laget i Trello

Kolonnene beskriver i hvilken tilstand oppgaven skrevet på lappene i kolonnen er i. Det er 5 kolonner i denne tavellen merket "Prioritert 2", "Backlog", "Goals for next Meeting", "Doing" og "Done". Om noe er i Prioritert 2 betyr det at det er små endringer som ikke er et krav fra arbeidsgiver, men heller noe jeg selv mener vil være en forbedring som burde ses på om det er tid. Backlog er oppgaver som må gjøres, her vil det legges inn fortløpende når nye ting som enten må legges til eller fikses blir oppdaget. "Goals for next Meeting" er ganske selv beskrivende, for hvert møte ble det foreslått hva som kunne lages eller forberedes til neste møte. Disse oppgavene blir lagt inn her for å fremhever prioriteringsnivået dems. "Doing" og "Done" er også nøyaktig det, hva jeg holder på med, og hva som er gjort.

Jeg har også fargekodet lappene for å kunne se hva jeg har igjen for de forskjellige delene av prosjektet. Grønn betyr en kode oppgave, rød betyr en skriveoppgave, og gul er for de mindre prioriterte. Dette er for å kunne skille mellom de grønne kode oppgavene som er mer krav fra oppdragsgiver, mens de røde skrive oppgavene er mer personlige mål for rapporten som skal skrives. Dette hjelper også med å kunne visualisere balansen av hva som blir gjort om jeg har mye igjen å skrive eller om jeg burde sette meg ned å kode.

2.4.3 Scrum

En av metodene innenfor agil gjennomføringsstrategi er Scrum. Siden jeg kun er en person i dette teamet er det noen deler av Scrum som jeg ikke kan ta fordeler av, men det er fortsatt deler av Scrum som kan læres av og utnyttes i dette prosjektet. Scrum beskriver en metodikk med en bestemt fremgangsmåte og er gjenkjent ved å ha 4 hoveddeler: Produktkø, Sprintkø, Sprint, og Fungerende bestanddeler (Vaagaasar & Skyttermoen, 2017, s.121). For min del i dette prosjektet har jeg ikke muligheten til å gjøre “Daglig Scrummøter”, eller ha en Scrum-master for det krever et større team som jobber tett sammen om den samme oppgaven. Det jeg kan bruke fra denne metodikken er i stedet produktkø for alt som må bli gjort. Sprintkø for hva som må bli prioritert for å kunne leveres neste møte med arbeidsgiver. Og hver sprint vil være definert som en uke, men uten daglige stand up møter. Denne metodikken er også reflektert i Kanban-tavlen.

2.5 Utviklingsprosessen

Siden jeg kun er en person på prosjektet har jeg måtte gjøre alt, så det er ingen avansert fordeling av oppgaver. Det er jeg selv som må stå for alle delene, frontend, backend, og håndtering av serveren. Prosessen har derfor bestått av en syklus som starter med et møte med arbeidsgiver hvor vi går over hvilken funksjonalitet som burde bli fokusert på. Jeg vil så jobbe med å få fullført det som ble diskutert. Jeg vil deretter sende en mail med fremgangen min, og det blir avtalt et nytt møte. På slutten av prosjektet ble det også innført ukentlige møter for en mer kontinuerlig oppfølging av produktet.

2.6 Testprosessen

Testing av løsningen er en essensiell del av prosjektet. En viktig del av oppgaven er å lage funksjonalitet som er forståelig og oversiktlig for brukeren. Så gjennom hele prosessen må det alltid utføres tester av løsningen for å kunne avgjøre om det er implementert noe som

har verdi. Det er også derfor det er laget flere versjoner av siden for å kunne gjøre testing på en måte hvor man kan sammenligne hva som føles best opp mot hverandre.

Testingen av siden starter naturlig nok med meg under utviklingen. Sjekke hva som er fornuftig å bruke i rammeverkene jeg bruker og prøve noen forskjellige versjoner. Gjennom min testing gjør jeg meg noen refleksjoner rundt det jeg har laget som tas med videre til møtet med arbeidsgiver.

Med serveren jeg hadde fått utdelt fra FrozaSys kunne jeg laste opp løsningen min der for at andre kunne også ta del i testingen. Jeg oppdaterte denne serveren når jeg hadde ny funksjonalitet eller gjort noen oppdateringer slik at ForzaSys kunne teste de nye versjonene og komme med feedback under det neste møte.

Det ble også utført en spørreundersøkelse om løsningen hvor jeg kunne også få tilbakemeldinger av eksterne kilder for deres meninger etter å ha blitt presentert løsningen.

Med tanke på debug testing så er dette noe jeg gjør selv fortløpende. Det er små isolerte funksjoner som bygges av gangen så denne typen testing av siden er ganske forutsigbar når man jobber alene. Siden jeg vil kjenne hele koden inn og ut er det litt enklere å kunne se over egen kode og justere på det som trengs, enn å for eksempel hvis man deler et team opp i frontend og backend. En frontend utvikler av programmet som får feilmeldinger fra backend siden vil ikke kunne like enkelt sette seg inn i all koden for å kunne rette opp feilen. Debugging er sett på nærmere i slutten av produktdokumentasjon delen.

2.7 Refleksjoner og Realitet

Her skal jeg gå over alt som gikk galt, for det var mye som gikk galt, og hva jeg kunne eventuelt ha gjort annerledes for å forhindre dette. Jeg er i utgangspunktet en som har dårlig selvdisiplin. Som nevnt tidligere er jeg en som har tatt opp fag tidligere og grunnen for dette er ofte at jeg ikke klarer å nå tidsfrister som er satt i de forskjellige emnene. I tillegg hadde jeg vært stresset i flere uker pga treg kommunikasjon fra studieadministrasjonen om hva som kom til å skje med det kommende semesteret. Så selv om jeg var motivert til å jobbe med et firma som ForzaSys, for sports statistikk er noe jeg er ganske interessert i, så var jeg tom for energi når alt var endelig lagt til rette. Jeg kom derfor veldig seint i gang med oppgaven som allerede hadde forsinket oppstart av å hoppe over et halvt år som brukes for å finne et prosjekt og planlegging. Men, nok om mine dårlige unnskyldninger og videre med hva som kunne vært gjort annerledes.

2.7.1 Planlegging

Planlegging er ikke noe som er skrevet om i prosessdokumentasjonen for det var ingen planlegging som ble gjort. Når prosjektet startet var tankene jeg hadde at jeg lå allerede bak skjema, og jeg måtte bare starte å jobbe med oppgavene jeg får tildelt av arbeidsgiver og ta det derfra. Jeg er kun en person i gruppa, så det eneste jeg er avhengig av er meg selv, så hvorfor planlegge hva jeg skal gjøre når jeg brude allerede vite hva som burde bli gjort?

Jeg har skrevet en del om agil gjennomføringsstrategi for jeg mener at dette er måten som det ideelt skulle vært utført, og noe som jeg hadde i tankene når prosjektet startet for vi er lært opp i studiet om å vurdere dette. Mye av dette ble dessverre ikke implementert før senere i prosjektet etter at mye tid hadde allerede blitt kastet bort.

Jeg innser nå at selv om jeg lå bak på fra starten av skulle jeg fortsatt satt av tid til å sette opp en plan for hvordan prosjektet skulle utføres. Om jeg hadde lagd en god, detaljert plan for hvordan utviklingsprosessen skulle blitt gjennomført fra starten av og tatt i bruk verktøyene som Kanban med en gang ville jeg kanskje ha laget en god struktur som jeg kunne forholdt meg til hele veien. Om jeg hadde lagd meg en slik plan ville jeg nok ha vært mer komfortabel med å foreslå mer regelmessig oppfølging med arbeidsgiver tidligere i prosjektet for å kunne sette igang en mer iterativ prosess med korte sprinter og kanskje kunne lage momentum i oppgaven allerede da. Siden jeg var i en litt spesiell situasjon, jeg var treg med å organisere det første møte med arbeidsgiver, og jeg er en innadvendt og sjenert type hadde jeg laget meg et dårlig førsteinntrykk som gjorde at jeg ikke var komfortabel med å lage noen som helst krav om hvor mye tid jeg ville ha med dem. Jeg tenkte jeg skulle ikke være til bry og ta ting over mail siden det er en kommunikasjonsmetode som ikke er like pressende. Jeg har nå erfart i etterkant at Pål Halvorsen er en ekstremt imøtekommende og hyggelig person som gjorde alt han kunne for at jeg skulle kunne fullføre prosjektet. Hadde jeg vært mer forberedt i startfasen av dette prosjektet ved å lage en plan over gjennomføringsstrategien ville det mest sannsynlig gått mye bedre.

2.7.2 Personlig innsats

Jeg klarte ikke å legge inn de timene som trengtes for å kunne fullføre alle oppgavene som ble diskutert i begynnelsen av prosjektet. Fra oktober til desember hadde jeg kun fått opp det mest grunnleggende som var spurt om, og design av siden var nesten uberørt, med en mange default innstillinger. Og rapporten var nesten ikke startet på å skrive.

I februar når prosjektet ble startet på igjen, etter at det ikke hadde gått som det skulle til januar, foreslo jeg til Pål om å ha mer regelmessige møter. Vi prøvde å diskutere en del om hva vi kunne gjøre for å øke effektiviteten min, men jeg klarte aldri å forbedre meg. Jeg trodde når jeg foreslo flere møter at jeg ville presse meg selv til til å vise mer resultater, men i stedet bare fortsatte jeg og skuffe dem om og om igjen.

Det er åpenbart noe jeg må jobbe med selv som person, og jeg prøvde å innføre en mer agil/scrum struktur mot slutten av prosjektet for å se om dette ville hjelpe meg. Dette hadde ikke så stor betydning som jeg håpet det skulle ha. Jeg klarte å sette opp Kanban, og tenkte at det så veldig nyttig ut, men jeg hadde kanskje mistet for mye motivasjon innen da av å være en skuffelse så mange ganger allerede til å kunne ta det i bruk.

En ting jeg hadde suksess med var å besøke broren min for å kunne ha et mer “jobb kontor” hvor jeg ikke blir like lett distrahert som jeg er hjemme. Det kan kanskje nevnes her om Covid, og hjemmekontor, osv, men det hadde nok ingen påvirkning på dette prosjektet siden jeg jobber alene. Jeg hadde ikke noen stor grunn til å møte opp på skolen med daglig for å jobbe sammen. Å jobbe i en gruppe ville kanskje ha hatt noe å si på effektiviteten min, kanskje om det var denne “Scrum-masteren” på “Daily Standup” som ville holde meg i gang, men jeg hadde ukentlige møter med Pål for å prøve å gjenskape dette, og det ikke fungerte like vel.

Jeg tror kanskje at ved å planlegge prosessen bedre kunne det ha hjulpet meg bedre i gang. Kanskje det ville være motiverende å ha en kanban-tavle til å begynne med for å kunne lettere visualisere fremgangen i prosjektet, men dette er nesten umulig å si. Jeg har erfart mye gjennom denne prosessen, jeg har også lært mye av alt som gikk feil i prosessen. Jeg vet derimot ikke om jeg vil klare å forbedre meg. Om jeg vil fortsette å skuffe, eller om en bedre oversikt og organisering av av et prosjekt kan ha en så stor effekt på min psyke.

3 Teknologier brukt i løsning og utvikling

I dette prosjektet var det ikke mange krav om hvilke språk eller rammeverk som skulle brukes for uten om c3js som skal brukes til å generere grafer, og databasesystemet som skal bruke er PostgreSQL. Utover dette har jeg valgt å bruke de språkene og rammeverkene som jeg er mest kjent med selv, men vil diskutere i et senere avsnitt hvilke andre muligheter jeg hadde, som de språkene ForzaSys har brukt i pmSys som er Go og React.

3.1 Backend teknologier

3.1.1 Python Flask

Flask er et web rammeverk skrevet i Python. Det er ansett som et micro-rammeverk, som betyr at den har kun det nødvendige for å kunne kjøre en web server, men lar brukeren bestemme selv hva som bygges rundt det (Pallets Projects, u.å.). Som i dette prosjektet var nytte siden det var krav om bruk av PostgreSQL, og det kan derfor enkelt legges til som en avhengighet. Flask er i dette prosjektet brukt for å lage web serveren til siden og håndterer derfor trafikken som går til siden, og er den som serverer nettsiden til brukeren ved hjelp av routing egenskapen dens. Flask er også API-et til siden som brukes til å kunne få tak i ressursene som trengs for de forskjellige spørringene som brukes til å lage grafene.

3.1.2 Python Psycopg2

Psycopg2 er en PostgreSQL database adapter for Python. Det er brukt til å kunne ha kommunikasjon mellom databasen, som er satt opp med et utsnitt tildelt fra ForzaSys, og web serveren. Den håndterte spørringene til databasen og returnerer tuples med resultatet som så kan bli håndtert av Python og sendt videre til klienten som JSON med Flask.

3.1.3 PostgreSQL

PostgreSQL var satt som et krav fra ForzaSys at skulle bli brukt og er derfor databasesystemet for dette prosjektet. PostgreSQL er et open source objekt-relasjons database system. Det er omtalt for å være et veldig stabilt, sikkert og skalerbar system med mange robuste funksjoner. Det har blitt utviklet i over 30 år, og som et open-source program har samfunnet rundt PostgreSQL hjulpet med å finne forbedringer over tid. (PostgreSQL, u.å.)

3.2 Frontend teknologier

3.2.1 HTML

For å bygge opp strukturen til nettsiden har jeg brukt ren HTML, men gjort finere med rammeverket Bootstrap. HTML er en markup language som definerer elementer på siden som kan inneholde og vise tekst og figurer, eller hjelpe strukturere hvordan innholdet vises. HTML er brukt for å definere elementene som knapper, menyer, og innholdsområder.

3.2.2 Bootstrap

Det er brukt Bootstrap 4 for å kunne få et enkelt og fint design på siden ved hjelp av dens forhånds definerte klasser. Bootstrap er et open-source front-end rammeverk for web applikasjoner. Bootstrap ble utviklet av Twitter og er bygd opp av HTML, CSS og JavaScript. Det ble laget med tanke på responsiv, mobil først sider, og dette gjør det enkelt å kunne lage en side som vil fungere på flere oppløsninger (GetBootstrap, u.å.). For dette prosjektet var nyttig siden fokuset var ikke nødvendigvis på design, men det fortsatt noe som trengs å vurderes når man lager en nettside, og Bootstrap gjør at det ikke er noe å sette for stort fokus på for den har enkle klasser som brukes for dette formålet. Bootstrap er brukt til blant annet å endre designet på knappene, nedtrekks listene, og tabellene.

3.2.3 JavaScript

JavaScript er brukt for kommunikasjon mellom klienten og serveren, dynamisk generering av diverse elementer på siden, og hente inn input fra brukeren. JavaScript er det mest brukte språket for webutvikling som klientside språket hvor statistikk viser til at 97.9% av nettsider bruker JavaScript. JavaScript er også et skriptspråk som kan brukes til mye annet også som software, hardware kontroller og servere (Meltzer, 2020). JavaScript er dette prosjektet brukt til for eksempel å om strukturere informasjonen som kommer inn fra serveren. Serveren gir fra seg en array med data, og JavaScript kjører så i løkker i tabellene mottatt fra serveren og strukturerer det som HTML tekst som kan skrives ut på siden.

3.2.4 jQuery

jQuery er et JavaScript bibliotek. jQuery gjør det enklere å bruke noen av funksjonene til JavaScript ved hjelp av deres forkortet tolkning av språket, hvor JavaScript ville brukt flere linjer med kode kan jQuery klarer det på bare en. Jeg har valg å bruke dette siden det gjør

koden litt mer lesbar og enklere å skrive etter min mening. En del av jQuery jeg har brukt er AJAX funksjonene deres som er de funksjonene som håndterer kommunikasjon med server API-et. Så ved GET kall gjennom AJAX vil den kunne få tak i ressursene på serveren, som databasespørringer, og kunne oppdatere nettsiden uten å oppfriske nettsiden.

3.2.5 C3.js

C3js er et bibliotek som hjelper til å lage grafer og er det biblioteket som ForzaSys bruker i pmSys og var derfor krav om å bruke det. C3js er basert på d3js som er et JavaScript bibliotek for manipulering av data (D3js, u.å.)(C3js, u.å.). C3js gjorde det mulig og kunne enkelt lage grafene på siden og kunne justere presentasjonen av dataene ved å bytte noen variabler, eller i forbindelse med tooltip delen, kan overskrives helt for egen tilpasning.

3.3 Utviklingsmiljø

Jeg fikk tildelt en Ubuntu server fra ForzaSys hvor jeg skulle kjøre løsningen. På denne serveren installerte jeg Docker for å kunne enkelt laste ned og bruke de programmene jeg trengte. Dette gjorde det ganske naturlig å kjøre alt i Docker gjennom hele prosessen både med filene på serveren og når jeg kjørte koden lokalt.

3.3.1 Docker

Docker er en open-source containerization platform. Containerization er et nyere alternativ til virtualisering hvor i stede for å kjøre en hel OS instanse trenger en container kun OS prosessene med de nødvendige avhengighetene. Docker var brukt til å kunne enkelt og trygt deployere siden på serveren ved at Docker vil alltid gi det samme resultatet uansett hvor filene kjøres (IBM, 2021). Så ved å lage en Dockerfile for å bygge en docker container lokalt med alle avhengighetene som er nødvendig, når den oppfører seg som den skal kan den lett overføres til serveren.

3.3.2 Visual Studio Code

VS Code er en kode editor laget av Microsoft. VS Code er et verktøy som vil kun gi hva som trengs for å ha en rask code-build-debug syklus, med nyttige funksjoner til ting som debugging, task running og version control (Microsoft, u.å.), . Den har mange utvidelser som kan lastes ned for hva som blir brukt, som i dette prosjektet hvor jeg kan enkelt få VS Code

til å gi syntax feedback for språk som SQL og Python, men også utvidelser for Docker for å kunne overvåke containerene og imagene der.

3.3.3 Git og GitHub

Git er et versjonskontrollsystem. Det hjelper med å kunne ha orden på endringer i koden og kunne tilbakestille til en tidligere versjon om nødvendig. GitHub er en side hvor man kan enkelt dele Git baserte prosjekter. I dette prosjektet har jeg jobbet alene så styrken til Git hvor man har kontroll over alle som jobber på prosjektet og hva som har blitt gjort og kunne raskt dele endringer er ikke tatt fordel av, men det har vært nyttig for å kunne ha en stabil versjon å falle tilbake på om noe skulle gå feil. I tillegg gjør det enklere å jobbe på forskjellige enheter (som desktop og laptop). Og GitHub var brukt for å kunne dele koden med ForzaSys så de kan se over det som er gjort når de ville.

3.4 Andre Verktøy

Det er noen verktøy som ikke er en del av verken frontend eller backend, men brukt i forbindelse med testing av løsningen.

3.4.1 Chrome DevTools

Mesteparten av prosessen har blitt kjørt i Chrome, men er også regelmessig sjekket i andre nettlesere som Safari og Edge. Chrome har en veldig nyttig fane hvor man kan se konsoll output, nettverk responser og hvor man kan kjøre siden steg for steg ved bruk av breakpoints. Mange errors blir fanget opp av Flask når det kommer til kommunikasjon med backend delen, men for frontend er Chrome sin DevTools veldig nyttig. Det ble for eksempel brukt mye i sammenheng med C3.js. Jeg kunne sende output og objekter som C3.js lager til konsollen i Chrome for å kunne lettere forstå deres struktur og med det finne informasjonen jeg trengte fra dem. Breakpoints er nyttig for å kunne fange opp feil som skjer gjennom løkker som mye av det visuelle for statistikken er bygd opp av.

3.4.2 HTTPie

HTTPie er et program som kan vise output fra API kall. Dataen som blir sendt mellom backend og frontend hadde ofte mange forskjellige elementer som måtte håndteres samtidig. Datoene som skulle vises, antall som mangler, navn på spillere som mangler, de

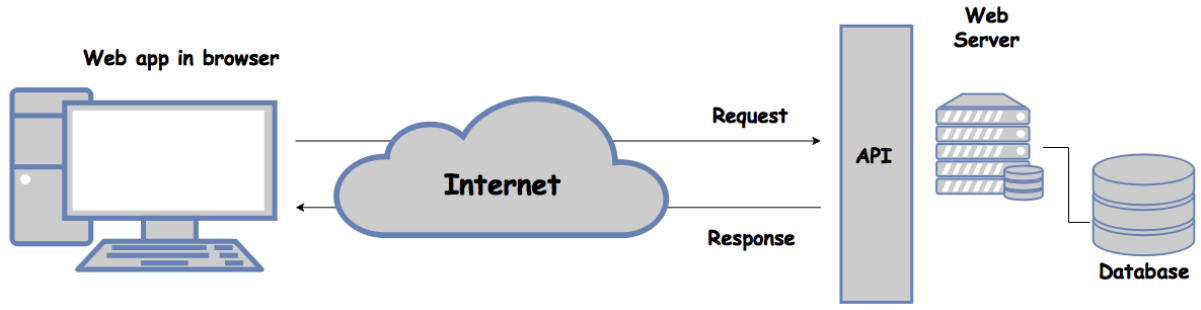
forskjellige type rapportene som alle skal kunne vises på en gang. HTTPie hjelper med å kunne enkelt gjøre et API kall i konsollen med parametrene man vil sjekke for og se om den gir ønsket output og struktur, uten å måtte gå til nettsiden å justere diverse input gjennom brukergrensesnittet.

4 Produktdokumentasjon

4.1 Introduksjon

Det som skal lages er et system som skal kunne lese data fra en database med statistikk fra spillerne på diverse lag om forskjellige rapport typer. Målet er å kunne lage grafer og tabeller som er enkle å forstå visuelt og har all nødvendig informasjon tilgjengelig. Det må kunne være fleksibilitet for hva man har lyst til å vise når det kommer til type rapport, og tidsintervaller.

Webapplikasjonen er bygd opp av en frontend, som er det bruken ser og bruker på siden, og en backend, som er hvor serveren håndterer flyten av data. Figur 2 viser en illustrasjon av hvordan alle komponentene er tilkoblet til hverandre. Når en bruker interagerer med knapper i brukergrensesnittet vil den sende en forespørsel gjennom API-et bygd på serveren som så vil gjøre en spørring til databasen og deretter sende en respons med dataen den fant til brukergrensesnittet som så viser dataen den fikk tilbake. .



Figur 2: : Illustrasjon av hvordan systemet kommuniserer. Hentet fra Gabry Martinez (Martinez, 2018).

Det er selvfølgelig mange webapplikasjoner med lik struktur som min. Jeg har selv laget sider som dette før i et tidligere emner i studiet, Datanettverk og Skytjenester, som jeg kunne bruke som utgangspunkt for systemarkitekturen. Det som det ikke er noen tidligere løsning for er grafene jeg skal lage. Dette er en ny funksjonalitet for ForzaSys og man vil ikke kunne finne en lignende løsning tilgjengelig for dette andre steder heller. Løsningen må derfor lages fra bunn med hjelp av rammeverkene jeg har tilgang til å lese gjennom deres dokumentasjon for å kunne få en forståelse for hvordan de kan brukes i denne løsningen. Og itereres på flere ganger for å få et design som vil være oversiktlig og brukervennlig for hensikten om å kunne ta til seg informasjonen vist gjennom grafene og tabellene.

4.2 Backend

Backenden består av API-et som frontend kommuniserer gjennom, spørringene til databasen, og routing til de forskjellige sidene. I denne delen vil jeg beskrive backenden struktur, kode og funksjonalitet.

4.2.1 Systemarkitektur for backend

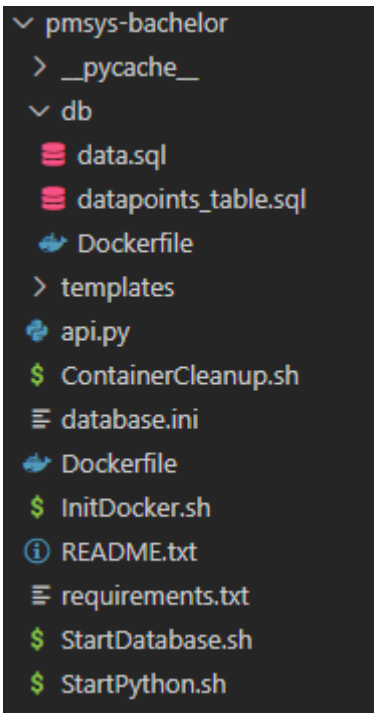
Systemarkitekturen beskriver hvordan komponentene er koblet sammen og hvordan dataflyten mellom dem er håndtert. Dette systemet er bygd opp av databasen, serveren og API-en.

Databasen inneholder all data om spillerne, serveren håndterer routing av nettsidene, og API-et distribuerer dataen til brukergrensesnittet. Det er serveren som genererer SQL spørringer ut i fra parameterne gitt gjennom API-et, og om formaterer både input fra API-et og output fra databasen.

Når en bruker for eksempel laster opp nettsiden vil serveren servere indeks siden som er versjon 1, deretter vil den sende et API kall til serveren hvor den spør om default dataen som skal vises ved oppstart. Serveren vil så starte en funksjon som gjør en rekke forespørsler til databasen om dataen nødvendig for å lage grafen, dataen nødvendig for å lage tabellen, og dataen nødvendig for å genere detaljerte tooltips. Dataen blir så konvertert til JSON for å så bli sendt til nettsiden.

4.2.2 Mappestruktur

Det er 2 mapper som holder databasen og frontend delen av siden, mens server delen ligger rett i rotmappen. Grunnen til dette er for at Flask skal kunne hente ut nettsider må disse lagres i en mappe som heter “templates” (for uten om index siden, men i denne sammenhengen ga det mening å samle alt på ett sted siden det ikke er en faktisk forside). Den enkleste måten å hente ut filer fra templates mappen er om den er et nivå over Flask filen siden dette gjør at man kan skrive en relative path fra Flask i stede for å definite en absolutt path. Dette gjør kanskje at det er en litt mindre oversiktlig mappestruktur i stede for å ha en dedikert frontend og backend mappe, men i et prosjekt med så få filer som dette virket det litt unødvendig å separere dem. Dette er mulig fordi



Figur 3: Skjermbilde av rotmappen

Flask er et lightweight server rammeverk som ikke trenger en robust mappestruktur med mange avhengighet filer rundt selve server filen.

Server filen, altså Flask applikasjonen, er den som heter api.py. Det er denne filen som inneholder alle API funksjoner og definerer routing til de forskjellige versjonene på nettsiden. Når denne filen startes vil den starte å lytte til port 80. Den kjøres når Dockerfilen i rotmappen kjøres.

Mappen db inneholder 3 filer hvor datapoints_table.sql definerer strukturen til databasen og data.sql inneholder all dataen som jeg har fått tildelt fra ForzaSys. Når Dockerfilen i denne mappen kjøres vil den sette opp databasen og starte å lytte til port 5432 som er standard port for PostgreSQL.

Denne mappen ble overført til serveren ved bruk av ssh for å logge seg på serveren for så å overføre og kjøre filene.

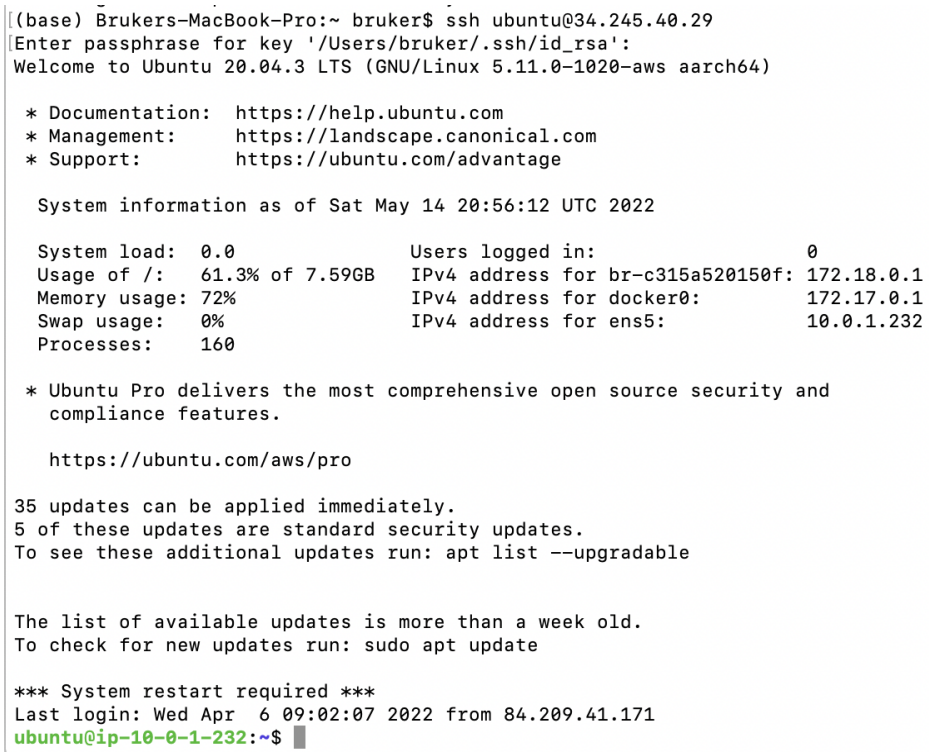
4.2.3 SSH-protokoll

Tomas Kupka fra ForzaSys ga meg tilgang til en Ubuntu server for å utgi løsningen min. For at han skulle kunne gi meg en sikker tilgang sendte jeg en ssh public key for at han skal kunne legge meg til som gyldig bruker på serveren. SSH public key-en ble generert slik:

```
(base) Brukers-MacBook-Pro:~ bruker$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/bruker/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/bruker/.ssh/id_rsa.
Your public key has been saved in /Users/bruker/.ssh/id_rsa.pub.
```

Figur 4: Skjermbilde av konsollen under generering av public keys

For å så lage en forbindelse til serveren bruker man følgende kommando og bruker passordet registeret i public key-en som ble registrert som vist i figur 5.



Figur 5: Skjermbilde av innlogging på Ubuntu serveren

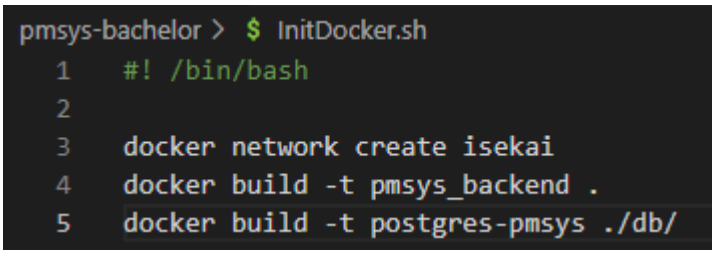
4.2.4 Overføring av filer

For å overføre filene mine til serveren brukte jeg GitHub. Ved å simpelthen klone repository-et en gang på serveren kan jeg så få alle oppdateringer jeg gjør til serveren med en eneste “git pull”.

4.2.5 Starte opp serveren

Med alle filene lastet ned på serveren kan løsningen startes enkelt opp med hjelp av script-filene som er lagt ved. De er ikke ekstremt komplekse, men de gjør at det ikke er nødvendig å huske lange docker kommandoer for hver gang man kjører siden så dette er min foretrukne måte å kjøre docker på.

Først vil man måtte kjøre InitDocker.sh.

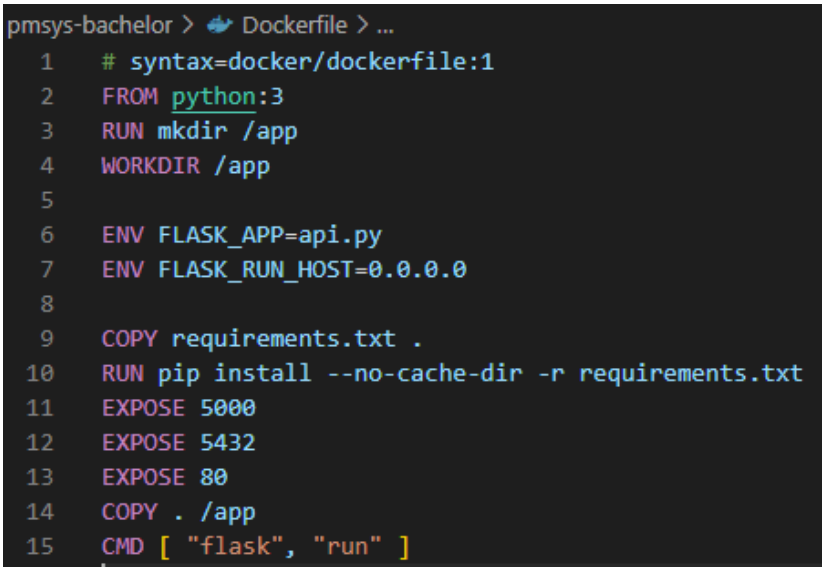


Figur 6: Skjermbilde av initDocker.sh filen som inneholder bygging av Docker images

Dette skriptet vil først definere et docker nettverk. Dette er nødvendig for at de 2 conainterene som lages skal kunne kommunisere med hverandre. Uten den vil ikke databasen kunne sende data til serveren.

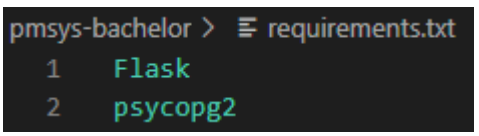
Deretter bygges 2 images, en for serveren og den andre for databasen. Disse er laget ut fra Dockerfilene som ligger i mappen som er angitt (“.” som er rotmappen der serveren er, og ./db/ for databasen). Docker har en bevisst forskjell på kommandoer som er kjørt når imaget bygges, og når en container basert på imaget startes, og dette er definert i Dockerfilen.

Figur 7 viser Dockerfilen til serveren. Den bygger imaget med Python som trengs for å kjøre Flask, den lager en mappe som alle filene skal kopieres over til og setter opp noen globale variabler som Flask trenger for å kunne definere serveren.



Figur 7: Skjermbilde av Dockerfilen for serveren

Filen requirements.txt inneholder en liten liste med de rammeverkene som må lastes ned. Ved å lage en tekstfil med programmene som trengs å lastes ned for programmet gjør det at for hver avhengighet som blir lagt til trengs ikke den å hardkodes inn i dockerfilen, men i stedet bare legge til navnet i requirements filen. Når imaget bygges vil disse bli lastet ned, og siden dette imaget er basert på Python kan pip brukes som er en package installer for Python. Dette er en nyttig egenskap som Docker har hvor ved å laste ned disse rammeverkene i bygg fasen av vil man kunne starte og stoppe containere så ofte man vil uten å laste ned mange filer om og om igjen siden de er allerede en del av imaget som definerer containeren.



Figur 8: Skjermbilde av requirements.txt filen

Deretter vil imaget åpne opp for kommunikasjon til port 5000, 5432, og 80. 5000 er porten Flask bruker som default. Så når man lokalt tester med Flask er det port 5000 som er brukt ved default å må derfor åpnes. 5432 er for kommunikasjon med PostgreSQL. Port 80 er porten brukt til HTTP, Hypertext Transfer Protocol, som er nødvendig å åpne for at serveren skal kunne servere nettsiden på nett. Det er en linje i bånd av api.py som hjelper å definere hvilken port som skal brukes av serveren som justeres etter som den er på serveren eller lokalt.

```python
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
```

Figur 9: Skjermbilde av koden som avgjørt porten som brukes

Imaget for databasen er bygget med følgende Dockerfile:

pmsys-bachelor > db > Dockerfile > ...

```
1  # syntax=docker/dockerfile:1
2  FROM postgres
3  ENV POSTGRES_PASSWORD pass123
4  ENV POSTGRES_DB pmsysdb
5  COPY datapoints_table.sql /docker-entrypoint-initdb.d/
6  COPY data.sql /docker-entrypoint-initdb.d/
7  EXPOSE 5432
```

Figur 10: Skjermbilde av Dockerfilen for databasen

Den lager et image med Postgres som base og lager noen globale variabler for å definere navn og passord til databasen. Den tar så å kopierer over sql filene og omgjør disse til filer som docker imaget av postgres kan bruke til å initiere databasen for så å åpne port 5432 for kommunikasjon med serveren.

Etter at imagene er bygd kan containere basert på imagene startes. Her kan filene StartDatabase.sh og StartPython.sh kjøres for å starte en container. Siden Flask filen er avhengig av at databasen kjøres må den startes først for at python applikasjonen skal kunne fungere.

StartDatabase.sh har kun 1 kommando i seg:

```
docker run -d -it --rm --network isekai --name postgres-odd -p 5432:5432 postgres-pmsys
```

Denne kommandoen vil kjøre en Docker container. De forskjellige opsjonene lagt til betyr følgende:

-d	Detached mode. Dette gjør at containeren vil kjøre i bakgrunnen. Og i kombinasjon med –rm holde containeren aktiv til den er fjernet.
-it	Interactive. Denne gjør at jeg kan koble meg på containeren for å kunne se output fra den som en vanlig konsoll. Dette er nyttig om jeg vil koble meg til for å se feilmeldinger for debugging.
--rm	Remove. Denne sletter containeren om den stoppes. Dette fjerner et trinn når man vil restarte en container ved at en simple “stop” vil både stoppe og slette containeren.
--network	Denne definerer hvilket nettverk som containeren skal kobles opp mot som er viktig for at flere containere skal kommunisere med hverandre.
--name	Definerer et navn for containeren. Alle containere har en ID som kan brukes for å referere til en spesifikk container, men ved bruk av navn blir det mye mer forutsigbart og leselig for hva containeren refereres til som.
-p	Publiserer portene som er definert. Som betyr her at det er mulig å lese fra port 5432.

Og det siste ordet “postgres-pmsys” definerer hvilket image som skal brukes som base for containeren.

Når databasen har startet kan Flask applikasjonen startes. Den er noen forskjeller her, men mye det samme også. StartPython.sh skriptet ser slik ut:

```
port=$1
docker run -d -it --rm --network isekai --name pyback-pmsys -p
$port:$port -v $(pwd)/api.py:/app/api.py -v
$(pwd)/templates/:/app/templates -t pmsys_backend python
/app/api.py
```

Denne tar inn en parameter siden den skal kunne kjøre både lokalt for testing og på serveren. Så når den kjøres lokalt legges port 5000 til, mens når den er på serveren port 80.

Den andre forskjellen er bruk av volumes. Docker volumes har en del fordeler. I denne løsningen er den brukt for å kunne synkronisere filer til imaget når en container startes. Uten

volumes vil filene som kopieres i bygg fasen av imaget bli brukt i stedet. Dette ble ikke brukt for database containeren siden dataen i databasen ikke oppdateres ofte etter å ha lagt inn utsnittet fra ForzaSys den første gangen.

Disse 2 filene kunne ha blitt gjort om til bare 1, men under testing er det praktisk å kunne kjøre en container i forskjellige vinduer for å se output fra begge. Det er også ikke alle filer som oppdateres i webapplikasjonen ved en simpel oppfriskning av nettleseren. For eksempel JavaScript filer oppdateres, men Flask applikasjonen gjør det ikke. For disse filene kreves en server restart. Databasen derimot endres ikke og trenger ingen restart. Derfor er det praktisk å kunne starte hver for seg selv, med et skript per.

4.2.6 Databasen

Jeg fikk tildelt et utsnitt fra pmSys sin database mellom 1. Mai 2021 og 25. Oktober 2021. Jeg fikk bare tilgang til det nødvendige for å lage spørringene jeg trengte, og ikke hele pmSys databasen. Dette betyr at jeg hadde tilgang til rapport tabellen deres og ikke noe mer. Dette er altså ikke en avansert relasjonsdatabase som jeg jobber med, men kun en tabell som har den dataen jeg trenger. De datapunktene jeg har å jobbe med er som følgene:

pmsys-bachelor > db > datapoints_table.sql

```
1  create table datapoints
2  (
3      sk                                          bigserial primary key,
4      oid                                         varchar(64)                          not null,
5      owner                                       varchar(64)                          not null,
6      created                                     timestamp with time zone default now() not null,
7      modified                                    timestamp with time zone default now() not null,
8      effective                                   timestamp with time zone             not null,
9      timezone                                    varchar(64)                          not null,
10     timeoffset                                  integer                              not null,
11     schema_namespace                            varchar(32)                          not null,
12     schema_name                                 varchar(32)                          not null,
13     schema_version_major                        integer                              not null,
14     schema_version_minor                        integer                              not null,
15     schema_version_patch                        integer                 default 0    not null,
16     acquisitionprovenance_sourcename            varchar(32)                          not null,
17     acquisitionprovenance_sourcecreationdatetime timestamp with time zone            not null,
18     acquisitionprovenance_modality              varchar(100)                         not null,
19     additionalproperties                        json                                 not null,
20     body                                        json                                 not null
21 );
```

Figur 11: Skjermbilde av datapunktene i databasen

Det er ikke mange av disse datapunktene som er relevant til alle spørringene som ble gjort. De mest aktuelle er “owner” som er en auth0 autoriserings token som identifiserer spilleren, en slik token kan se for eksempel slik ut: “auth0|59b814917091e711e6300265”. “Created” er

dagen rapporten ble laget. “Schema_name” er navn på rapporten, hvorav jeg har blitt tildelt 3 rapport typer, wellness, participation og srpe. En utfordring med dataen er at jeg har ingen info om spillerne mer enn en tilsynelatende tilfeldig streng av tall og bokstaver. Jeg fikk beskjed om at en trener som ser på statistikken vil kunne ha tilgang til fler enn et lag med spillere, men siden jeg ikke har tilgang til informasjon om spillerne ble jeg bedt om å finne en annen måte å definere et lag på. Jeg har heller ikke noe navn på spillerne så jeg må også finne en måte å omgjøre dette siden det ikke ser bra ut eller realistisk ut i løsningen med denne tilfeldige strengen.

For å identifisere lag fant jeg et mønster i auth0 token-ene. Alle starter med “auth0|”, etter det kommer 3 forskjellige mønstre som flere spillere deler. De er 59b8149, 59b8148 og 59b6e. For å sjekke hvilket lage en spiller har i spørringen gjøres: `owner LIKE '%{}%'` hvor {} vil være input for hvilket lage spørringen gjelder, som for eksemple %59b6e%.

For den andre utfordringen med realistiske navn ble dette løst på frontend-en. En javascript funksjon har alle auth0 tokene jeg har i datasettet mitt (hvorav det er 24 totalt) med en rekke if-statements som for hver spiller vil sjekke auth0 strengen og oversette den til et bestemt navn definert for den spilleren. Disse navnene var generert ved bruk av en nettside som heter www.behindthename.com for å enkelt få en liste med navn jeg kunne bruke.

For å koble databasen til serveren er det en Psycopg2 funksjon som heter “connect” som tar inn parametere, som navn og passord, for å starte en link til databasen.

```
conn = psycopg2.connect(
    host="postgres-odd",
    database="pmsysdb",
    user="postgres",
    password="pass123")
```

Figur 12: Skjermbilde av verdier nødvendig for å kobles til databasen

4.2.7 Flask Webserver og API

For routing på siden for å servere nettsidene brukes en Flask funksjon som heter “render_template”. Det er denne som gjør at det er nødvendig å ha frontend filene i en mappe som heter “templates”. Det er der denne funksjonen leter etter filen spesifiser og sender den til nettleseren.

```
@app.route("/")
def version1():
    return flask.render_template("stats.html")
```

```
@app.route("/version2")
def version2():
    return flask.render_template("stats2.html")
```

Figur 13: Skjermbilde av serverens servering av nettsidene

Når siden startes opp vil den gjøre et API kall med den initielle dataen siden vil vise. Denne vil returnere 3 tabeller til klienten. Den første med data for å lage grafen. Den andre med data for å lage en tabell med dagens rapport status. Og den tredje med en liste navn/auth0-tokens om hvem som mangler for de forskjellige rapportene per dag.

```python
@app.route("/api/stats/initv1", methods=["GET"])
def initalFetchForVersion1():
    totalReturn = []
    print("Trying to add Initial Stats")
    totalReturn.append(getReportStat(1))
    print("Trying to add TodaysReport")
    totalReturn.append(getTodaysReport(1))
    print("Trying to make list of absent players.")
    totalReturn.append(listOfAbsentPlayers(1))
    return flask.jsonify(totalReturn)
```

Figur 14: Skjermbilde av funksjonen som startes ved åpning av nettsiden

4.2.7.1 getReportStat()

Denne funksjonen gjøre en spørring til databasen etter hvor mange som har rapportert per dag i et angitt tidsintervall.

```python
def getReportStat(called=0):
    args1 = flask.request.args.get("reports")
    args2 = flask.request.args.get("start_time")
    args4 = flask.request.args.get("end_time")
    args3 = flask.request.args.get("team")
    args5 = flask.request.args.get("ignore")
    print("Printing ignore input: {}".format(args5))
    ignored = args5.split(",")
    print("Printing handled ignores: {}".format(ignored))

    reports = args1.split(",")

    print("Checking reports for {}, start-time {}, end-time {}, team {}".format(reports, args2, args4, args3))
    cur = conn.cursor()
    all_data = []

    ignore_string = "("
    for i in ignored:
        ignore_string += "'" + i + "'"
        if i != ignored[-1]:
            ignore_string += ", "
    ignore_string += ")"
    print("SQL ready ignore list: {}".format(ignore_string))

    for r in reports:
        cur.execute("""SELECT DATE(created), count(DISTINCT owner)
                       FROM datapoints
                       WHERE schema_name='{}' AND created>='{}' AND created<='{}' AND owner LIKE '%{}%' AND owner NOT IN {}
                       GROUP BY DATE(created) ORDER BY DATE(created);""".format(r, args2, args4, args3, ignore_string))
        db_query = cur.fetchall()
        db_query.insert(0, r)
        all_data.append(db_query)

    if called == 1:
        return all_data
    else:
        return flask.jsonify(all_data)
```

Figur 15: Skjermbilde av koden som skaffer rapporterings statistikken

Først leses inn alle parametrene sendt med API kallet. Deretter om formatere noen av dataen som kommer inn. Alt som sendes over API-en er bare tekststrenger så listen med rapporter som skal sjekkes må gjøres om til en faktisk liste for å enklere kunne gå gjennom dem. Og samme for listen med ignorerte spillere. Listen for ignorerte spillere må også omformateres litt ekstra for å gjøres kompatibel med SQL syntaks for når spørringen blir utført.

Det må så lages en cursor fra Psycopg2 for å kunne gjøre databasespørringer. Den kjøres så gjennom en løkke for hver rapport som blir spurt etter. For hver rapport vil den lage en liste med alle datoene i tidsintervallet som er angitt og hvor mange for hver dato rapporterte på den dagen. Det er viktig her å definere “DISTINCT” for spillerne siden det er noen type rapporter hvor spillere kan rapportere flere ganger om dagen, men det er ikke viktig i denne statistikken, bare at de har rapportert i det hele tatt. Så dette gjør at man bare får unike spillere telt opp per dag. SQL spørringene er også gjort veldig oversiktelig med Python sin egenskap til å lage flerlinjet strenger ved bruk av triple anførselstegn (“““). På den måten er det lettere å følge med på hva som spørres etter når man skriver spørringene.

Listen som blir laget blir så samlet for å kunne sende til frontend-en. Den siste if-statement-en er for å sjekke om denne funksjonen ble kalt på av en annen funksjon eller ikke. Noen ganger gir det mening å gjøre et API kall direkte til denne funksjonen, mens andre ganger gir det mer mening å samle data fra flere funksjoner på en gang, som her hvor funksjonen ble kalt samtidig som andre funksjoner nødvendig for oppstart. Derfor sender oppstarts funksjonen med en 1’er for å definere at det ikke er nødvendig for denne funksjonen å gjøre dataen klar for nettverk overføring og heller sende Python listen. Det er her nyttig å kunne bruke Python sin “default parameter” egenskap, så om den ble kalt direkte av API-et vil den ha verdien som er definert ved bruk av default parameteren.

4.2.7.2 getTodaysReport()

Denne funksjonen gjør en spørring for en spesifikk dag og sjekker om hver rapport har blitt rapportert eller ikke rapportert for hver spiller på et angitt laget.

```
@app.route("/api/stat/report/today", methods=["GET"])
def getTodaysReport(called=0):
    team = flask.request.args.get("team")
    date = flask.request.args.get("end_time")
    args3 = flask.request.args.get("reports")
    reports = args3.split(",")
    print("Input Reports: {}".format(reports))
    #reports = ["wellness", "participation", "srpe"]

    cur = conn.cursor()
    all_data = []

    # Finds all players on the team.
    cur.execute("""SELECT DISTINCT owner
            FROM datapoints
            WHERE owner LIKE '%{}%'""".format(team))

    db_query = cur.fetchall()

    db_list = [list(i) for i in db_query]  # Converts to list. I can not change a tuple which is what the queries gets me.

    # Fills the list with the name of all players.
    for n in db_list:
        all_data.append(n)

    # Finds everyone that reported for the given day on the given team.
    # Looping through the reports selected.
    for r in reports:
        cur.execute("""SELECT DISTINCT owner
                FROM datapoints
                WHERE schema_name='{}' AND owner LIKE '%{}%' AND to_date(CAST(created as char(10)), 'YYYY-MM-DD') ='{}'
                ORDER BY(owner);""".format( r, team, date))
        db_query = cur.fetchall()
        db_list = [list(i) for i in db_query] # Converts tuple to list
        # Loops through all players and checks if the name can be found in the reported list.
        # And if it is found appends 1 and if not found it will trigger an error which is caught and adjusts the check accordingly.
        for n in all_data:
            try:
                x = [n[0]]      # As this is an array of names the name searched for needs to also be in an array to find identical searches.
                report_check = db_list.index(x)
            except ValueError:
                report_check = -1

            if report_check != -1:
                n.append(1)
            else:
                n.append(0)

    all_data.insert(0, reports)
    print("Here is All Data: ")
    print(all_data)

    if called == 1:
        return all_data
    else:
        return flask.jsonify(all_data)
```

Figur 16: Skjermbilde av kode som lager en tabell med statusen for alle spilleres rapporter

Samme som getReportStat() hvor den tar inn parametere og lager en kobling til databasen gjennom Psycopg2. Først gjøres en spørring etter alle spillerne for laget angitt. Når man bruker fetchall() funksjonen til Psycopg2 returnerer den en tuple som er en Python spesifikk datatype som kan se veldig lik ut som en liste, men har en egenskap som hindrer den fra å

bli endret på. Dette gjør dem umulig å jobbe med for dette formålet så jeg må gjøre alle tuples om til lister for å kunne slå sammen dataen jeg ønsker å sende videre.

Den andre SQL spørringen går i en løkke for å sjekke de forskjellige rapportene, og for hver rapport vil den notere navn/auth0-token på de den finner for den dagen.
Den skal så lage en liste som registrerer om en spiller har rapportert eller ikke. En løkke går gjennom alle spillerne på laget som ble funnet i begynnelsen og sjekker om den spilleren er i listen for de som har rapportert. Python har en nyttig funksjon for lister som kan sjekke om et element finnes, denne heter .index(). Problemet med index funksjonen derimot er at den sender en error om den ikke finner det som er søkt etter. Så for å fange opp at en spiller ikke finnes i listen med rapporter må en try settes rundt funksjonen og om den fanger en ValueError vil den justere verdien som sjekkes videre. Om spilleren har rapportert vil index returnere posisjonen dems i listen, som vil være et tall som er 0 eller større. Det er nå en klar definering på om en spiller er i rapporterings listen eller ikke og det blir lagt til i en listen som skal sendes til frontend som 1 eller 0 om de har rapportert eller ikke. Det legges også tilslutt inn hvilke rapporter som er sjekket og igjen en sjekk om denne funksjonen trengs å pakke sammen dataen for å sendes til frontend, eller om den skal samles et sted først.

4.2.7.3 listOfAbsentPlayers()

I denne funksjonen skal det lages en liste med spillere som ikke har rapportert per dag i det angitte tidsintervallet. Det var viktig å være nøye her med hvordan dataen ble strukturert som er hvorfor datoen blir formatert til noe mer enn en tekststreng. Dette er fordi C3.js har ingen funksjon for å lese info fra x-aksen (der man vil vanligvis se datoene) fra hva jeg fant gjennom dokumentasjonen og testing av C3.js objektene jeg kan lese fra. Så det var derfor viktig å ha en komplett, en-til-en, liste med datoer for å kunne heller bruke C3.js sin index parameter, som hver node i grafen har tilgjengelig, for å kunne vite hvilken dato i listen for manglende spillere den noden samsvarer med.

```python
@app.route("/api/stats/absent/all", methods=["GET"])
def listOfAbsentPlayers(called=0):

    team = flask.request.args.get("team")
    start_date = flask.request.args.get("start_time")
    end_date = flask.request.args.get("end_time")
    args3 = flask.request.args.get("reports")
    reports = args3.split(",")
    print("Input Reports: {}".format(reports))

    args5 = flask.request.args.get("ignore")
    ignored = args5.split(",")

    ignore_string = "("
    for i in ignored:
        ignore_string += "'" + i + "'"
        if i != ignored[-1]:
            ignore_string += ", "
    ignore_string += ")"

    datetime_start = datetime.date(int(start_date[0:4]), int(start_date[5:7]), int(start_date[8:10]))
    datetime_end = datetime.date(int(end_date[0:4]), int(end_date[5:7]), int(end_date[8:10]))
    print("Start Datetime object: {}".format(datetime_start))

    cur = conn.cursor()
    all_data = []

    while datetime_start < datetime_end:
        temp_day = []
        temp_day.append(datetime_start)
        for r in reports:
            cur.execute("""SELECT DISTINCT owner
                FROM datapoints
                WHERE owner LIKE '%{}%' AND owner NOT IN {}
                EXCEPT
                SELECT DISTINCT owner
                FROM datapoints
                WHERE schema_name='{}' AND owner LIKE '%{}%' AND to_date(CAST(created as char(10)), 'YYYY-MM-DD') ='{}'
                ORDER BY(owner);""".format(team, ignore_string, r, team, datetime_start))

            db_query = cur.fetchall()
            temp_data = []
            temp_data.append(r)
            temp_data.append(db_query)
            temp_day.append(temp_data)

        all_data.append(temp_day)
        datetime_start = datetime_start + datetime.timedelta(days=1)
    if called == 1:
        return all_data
    else:
        return flask.jsonify(all_data)
```

Figur 18: Skjermbilde av kode som lager en liste med spillere som mangler rapporter

Som i getTodaysReport () og getRapportStat() leses inn parametere og omformateres til å være SQL kompatible, men her må også datoen gjøres om til et datetime-objekt. Datetime er fra et Python bibliotek for å håndtere tider og datoer. Dette er nødvendig for å kunne lage en løkke som går gjennom alle dagene i tidsintervallet. Denne SQL spørringen finner først alle spillerne på et lag, og etter det trekker fra alle spillere som har rapportert på det laget. Dette gjør da at de som er igjen er de spillerne på et lag som mangler rapporten spurt etter. Det er mye som må søkes gjennom i denne spørringen, men det er mest på grunn av konsekvensen av å ikke ha tilgang til hvem som spiller for hvert lag. Om dette var en større relasjonsdatabase ville det mest sannsynlig være en tabell med lag, og spillere til hvert lag som kunne leses fra i stede for måten jeg definerer lag på hvor jeg trenger å gå gjennom hele tabellen for å finne alle mulige spillere for et lag.

4.2.7.4 getReportStatOnlyAbsent()

De resterende funksjonene til serveren bygger mye på hva de 3 første. Denne funksjonen skal vise det motsatte av getReportStat(), altså hvem som mangler i motsetning til hvem som finnes. Så den gjøre det samme, men vil også gjøre en spørring for hvor mange er på laget, for så å kunne ta antall rapporter funnet og trekke fra totalt antall spillere og finner dermed hvor mange spillere som mangler. Koden for denne funksjonen er i vedlegg 1 om en skulle ønske en nærmere titt.

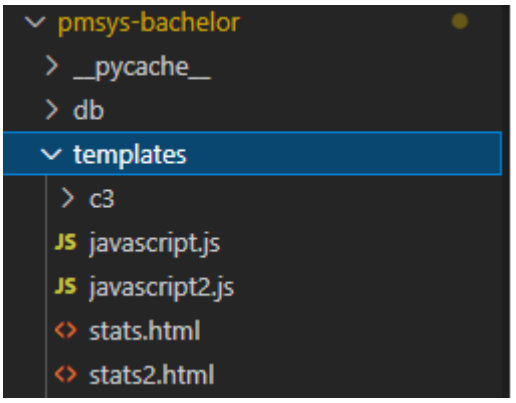
4.2.7.5 getAbsenceNames()

Denne funksjonen er en forenklet versjon av listOfAbsentPlayers(), den sender kun en liste med navn for en angitt dato, i stede for en liste med alle datoer i et tidsintervall. Grunnen til at det er to forskjellige er at listen for et helt tidsintervall som blir laget er ment for C3.js sin hover tooltip, mens denne funksjonen er ment for en onClick funksjon. Det er en stor forskjell her på hvordan C3.js håndterer de to forskjellige interaksjonene som er grunnen til at det ble laget en funksjon for vær, men dette vil bli gått i mer detalj under gjennomgangen av frontend-en. Koden for denne funksjonen kan bli funnet i vedlegg 2.

4.3 Frontend

Frontend-en består av 3 definerende delere. HTML filene som inneholder de statiske elementene, som beskrivelsen av versjonen og de nødvendige knappene for å interaktere med dataen, samt definierer en struktur for elementers plasering. JavaScript filene som kommuniserer med backend-en og genererer nye elementer for nettsiden. Og C3.js rammeverket som genererer grafene. I denne delen dekkes design valg, funksjonalitet, og bruken av C3.js.

4.3.1 Mappestruktur og koble filer sammen



Figur 19: Skjermbilde av frontend filene

På grunn av Flask sin måte å servere nettsider på må HTML filer legges i “templates” mappen. Og for å enkelt kunne linke dem sammen er JavaScript filene og C3.js biblioteket lagt i den samme mappen. Det er 2 forskjellige versjoner laget, så det er et sett med HTML og JavaScript per. De har mye av den samme koden i seg som sikkert kunne vært håndtert på en bedre måte enn å kopiere mellom dem, men det var også nok forskjeller til at det ga litt mening å skille dem fra hverandre. En mer optimal måte å strukturere dette på ville nok vært med en tredje JavaScript fil som holder funksjonene som er felles, som for eksempel en funksjon som oversetter auth0-tokens til et lesbart navn. Dette var dessverre ikke noe jeg fikk ryddet opp i.

For å koble alle filene og rammeverkene sammen for å kjøre siden hentes de enten rett fra rammeverket/bibliotekets nettside, eller ved bruk av Flask sin url_for() funksjon. Url_for() er en funksjon som mest er brukt til å kunne lage dynamiske linker, men etter min erfaring med Flask så virker det som den mest konsistente måte å lese inn filer generelt. Noen ganger om jeg skrev inn en vanlig <link href=”path”> for ville den ikke leses inn, mens ved bruk av url_for funksjonen gikk det være gang. Her er koden for å laste inn JavaScript dokumentet:

```
<!-- Javascript import -->
<script src="{{ url_for('static', filename='templates/javascript2.js') }}"></script>
```

Figur 20: Skjermbilde av kode for å lese inn filer til frontend

En viktig ting når filer leses inn i nettsiden er også hvor de er plassert. jQuery og Bootstrap rammeverkene må leses inn tidlig for å kunne ha tilgang til funksjonene og klassene de

tilbyr. Derfor ble disse lagt i <head> delen. JavaScript derimot er avhengig av å vite strukturen på siden for å kunne vite hvor den kan plassere elementene den genererer og ligger derfor på bånn av <body>.

4.3.2 HTML Struktur

Store deler av nettsiden blir generert av JavaScript, men HTML filene er nødvendig for å lage “skjelettet” til siden. HTML tag-ene definerer steder hvor innhold kan bli plassert som JavaScript kan så bruke for å kunne sette inn delene av siden den genererer.

Her defineres en div-boks kalt “intro-text” som inneholder en beskrivelse av siden, en div-boks, “chart”, som er avsatt plass til å generere grafer, og en div-boks, “textbox”, som er avsatt plass til å generere tabellen for dagens rapport status.

```
69  <body>
70    <center><h1>pmSys Concept - Player Report Count</h1></center>
71    <center><h2><a href="{{ url_for('version1') }}">Version 1</a> - <a href="{{ url_for('version2') }}">Version 2</a></h2></center>
72      <div class="container">
73        <div id="intro-text" class="card"><div class="card-body">Version 2 has a line design, this might be more visually pleasing w
74        <div id="everythingelse">
75 >        <form action="#">···
99          </form>
100
101         <div id="chart"></div>
102
103         <div id="textbox" class="row"  style="padding-top: 30px;">
104           <div id="left-text" class="mx-auto" style="text-align: center;"> <!-- class="col-sm-6" -->
105
106           </div>
107           <!--<div class="col-sm-1"></div>
108           <div id="right-text"> class="col-sm-6"
109
110           </div>-->
111         </div>
112       </div>
113     </div>
```

Figur 21: Skjermbilde av HTML strukturen på nettsiden

Div-boksen, “container”, som omringer alle de andre div-boksene er en Flexbox. Flex attributten hjelper med å gjøre siden responsiv. Med responsiv menes en side som kan fungere på flere enheter, som kan være viktig for når siden skal testes av brukerne. Selv om funksjonaliteten er laget for treningsportalen som er ment for å ses på en desktop er det realistisk at en brukere vil sjekke ut siden når som helst gjennom telefonen om de har noen minutter til overs.

```
.container{
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: center;
}
```

Figur 22: Skjermbilde av CSS nødvendig for å definere flexbox

Bootstrap er også blitt brukt i sammenheng med å gjøre siden mer responsiv. Bootstrap rammeverket har nyttige klasser som klassen “row” som hjelper å plassere innholdsområder

etter hverandre om det er plass. I versjon 1 av siden er venstre side holdt av til status tabellen, mens høyre side er til en onClick interaksjon. Bootstrap med kun å sette inn denne klassen hjelper å justere siden om nødvendig. Bootstrap har også klassen “mx-auto” som vil midtstille innholdet der den er brukt, noe som kan være utfordrende å løse, men igjen har bootstrap en enkel klasse som kan legges til. Bootstrap har også flere nyttige klasser brukt i løsningen for design av siden.

4.3.3 Bootstrap

I dette prosjektet var det ikke et krav om å ha et vakkert design på siden som ble laget, alt ForzaSys var ute etter var at funksjonaliteten så bra ut. Med hjelp av Bootstrap rammeverket er det derimot enkelt å kunne få et design som ikke ser forferdelig ut.

Bootstrap var blant annet brukt i løsningen til å lage knapper. Figur 23 viser default knapper i Chrome på venstre side og Bootstrap sin “btn” klasse på høyre side.

Figur 23: Skjermbilde av default knapper til venstre, Bootstrap klasse knapper til høyre

I HTML ser koden slik ut:

```
<input type="submit" class="btn btn-primary" id="show-rep-btn" value="Show Reported" onclick="showAttendance()" />
<input type="submit" class="btn btn-secondary" id="not-rep-btn" value="Show NOT Reported" onclick="showAbsence()" />
```

Figur 24: Skjermbilde av kode for å lage knappene

Ikke bare har Bootstrap et bedre design for knapper lett tilgjengelig, men også flere typer design for diverse bruksområder. Her er klassen “btn-primary” og “btn-secondary” brukt som hjelper å visualisere til brukeren hvilken innstilling som for øyeblikket er aktiv. Default knappene derimot har ingen måte å skille hverandre ut på. Det er selvfølgelig mulig å laget et eget design på knappene, men dette ville brukt lengre tid å utføre enn å bruke Bootstrap sine egenskaper.

Bootstrap var også brukt for design av status tabellen. Det er brukt nyttige klasser som thead-light for å lage en header rad som skiller seg ut fra resten av tabellen, og noen forhåndsdefinerte farge klasser som vil fylle ut bakgrunnen til en celle i tabellen kalt “table-danger”, en rød farge for å fremme en manglende rapport, og “table-success” for en rapportert rapport. Figur 25 viser et utsnitt av tabellen.

```
▼<table class="table table-bordered">
  ▶<thead class="thead-light">…</thead>
  ▶<tbody class="align-middle">…</tbody>
  ▼<tbody class="align-middle">
    ▼<tr>
        <td scope="row" align="left" style="padding-left: 12px;"> Lambert Peyton
        </td>
        <td class="table-danger">NONE</td>
        <td class="table-success">OK</td>
        <td class="table-success">OK</td>
      ▶<td class="ignore-btn-td">…</td>
     </tr>
   </tbody>
```

Figur 25: Skjermbilde av koden brukt for å designe dagens rapport tabell

Bootstrap var også brukt til et par andre småting som å ramme inn beskrivelse teksten på toppen av siden med hjelp av klassen “card” som gjorde at teksten så mer organisert ut på siden. Og klassen “custom-select” ble brukt til å lage en nedtrekksmeny for å matche designet til resten av siden.

4.3.4 Form Input

For å lese inn input ble det brukt en form-tag med alle parametrene som skulle til for å generere en graf. Koden for denne ser slik ut:

```
<form action="#">
  <div id="checkboxlist">
    <label for="report">Reports:</label>
    <input type="checkbox" onclick="checkall(this)" id="all" class="selectall" value="Select All" checked>
    <label for="all">Select All</label>
    <input type="checkbox" id="wellness" name="reports" value="wellness" checked>
    <label for="wellness">Wellness</label>
    <input type="checkbox" id="participation" name="reports" value="participation" checked>
    <label for="participation">Participation</label>
    <input type="checkbox" id="spre" name="reports" value="srpe" checked>
    <label for="spre">SRPE</label>
  </div>                                                                          Checkboxes
  <span style="font-size: 18px;"><label for="start_time">From: </label></span>
  <input type="date" id="start_time" name="start_time">
  <span style="font-size: 18px;"><label for="end_time">To: </label></span>
  <input type="date" id="end_time" name="end_time">                               Datovelger
  <span style="font-size: 18px;"><label for="team">Team: </label></span>
  <select class="custom-select" style="width: 100px;" name="teams" id="team">
    <option value="59b8149">Team 1</option>
    <option value="59b8148">Team 2</option>
    <option value="59b6e">Team 3</option>                                         Dropdown
  </select>
  <input type="submit" class="btn btn-primary" id="show-rep-btn" value="Show Reported" onclick="showAttendance()" />
  <input type="submit" class="btn btn-secondary" id="not-rep-btn" value="Show NOT Reported" onclick="showAbsence()" />  Knapper
</form>
```

Figur 26: Skjermbilde av koden for de forskjellige elementene brukt for bruker input

Checkbox-er for rapporttyper siden det er relevant og sjekke flere på en gang. Siden man er ute etter avvik fra grafen spiller det ingen rolle hvor mange rapporter man ser på en gang siden avvikene vil kunne være synlige.

Datovelgere for tidsintervallet laget ved bruk av ‘type=”date”’. Form sin datovelger er derimot ikke universal for alle nettlesere som gjør den litt problematisk, men dette vil bli gått litt mer gjennom i testing avsnitte.

For valg av lag ble en nedtrekksliste brukt. Å velge flere lag på en gang kan være nyttig om en trener har oversikt over mange lag. Det var mer et design valg fra min side å bare ha et lag av gangen, men den kunne selvfølgelig vært gjort om til en flervalgs.

Og til slutt en input av type “submit” for å aktivere en JavaScript funksjon som leser inn inputen i form-en.

4.3.5 JavaScript Funksjonene

JavaScript-et er delen av siden som genererer grafene og tabellene. Det er mye av det som skjer i denne filen som er hva ForzaSys er ute etter for utenom SQL-spørringene i backend-en så jeg vil her gå litt i detalj for noen designvalg og funksjoner siden det var krav om å lage grafene på en fin og ryddig måte. Funksjonene som genererer statistikken er displayStats() og displayArray(), begge disse blir utført når siden åpnes av Ready-funksjonen.

4.3.5.1 Ready-funksjon

En Ready-funksjon er en funksjon fra jQuery som kjøres når siden åpnes. Her vil den sette default inputs og generere grafen og tabellen for denne inputen. Den nederste linjen er nødvendig for å hindre nettsiden i å oppfriske seg når en “submit” knapp blir trykket på. Om den ikke er der vil nettsiden lastes opp på nytt og miste all input for hver gang noe endres.

```
$(function(){

    // This part initiallize the values of the date selectors.
    // It make the End Time be the current day, and the Start Time be 1 week ago.
    var dateControl = document.getElementById("start_time");
    var dateControl2 = document.getElementById("end_time");
    var today = new Date();
    lastWeek = new Date(today.getTime()  - 7 * 24 * 60 * 60 * 1000)
    var date = lastWeek.getFullYear()+'-'+String((lastWeek.getMonth()+1)).padStart(2, '0')+'-'+ String(lastWeek.getDate()).padStart(2, '0');
    var date2 = today.getFullYear()+'-'+String((today.getMonth()+1)).padStart(2, '0')+'-'+ String(today.getDate()).padStart(2, '0');

    dateControl.value = date;
    dateControl2.value = date2;

    // Because of testing these are override with more reasonable values.
    dateControl.value = "2021-05-01"
    dateControl2.value = "2021-05-31"

    // Initial fetch of stats from the last week.
    let reportInput = $("input[type=checkbox][name=reports]:checked").map(function(){
        return $(this).val();
    }).get();
    let starttimeInput = $("#start_time").val();
    let endtimeInput = $("#end_time").val();
    let teamInput = $("#team").val();

    // Because of testing reasons I override the input with static input for more reasonable opening view.
    // reportInput = "wellness,srpe";
    starttimeInput = "2021-05-01";
    endtimeInput = "2021-05-31";
    teamInput = "59b8149";

    url = "/api/stats/initv1?reports=" + reportInput + "&&start_time=" + starttimeInput + "&&end_time=" + endtimeInput +
        "&&team=" + teamInput + "&&ignore=" + JSON.parse(localStorage.getItem("ignored_players"));

    $.get(url, function (data){
        console.log("Fetching stats");
        console.log(data[0]);

        displayStats(data[0]);
        makeAbsenceList(data[2]);
        displayArray(data[1]);
    });

    // This line is to prevent whenever anything is submitted that the page wouldn't refresh and start over.
    $("form").submit(function() { return false; });

});
```

Figur 27: Skjermbilde av kode funksjonen som starter i det siden åpnes

Først vil den finne tidsintervallet grafen skal vise. Siden dataen jeg har tilgang til ikke er up-to-date med dagens data var det nødvendig å hardkode inn default innstillinger, men jeg lagde likevel kode for å kunne finne dagens dato og sette dette som default input siden det ville vært en mer realistisk brukstilfelle.

Deretter kjøres en funksjon for å lese inn hvilke rapporter som er merket av. Siden dette er noe som skjer hver oppstart av siden ville det ikke nødvendigvis være en grunn til å lese inn

denne infoen siden det vil alltid være den samme, men ingen grunn til å hardkode noe i tilfelle flere valg blir lagt til senere som da fører til å måtte gjøre endringer flere steder.

Resten av input verdiene blir så lest inn og en URL til et API endepunkt blir generert for disse verdiene. Her brukes jQuery sin AJAX funksjon til å gjøre et GET kall til serveren. Dataen serveren sender tilbake er lagret i “data” variabelen. “Data” inneholder 3 tabeller. Måten disse tabellene er generert på er beskrevet i “Flask Webserver og API” kapittelet. De blir så fordelt til funksjoner som håndterer dataen videre.

4.3.5.2 displayStats(data)

displayStats() funksjonen lager C3.js grafen. “Data” som funksjonen får sendt til seg er en tabell sendt fra backend-en som har strukturen vist i figur 28.

```
[
    [
        "wellness",
        [
            "Sat, 01 May 2021 00:00:00 GMT",
            1
        ]
    ],
    [
        "participation",
        [
            "Sat, 01 May 2021 00:00:00 GMT",
            1
        ]
    ],
    [
        "srpe",
        [
            "Sat, 01 May 2021 00:00:00 GMT",
            1
        ]
    ]
],
```

Figur 28: Skjermbilde av strukturen til data motatt fra backend

“Data” parameteren inneholder en tabell med en tabell for hver rapport type. Det første elementet i disse tabellene er navnet på rapporten, etterfulgt av tabeller som hver inneholder en dato og hvor mange manglende rapporter som var på den dagen (eller antall rapportert avhengig av hvilken innstilling på siden som er brukt).

Dataen i denne tabellen blir så omformatert til diverse variabler for å kunne brukes videre. C3.js har flere måter å lese inn data på. Måten jeg har brukt er formatet “columns” som krever strukturen vist i figur 29. Med strukturen sendt fra backend-en er det nært, men et par nestet for-løkker brukes for å gjør det riktig. Grunnen til at jeg ikke sender dataen i et format som er riktig med en gang er fordi det ville ha krevd å gjøre flere spørringer for å separere datoen fra resten av dataen. Og debug verktøyene jeg har tilgang til for frontend delen er enklere å bruke for å få oversikt over tabeller enn det jeg har i backend og derfor ga mer mening å gjøre om dataen i JavaScript i stedet for å gjøre de

```
columns: [
    ['navn_til_dataen1', 1, 1, 0, 0],
    ['navn_til_dataen2', 0, 2, 2, 0]
]
```

Rapport navn i dette tilfelle

dataen som vises på grafen (her ikke rapporterte)

Figur 29: Skjermbilde som demonstrerer formatet C3.js krever

samme funksjonene i Python. Jeg tror også at dette kan være fornuftig for å ikke legge mer press på serveren og heller ha disse løkkene på klienten sin side for å ikke ha serveren jobbe mer enn nødvendig.

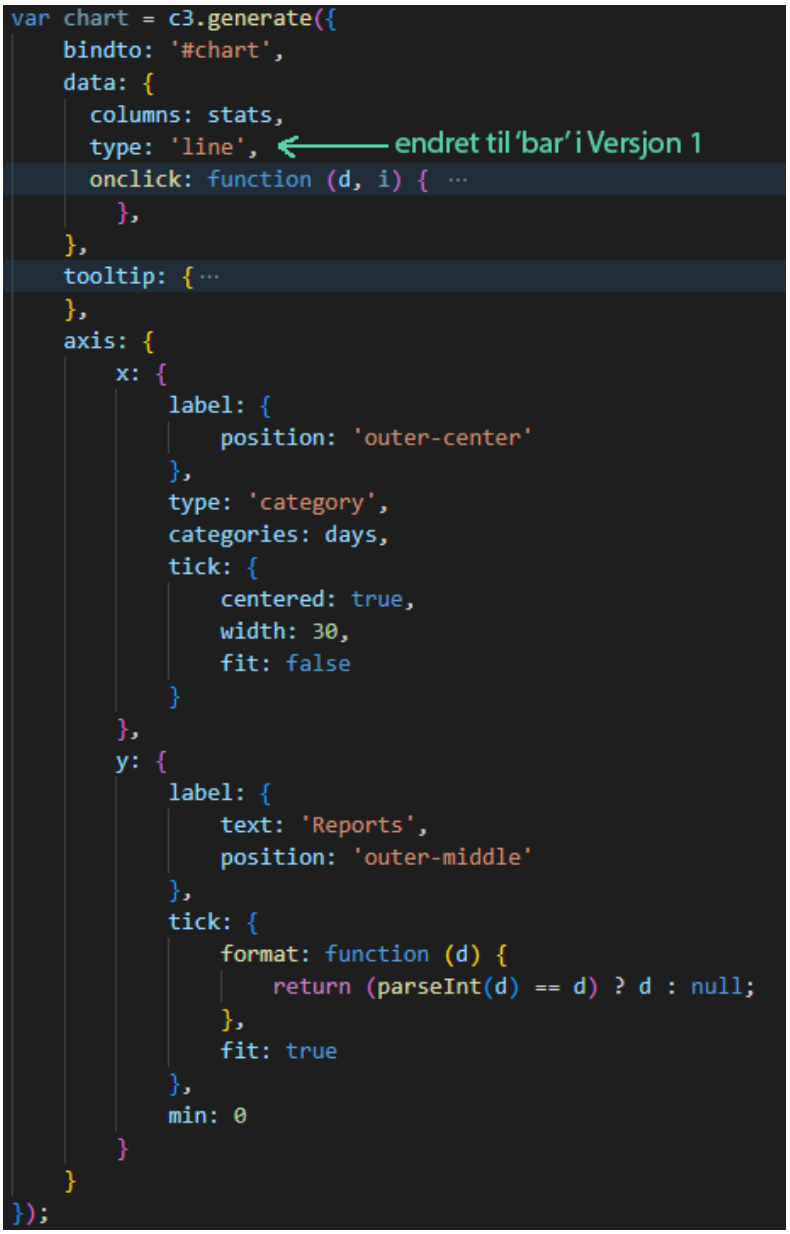
Koden for å skrive om dataen til det riktige formatet ser slik ut:

```
let stats = [];
let days = [];
let by_report_array = []
for (let r in data){
    by_report_array = []
    by_report_array.push(data[r][0]);
    data[r].splice(0, 1);
    for (let d in data[r]){
        by_report_array.push(data[r][d][1]);
        if(r == 0){
            datetime_o = new Date(data[r][d][0]);
            days.push(datetime_o.toDateString().split(' ').slice(1, 3).join(' '));
            // Removing the time of day and year to make it look clearner.
        }
    }
    stats.push(by_report_array)
}
```

Figur 30: Skjermbilde av koden for omformatering av dataen fra backend

Variablene jeg trenger for å lage grafen er “stats”, som er selve punktene på grafen, og “days” som er x-akse informasjonen. Stats er variabelen som trenger formatet beskrevet i figur 29. Den finner først navnet på rapporten som alltid ligger i det første elementet, deretter plukker ut kun dataen for hver dato. Om det er den første rapporten som sjekkes vil den også samle alle datoene i “days”, men som figur 28 viser er dato strengen som er sendt fra backend-en veldig lang, og dette vil ikke se bra ut på grafen. Derfor splittes den opp for å bare vise måned og dato for å kunne ta mindre plass. Med dataen endret til en C3.js kompatibel struktur kan grafen genereres.

Grafene lages med en C3.js funksjon som heter generate(). Generate() kan ta inn flere parametere. De brukt for denne løsningen er bindto, data, tooltip, og axis.




```
var chart = c3.generate({
    bindto: '#chart',
    data: {
      columns: stats,
      type: 'line',
      onclick: function (d, i) { ···
        },
    },
    tooltip: {···
    },
    axis: {
        x: {
            label: {
                position: 'outer-center'
            },
            type: 'category',
            categories: days,
            tick: {
                centered: true,
                width: 30,
                fit: false
            }
        },
        y: {
            label: {
                text: 'Reports',
                position: 'outer-middle'
            },
            tick: {
                format: function (d) {
                    return (parseInt(d) == d) ? d : null;
                },
                fit: true
            },
            min: 0
        }
    }
});
```

Figur 31: Skjermbilde av strukturen til C3.js sin generate funksjon

Bindto er enkelt nok bare en parameter som sier hvor grafen skal vises. Her spesifiseres "#chart" som betyr elementet med ID-en "chart", som var beskrevet i HTML delen som en div-boks satt av til denne grafen.

Data parameteren inneholder beskrivelsen av dataen og hvordan den skal håndteres. Dataen er lest inn med "columns" formatet beskrevet tidligere. "Type" er hvilken type graf som skal lages. "Line" spesifiserer et linjediagram, mens "bar" spesifiserer et søylediagram. Og tilslutt er det her man kan sette inn en onClick funksjon for når man interagerer med

dataen. Denne funksjonen er brukt i versjon 1 av siden hvor å klikke på en søyle i grafen vil skrive ut navn på alle som mangler rapporter for dagen trykket på. OnClick funksjonen til C3.js sender med noen nyttige parametere som kan brukes. Det sender blant annet med et JSON object med attributtene “id” som er navn på dataen, som vil være et rapportnavn i dette tilfellet, “value” som er verdien noden viser, og “index” som er nummeret som denne noden har i rekken med data. Man har derimot ikke den faktiske informasjonen til x-aksen, man har kun “index” som sier noe om den. Derfor må det regnes ut hvilken dato som egentlig blir spurt etter av å bruke informasjonen som er i datovelgeren. Denne vil alltid være datoen i den første noden av grafen og man kan dermed regne seg fram til hvilken dato som blir spurt etter ved å legge til indeks nummeret til den datoen. Den gjør så et API kall for å finne hvem som mangler for den datoen og rapporten trykken på og skriver dette ut på nettsiden. Koden for denne funksjonen finnes i vedlegg 3.

Den neste parameteren til generate() er tooltip. Default tooltips til C3.js er kun verdien til grafen. Dette er ikke nødvendigvis den mest relevante informasjonen man er ute etter, men heller navn på hvem som mangler i dataen. For å kunne redigere på tooltip-et brukes da tooltip parameteren, mer spesifikt “contents” attributten hvor en funksjon for å generere tooltips kan justeres. Denne funksjonen ble skrevet slik:

```
tooltip: {
    contents: function (d, defaultTitleFormat, defaultValueFormat, color){
        let index = parseInt(d[0].x);
        let aDate = listOfAbsence[index][0].split(' ').slice(1, 3).join(' ');

        let output = `<table class='${this.CLASS.tooltip}'><th style=\"width: 1%\">${aDate}</th><tbody>`;
        for (let i in listOfAbsence[index]){
            if (i == 0){
                continue;
            }
            output += `<tr><td style=\"display: flex; justify-content: flex-start; align-items: baseline; align-content: flex-start; \">
            <div class=\"toolcolor\" style=\"background-color:${color(d[i-1].id)}\"></div>  ` + listOfAbsence[index][i][0] + "</td></tr>"

            if (listOfAbsence[index][i][1][0] === undefined || listOfAbsence[index][i][1][0].lenght == 0){
                output += "<tr><td style=\"width: 1%\"><i>All reports in.</i></td></tr>";
            }
            for (let j in listOfAbsence[index][i][1]){
                output += "<tr><td style=\"width: 1%\"><b>" + convertName(listOfAbsence[index][i][1][j]) + "</b></td></tr>";
            }
        }
        output += `</tbody></table>`;
        return output;
    }
},
```

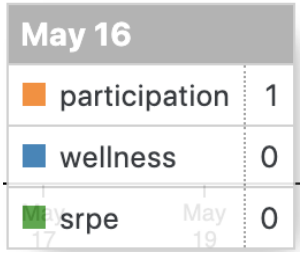
Figur 32: Skjermbilde av koden brukt for å generere tooltips

Igjen sender C3.js med et par hjelpsomme parametere, her brukes “d” som er JSON objektet med info om indeks nummer, verdi, rapportnavn, og “color” som er en funksjon som leser ut fargen på grafen.

Tooltip-et er bygd opp av en tabell. C3.js tilbyr klasser som den bruker selv for å generere sine tooltips. Dette gjør at ved å bruke disse klassene kan det enkelt lages et tooltip med

samme stil som vanlige C3.js tooltips som er fint for å kunne lage et konsistent design. Det er kun farger brukt i tooltip-et som er tilbudt, så resten må bygges opp fra bånn av.

Et C3.js vil med default innstillinger se ut som i figur 33. Den har en grå fane på toppen med informasjon om hvilken node som er i fokus. Listen under har en liten firkant med fargen av grafen, etterfulgt av navn på rapportene. Så for å få en lik fane på toppen ble C3.js sin "tooltip" klasse brukt. Fargen til grafene ble funnet med "color" funksjonen som fulgte med som en parameter nevnt tidligere. Boksen ble laget ved å lage en liten div-boks som er 10 pixler høy og bred, med en 1 pixel sort ramme. For å så fylle inn spillerne som mangler sjekkes det om det er noen spillere for den datoen som faktisk mangler. Om det ikke er det vil det skrives ut i tooltip-et "All-reports-in", og om det er noen som mangler vil en løkke skrive ut hvem det er.



Figur 33: Skjermbilde av C3.js default tooltip

```
.toolcolor{
  width: 10px;
  height: 10px;
  border: 1px solid rgba(0, 0, 0, .2);
}
```
Figur 34: Skjermbilde av CSS brukt for å gjenskape C3.js sin lille farge boks

Det er en ganske viktig forskjell mellom onClick funksjonen og et hover tooltip som er grunnen til hvilken data som er brukt. Forskjellen er hvor ofte grafen leser inn interaksjonen til brukeren. Et onClick kall skjer kun en gang i det øyeblikke du klikker, mens tooltip-et kalles hver eneste gang musa flyttes. For onClick funksjonen var det derfor ikke noe problem å gjøre et API kall for å kunne hente ut informasjonen som ble spurt etter, mens når dette ble forsøkt for tooltip-et krasjet serveren med en gang av at den mottok hundrevis av kall på en gang siden den ville sende en spørring for hver eneste pixel som musa flyttet på seg. Så for å kunne bruke tooltip til å vise data som ikke er tilgjengelig i grafen (siden grafen kun holder verdien den viser og indeksen sin) måtte det lages en global tabell i JavaScript-et som holder på denne informasjonen, listOfAbsence.

Den siste parameteren til generate() er axis. Med denne kan man justere på hva aksene skal vise og hvor det skal vises. X-aksen tar inn en tabell med dagen grafen skal vise. Om tidsintervallet er stort nok, vil x-aksen få flere punkter uten at grafen blir noe større. Dette gjør at når grafen går over et visst antall kommer teksten til å overlappe hverandre. Dette er forhindres ved bruk av "tick" attributtet ved å definere at det skal kun være tekst på x-aksen per 30 pixler, og med "fit: flase" vil den fjerne tekst som ikke vil kunne gi plass til minst 30 pixler ved siden av hverandre. For y-aksen måtte en justering til for å kunne fjerne unødvendig informasjon. C3.js vil default putte inn tall på y-aksen som går fra 0 til den maksimale verdien til grafen, og lage 10 punkter på y-aksen med verdier i mellom. Om man

med default innstillinger lager en graf for ikke rapporterte spillere, og den dataen har kun 1 som maksimal verdi, vil den dele y-aksen inn i 0, 0.1, 0.2, 0.3, osv. Men det er ikke noe som er “0.1 manglene spillere”, derfor er det en format funksjon i “tick” delen av y-aksen som leser inn nummeret på aksen, gjør den om til en integer, som bare vil holde heltall, og sjekke om heltallet er det samme tallet som grafen prøvde å vise. Om det ikke er det vil den returnere null, altså ingenting som skal vises.

```
tick: {
    format: function (d) {
        return (parseInt(d) == d) ? d : null;
    },
```

Figur 35: Skjermbilde av koden for å lage heltall i y-aksen

4.3.5.3 displayArray(data)

displayArray lager en tabell som viser informasjon hvilke rapporter hver enkelt spiller har registrert. Den har 2 versjoner hvor den ene vil skrive ut OK for de som har registrert en rapport, og lage en knapp med teksten “Notify” for der noen mangler en rapport. Mens den andre versjonen vil helle printe ut “None” for manglende rapporter, og ha en felles “Notify All” knapp for hver rapport type for å sende til flere av gangen.

```
[
    "wellness",
    "participation",
    "srpe"
],
[
    "auth0|59b814967091e711e630026d",
    1,
    1,
    1
],
[
    "auth0|59b814917091e711e6300265",
    1,
    1,
    1
],
```

Figur 36: Skjermbilde av dataen mottatt fra backend

Dataen denne funksjonen får inn er som vist i figur 36. Det første leddet i tabellen er hvilke rapporter som er valgt for å vises, mens de neste leddene har navnet på spilleren etterfulgt av enten 1 eller 0 for om de har registrert eller ikke i rekkefølgen angitt av det første leddet med rapportnavn.

Funksjonen går gjennom flere løkker for å kunne generere tabellen. For å lage headeren til tabellen leser den kun inn det første leddet som inneholder rapport navnene.

```
// Prints Headers, meaning the name of the reports.
output += "<th scope=\"col\" style=\"width: 160px;\">Name</th>"
for (let r in data[0]){
    let header = data[0][r].charAt(0).toUpperCase() + data[0][r].slice(1);
    if(header == "Srpe")
        header = header.toUpperCase();
    output += "<th scope=\"col\" style=\"width: 120px;\">" + header + "</th>";
}
output += "<th>Ignore</th></tr></thead>"
```

Figur 37: Skjermbilde av koden som genererer headere til tabellen

Den manipulerer strengen litt i tillegg for å lage store forbokstaver, eller i tilfelle en forkortelse gjøres hele til store bokstaver.

For versjon 2 er det en global variable som holder spillerne som ikke har registrert rapporter, notifyList. Det første leddet inneholder rapport navnene så den lagrer det i sitt første ledd også, og lager en tom tabell for hver rapport type.

```
id = 0;
notifyList = [];
for (let d in data){        // Per player
    let notifyNeeded = [];
    // skipping headers.
    if (d == 0){
        console.log(data)
        notifyList.push(data[0]);
        for(let a in data[0]){
            notifyList.push([]);
        }
        continue;
    }
    output += "<tbody class=\"align-middle\"><tr>";
    for (let i in data[d]){     // Per Report that player has.
        id += 1;
        if (data[d][i] == 1)
            output += "<td class=\"table-success\">OK</td>";
        else if (data[d][i] == 0){
            notifyList[i].push(data[d][0]);
            output += "<td class=\"table-danger\">NONE</td>";
        }
        else
            output += "<td scope=\"row\" align=\"left\" style=\"padding-left: 12px;\"> "
                    + convertName(data[d][i]) + "</td>";
    }
    output += "<td class=\"ignore-btn-td\"><button id=\""+ id
        + "\" class=\"btn btn-info mx-auto\" value=\"ignore\" onclick=\"ignorePlayer(\'"
        + data[d][0] + "\', " + id + ")\">Ignore</button></td></tr>";

}
output += "<tr><td></td>"
```

Figur 38: Skjermbilde av koden i versjon 2 for generering av celler i rapporterings tabellen

Den vil så gå gjennom hver spiller i dataen, om dataen leses inn som 1 så lager den en rute med “OK”, om den sier 0 lager den en rute med “None”, og om det ikke er noen av delene så er det et navn som blir lest inn og din konverterer auth0-token-en til et navn som kan skrives ut. Den vil også på slutten av være rad legges inn en “Ignore” knapp. Funksjonen denne har krever 2 parametere, den ene er navnet på spilleren som skal ignoreres, mens den andre en ID som denne knappen har. Dette er for å kunne lage en funksjon som kan finne denne knappen å toggle den om til en “Unignore” knapp om den er klikket på.

Og helt til slutt lages “Notify All” knappene som kun trenger en rapport ID for å kunne sjekke mot notifyList-en.

```
// "Notify All" Buttons at the bottom
for (let r in data[0]){
    output += "<td><button id=\"repID" + r
    + "\" class=\"btn btn-danger mx-auto\" value=\"notify\" onclick=\"notifyAll("
    + r + ")\">Notify All</button></td>"
}

output += "<td></td></tr></tbody></table>";
$("#left-text").html(output);
```

Figur 39: Skjermbilde av koden i versjon 2 for å generere en knapp til å varsle flere spillere

For versjon 1 er det i stedet for en enkel knapp på bånn med “Notify All” en knapp for hver manglende rapport status. Så hvor funksjonen over i versjon 2 bare skriver ut “None” og legger til et navn i en tabell, er koden til versjon 1 slik:

```
else if (data[d][i] == 0)
    output += "<td class=\"notify-btn-td table-danger\"><button id=\""+ d + i
    + "\" class=\"btn btn-dark mx-auto\" value=\"notify\" onclick=\"notify(\'"
    + data[d][0] + "\', \'" + data[0][i-1] + "\', "+ d + i + ")\">NOTIFY!</button></td>";
```

Figur 40: Skjermbilde av koden i versjon 1 for å generere en knapp til å varsle 1 spiller

Det som var utfordrende her var å holde orden på hvilken knapp som ble trykket på og være sikker på hele beskrivelsen av hva og hvem som skal varsles legges til rette for senere. Det kan hende at det var en smart måte å bruke .this() funksjonen for å finne knapper, men måten jeg endte opp med å løse dette på var heller å gi hver knapp en ID basert på hvilken x og y koordinat knappen hadde i tabellen. Det var nødvendig å kunne få tak i hvert knapp-element for å kunne endre knappen sin status fra “Notify” til “Notified” om den ble trykke for at brukeren skulle kunne vite at en varsling ble sendt.

Resten av funksjonene er små funksjoner som støtter rundt grafen og tabellen, eller som kjører disse funksjonene gjennom andre interaksjoner på siden. De har dukket opp noen ganger i beskrivelsen av genereringen av grafen og tabellen, men her er en liten gjennomgang av de også.

4.3.5.4 showAttendance()

Dette er funksjonen som kjøres når man trykker på “Show Reported” knappen. Den vil gjøre knappen blå og “Show NOT Reported” knappen grå for å indikere hvilken statistikk som vises på grafen. Dette gjøres med å legge til og fjerne Bootstrap klasser for knappene. Informasjonen leses så fra form-skjemaet på siden for å lage et API-endepunkt som sender tilbake data om hvor mange som har rapportert.Dataen sendes så videre til displayStats(data). Koden finnes i vedlegg 4.

4.3.5.5 showAbsence()

Denne funksjonen er nesten helt lik showAttendance(), men for knappen “Show NOT Reported”. Den vil sette “Show Reported” grå, og “NOT Reported” blå, og API kallet som gjøres vil i stedet returnere dataen for hvor mange som ikke har rapportert.

4.3.5.6 makeAbsenceList(data)

Denne funksjonen blir kalt hver gang en graf lages, men alt som skjer er at en tabell som for hver dag i et tidsintervall vil ha en tabell for hver rapport om hvem som mangler for disse rapportene. Denne tabellen kan overføres med en gang fra formatet sendt fra API-en, men den leses inn over flere funksjoner så det virket fornuftig å lage en egen funksjon for å lese den inn i tilfelle det ble nødvendig å endre på formatet, eller om den en gang ville blitt sendt inn som et response objekt og ville da måtte omformateres.

4.3.5.7 convertName(autotokenid)

Denne funksjonen er en lang liste med if-statements for hver auth0-token. Denne er kun for å kunne ha mer realistisk fremvisning av dataen enn bare en mange tilfeldige strenger. Om token-en ikke samsvarer med noen av de foreslåtte skrives token-en ut i konsollen for å kunne se hva som gikk feil. Koden for denne finnes i vedlegg 5.

4.3.5.8 checkTeam(teamid)

Dette er en liten if-statement som omskriver lag koden om til noe mer leselig. Som nevnt tidligere er lagene definert ut fra et mønster i auth0 token-ene. Den vil returnere en oversetter disse som “Team 1”, “Team 2” eller “Team 3”.

4.3.5.9 ignorePlayer(name, id)

Når en “Ignore” knapp for en spiller er trykket på vil denne funksjonen legge til navnet sent med inn i en liste for ignorerte spillere og endre knappen fra å si “Ignore” til “Unignore” ved å finnet knappen gjennom ID-en som ble send med. Om knappen var satt til “Unignore” vil derimot det motsatte skje, spilleren vil ble fjernet fra ignorert listen og knappen gjøres tilbake

til “Ignore”. Etter at dette er utført vil grafen oppdatere seg, for grunnen til å ignorere en spiller er å rydde opp infoen i grafen så naturlig nok skal den oppdateres. Koden for denne finnes i Vedlegg 6.

4.3.5.10 notify(name, report, id)

Dette er varslings funksjonen for versjon 1. Her sendes varsler til en spiller av gangen. Dette er gjort med alert() funksjonen med tekst inni om navn på den som skal bli varslet og for hvilken rapport, som ligger i parametrene sendt med. Det er selvfølgelig ingen varsling sendt til noen spillere, men det virket nødvendig med en alert() for å kunne gi en feedback til brukeren om at noe skjedde. Det virket veldig unaturlig å trykke på en knapp og alt den gjorde var å endres til “Notified”, når du egentlig ikke vet om noen relevant data var vurdert i det trykket. Koden er lagt ved i vedlegg 7.

4.3.5.11 notifyAll(repID)

Dette er varslings funksjonen for versjon 2. Her sendes varslinger for alle relevante spillere for en rapport type. Den gjør mye det samme som notify(), men i stede for å få inn navnet gjennom en parameter sendes det med en ID som spesifiserer hvor den skal sjekke i den globale listen notifyList. Denne listen inneholder alle spillerne som trengs å sendes varslinger sortert per rapport type. Den går gjennom en løkke for å sjekke innholdet til notifyList[repID+1] for å finne spillerne som skal varsles og bruker convertName() for å få finere output. Det må legges til +1 for i det første elementet finner man navnet på rapportene og deretter etterfulgt av navnene som letes etter. Det blir igjen brukt en alert() for å gi feedback to brukeren. Koden er i vedlegg 8.

4.3.5.12 Checkbox “Select All” funksjoner

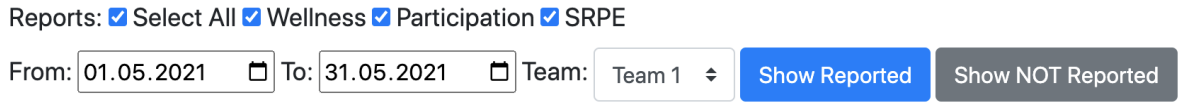
Det er 2 funksjoner som trengs for å kunne lage en “Select All” checkbox. Funksjonen for å sjekke alle boksene når den blir valgt er en løkke som går gjennom alle elementer med name-tag-en “reports” og setter alle disse til .checked. For at en “Select All” skal gi mening må denne også fjerne sin egen hake nå noe fjernes, for da er naturlig nok ikke alle valg. Dette gjøres ved at når en checkbox er trykket på vil den sjekke om den ble trykket av. Om den ble trykket av vil den fjerne haken fra “Select All”. Koden er i vedlegg 9.

4.4 Brukerguide til nettsiden

I dette avsnittet blir det utført en gjennomgang av hvordan man bruker nettsiden. Den består av 2 sider, Version 1 og Version 2. Gjennom brukerguiden vil det være små utklipp fra siden for å vise elementene. Et skjermbilde av siden i sin helhet kan ses i vedlegg 10 for versjon 1 og i vedlegg 11 for versjon 2.

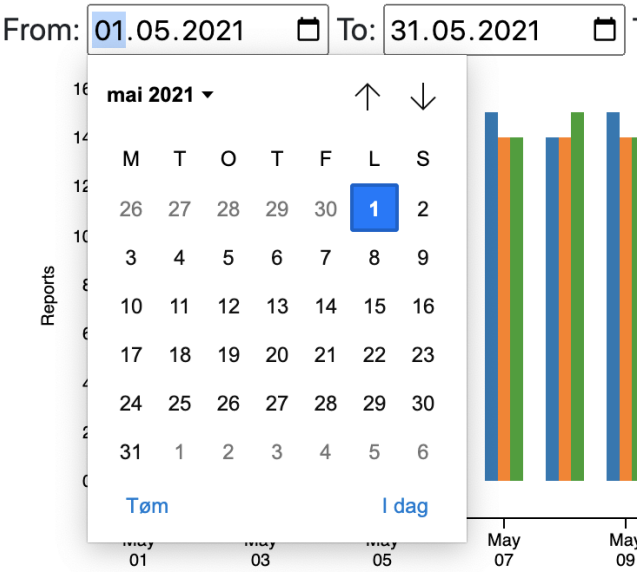
På toppen velger man mellom forskjellige versjoner av løsningen. Versjonene har noen små endringer mellom seg, og disse er beskrevet i boksen under tittlen. I versjon 1 er det også foreslått her å sette en av spilleren på ignore for å gjøre dataen litt renere.

Når siden åpnes er det allerede en graf vist med default innstillinger. Disse innstillingene kan endres ved bruk av de forskjellige elementene over grafen.



Figur 41: Skjermbilde av elementene på siden til bruker input

Man kan så huke av hvilke rapporter man har lyst å vise på grafen. Select All knappen vil slå på alle rapport typene om noen ikke er allerede valgt. Om alle er valg vil den fjerne alle rapportene fra å være markert.



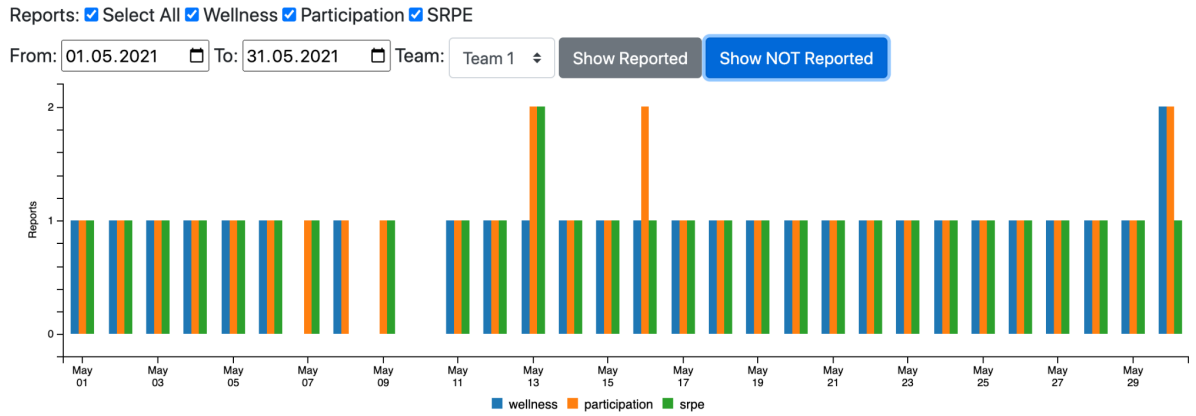
Figur 42: Skjermbilde av datovelgeren i chrome

For bestemme tidsintervallet som grafen skal vise brukes to datovelgere. Når man klikker på en datovelger vil den vise en kalender der man kan velge en hvilken som helst dato. Om man ønsker kan man også skrive inn manuelt i stedet for å bruke datovelger brukergrensesnittet. Datasette har derimot kun data mellom 1. mai, og 25. oktober 2021. Alle datoer valgt utenfor dette tidsintervallet vil ikke vises på grafen.

Deretter velges et lag fra en nedtrekksliste. Dette datasettet har 3 definerte lag man kan velge blant.

Tilslutt er det 2 knapper merket “Show Reported” og “Show NOT Reported”. Disse to knappene avgjør hvilken type statistikk som vises på grafen. Ved å trykke “Show Reported” vil den lage en graf som viser hvor mange spillere som rapporterer hver dag. Om man trykker på “Show NOT Reported” vil den kun vise hvor mange som ikke har rapportert. Versjon 1 viser dataen som et søylediagram. For hver rapport som er er markert vil en ny søyle bli plassert ved siden av de andre for hver dag i tidsintervallet.

Versjon 1 viser et søylediagram for dataen som er spurt etter vist i figur 43a.

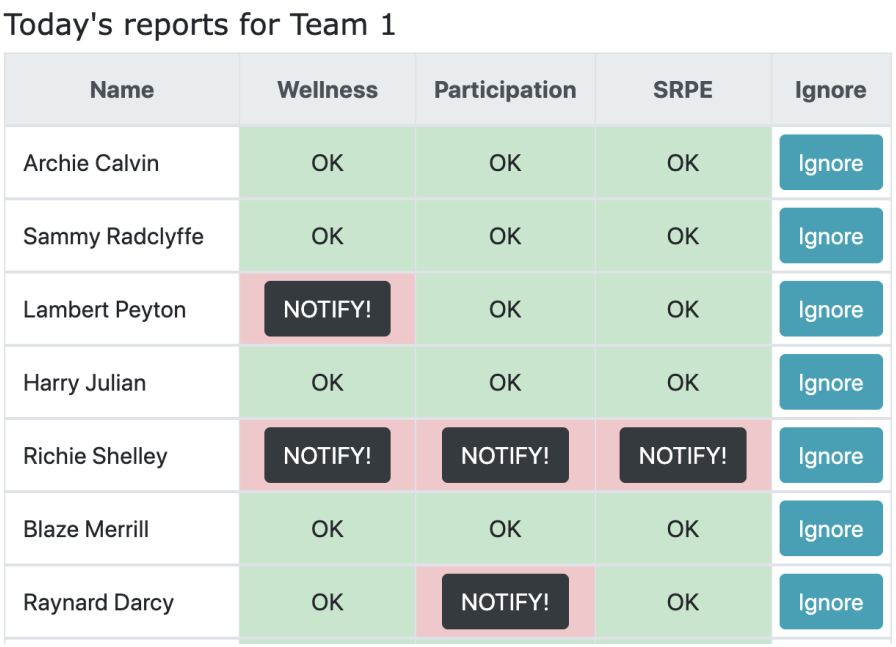


Figur 43a: Skjermbilde av søylediagrammet i versjon 1

I versjon 1 vil et tooltip med verdien for hva grafen viser bli skrevet ut nå man hoverer over en søyle. Om man klikker på søylen vil en liste med navn på de som ikke hadde rapportert på den dagen for rapport typen til søylen klikket på vises under grafen.

Figur 43b: Skjermbilde av tooltip og navn utskriften i versjon 1

Under grafen finner man en tabell som viser alle spillerne på Team 1 og deres rapporterings status for en spesifikk dag. Dette vil realistisk sett vært dagens dato for å kunne ha en oversikt om hvilke rapporter som ikke er registrert og kan deretter varsles om. Dette datasettet har derimot ikke dagens registreringer og viser derfor en bestemt dag med relevant data å se over.

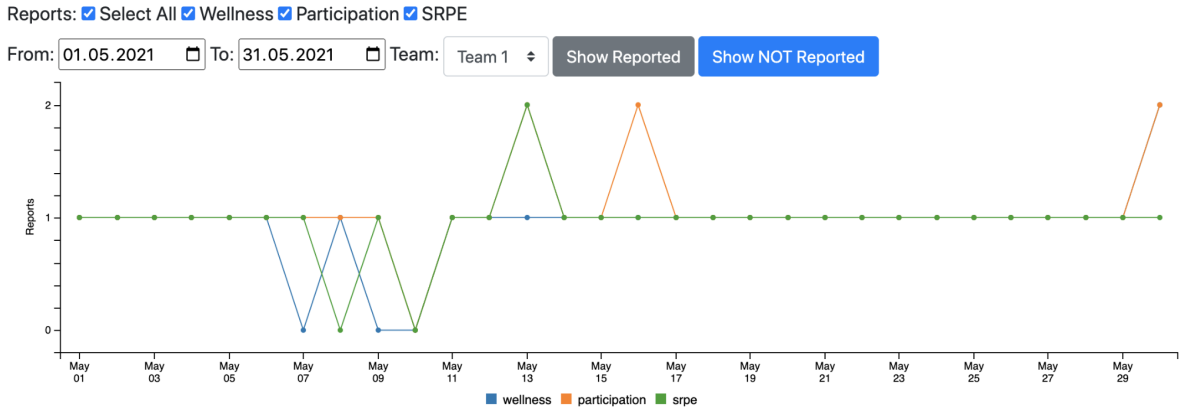


Figur 44: Skjermbilde av dagens rapport tabell i versjon 1

Her kan man trykke på en “NOTIFY!” knapp for å sende ut en varsling. Dette er selvfølgelig bare en test, og ingen varsel blir egentlig sendt, men i stedet blir det skrevet ut en melding på skjermen med infoen relatert til hvilken knapp som ble trykket på.

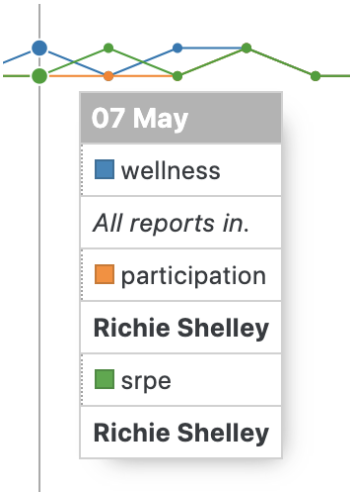
Ved å trykke på en “Ignore” knappen på høyre side av tabellen vil spilleren på raden til knappen bli ignorert i grafen ovenfor. Etter å ha trykket på “Ignore” vil den oppdatere grafen med en gang, og da uten spilleren ignorert, og knappen vil endres til “Unignore”. Denne kan så bli trykket på igjen for å legge spilleren tilbake i datasettet.

Videre i versjon 2 av siden vil den ha de samme elementene, men noen små endringer. Den mest åpenbare endringen er at grafen er nå et linjediagram som vist i figur 45a.



Figur 45a: Skjermbilde av linjediagrammet i versjon 2

Denne grafen kan ikke klikkes på for å få skrevet ut mer informasjon, men denne informasjonen er i stedet flyttet inn i tooltip-et. Tooltip-et som dukker opp ved å hovere musen over en node vil nå vise en liste med alle spillere som ikke har rapportert for den dagen. Om alle har rapportert for denne dagen vil den fylle inn med “All reports in.”.



Figur 45b: Skjermbilde av tooltip i versjon 2

Tabellen under grafen har nå en knapp på bånn av hver kolonne for å varsle alle spillere som mangler å registrere rapport typen klikket på i motsetning til å varsle hver enkelt spiller i versjon 1.

Today's reports for Team 1

Name	Wellness	Participation	SRPE	Ignore
Lambert Peyton	NONE	OK	OK	Ignore
Harry Julian	OK	OK	OK	Ignore
Richie Shelley	NONE	NONE	NONE	Unignore
Blaze Merrill	OK	OK	OK	Ignore
Raynard Darcy	OK	NONE	OK	Ignore
	Notify All	Notify All	Notify All	

Figur 46: Skjermbilde av dagens rapport tabell i versjon 2

4.5 Design valg

Designet på selve siden var ikke et fokus i dette prosjektet. Designet av statistikken som blir vist derimot var en viktig del av prosjektet. Det ble to forskjellige versjoner laget for å kunne teste ut flere design av grafene for å kunne få et bedre perspektiv. Siden design av data visualiseringen var et av de mest avgjørende punktene i prosjektet var dette diskutert mye om i møtene med ForzaSys for å kunne komme til den beste avsluttningen. Her beskrives de designene som ble brukt i løsningen, men en diskusjon rundt hvor bra de fungerte er gjort i refleksjonsdelen i slutten av produktdokumentasjonen.

4.5.1 Søylediagram

Søylediagram bruker horisontale eller vertikale søyler for å vise en diskret numerisk sammenligning over flere kategorier, og svarer på spørsmålet “hvor mange” (Datavizcatalogue, u.å.). Dette kan derfor være en fornuftig måte å vise dataen på siden man ønsker å kunne sammenligne dagene mot hverandre for å kunne lettere få øye på hvor avvik dukker opp, og det relevante spørsmålet er nøyaktig “hvor mange”. I dette formålet kan også betegnes som et “histogram” over “søylediagram”, men det er litt snakk om begge deler siden det både er søyle visualisering over et tidsintervall (histogram), men også flere rapporttyper på en gang. Det blir kanskje litt mer riktig å klassifisere dette som histogram, men jeg har i denne rapporten holdt meg til å referere til dette som søylediagram.

4.5.2 Linjediagram

Linjediagram er brukt til å vise kvantitative verdier over et tidsintervall, og er mest brukt for å vise trender og analyse av data som endrer seg over tid (Datavizcatalogue, u.å.). Dette er igjen noe som kan brukes til dette formålet. Det som er viktig å vise er når det kommer avvik i dataen og i et linjediagram vil det vises som en klar nedover/oppoverbakke i grafen hvor dataen endret seg fra “normalen”.

4.5.3 Tabellene

For designet av tabellene brukt til å lage en oversikt over dagens rapport var det mest eksperimentert med i hvilken grad man skulle gi muligheter for varslinger til spillerne. Jeg var ganske fornøyd med det første utkastet med en vanlig fargekoding for rapport statusen. Om cellen har en grønn OK er den levert, mens om den har en rød NONE er den ikke. Denne form for identifisering av gode og dårlige utfall er ganske normalt og det virket naturlig å bruke de i denne delen av løsningen.

4.5.4 Responsiv

Det var som nevnt ikke et krav om at siden i seg selv skulle være fin, men et responsivt design av siden er fortsatt viktig å vurdere under utviklingen. Det er umulig å forutse hvilken plattform en bruker ville velge for å kjøre siden min. Derfor følte jeg det nødvendig å legge til rette for at den skal kunne fungere på flere enheter. Det er ikke mange elementer på siden som gjør det lettvint å lage en responsiv struktur. Om siden ikke hadde vært responsiv kunne dette ha negativ effekt på brukerne når siden testes og kanskje medføre at funksjonaliteten virker mer uhåndterlig enn det den kanskje er.

4.6 Monitoring og Debugging

Det dukker alltid opp bugs mens man jobber med programmering. I denne delen skal jeg beskrive alle metodene og verktøyene som ble brukt i denne prosessen for å fange opp alle feil som ble gjort underveis.

4.6.1 Backend

Flask tilbyr veldig oversiktlig monitorering i konsollvinduet programmet blir kjørt i. Den vil vise all trafikk som kommer til serveren og vil sende responser både om et API kall ble vellykket eller ikke. Jeg kan også legge inn print funksjoner i Flask funksjonene for å kunne skrive ut noe til konsollen for å se hvilken data som kommer inn og hvilke data som blir sendt tilbake. Mine konsoll meldinger er mest for å sjekke logikken i koden min, mens Flask vil fange opp problemer knyttet til nettverksoverføringer.

Her er et eksempel på en vellykket spørring mot API-et. Dette er et kall som spør etter listen for manglende rapporter mellom 1 mai og 4 mai.

```
Start Datetime object: 2021-05-01
Input Reports: ['wellness', 'participation', 'srpe']
Here is All Data:
[['wellness', 'participation', 'srpe'], ['auth0|59b814967091e711e630026d', 1, 1, 1], ['auth0|59b814
917091e711e6300265', 1, 1, 1], ['auth0|59b814917091e711e6300266', 1, 1, 1], ['auth0|59b814907091e71
1e6300264', 1, 1, 1], ['auth0|59b814937091e711e6300268', 1, 1, 1], ['auth0|59b814947091e711e630026a
', 1, 1, 1], ['auth0|59b81497cbb73b1e63a1cbc6', 1, 1, 1], ['auth0|59b81492cbb73b1e63a1cbc5', 1, 1,
1], ['auth0|59b81494c71350685f02daa6', 1, 1, 1], ['auth0|59b81499cbb73b1e63a1cbc8', 0, 0, 0], ['aut
h0|59b814937091e711e6300269', 1, 1, 1], ['auth0|59b814967091e711e630026c', 1, 1, 1], ['auth0|59b814
95c71350685f02daa7', 1, 1, 1], ['auth0|59b81497cbb73b1e63a1cbc7', 1, 1, 1], ['auth0|59b81490cbb73b1
e63a1cbc4', 1, 1, 1]]
172.18.0.1 - - [23/May/2022 18:04:20] "GET /api/stat/onlyabsent?reports=wellness,participation,srpe
&&start_time=2021-05-01&&end_time=2021-05-04&&team=59b8149&&ignore=au HTTP/1.1" 200 -
```

Figur 47: Skjermbilde av konsollen som kjører Flask under et vellykket API kall

Jeg har mine egne konsoll meldinger først med informasjon om noen av parametrene som blir sendt med for å sjekke at alle blir lest inn riktig og omformatert på riktig måte. Det printes også ut hva som sendes tilbake til klienten i noen funksjoner for å se at jeg sender over den informasjonen jeg forventes at skal sendes. Helt tilslutt printer Flask ut HTTP response

status koden, som også blir sendt til klienten. I dette tilfelle var API kallet vellykket som er indikert med en suksess koden 200.

I figur 48 vises en spørring som ikke er vellykket. Det samme kallet blir gjort, men i HTTPie uten å gjøre riktig syntaks. HTTPie krever at URL-en skrives inn i anførselstegn (“”). Om man utelater disse vil det fortsatt gjøres et API kall, men bare den første parameteren blir registrert .

```
Printing ignore input: None
[2022-05-23 18:05:46,660] ERROR in app: Exception on /api/stat/onlyabsent [GET]
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 2077, in wsgi_app
    response = self.full_dispatch_request()
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 1525, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 1523, in full_dispatch_request
    rv = self.dispatch_request()
  File "/usr/local/lib/python3.10/site-packages/flask/app.py", line 1509, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**req.view_args)
  File "/app/api.py", line 87, in getAbsenceReport
    total_returned.append(getReportStatOnlyAbsent(1))
  File "/app/api.py", line 99, in getReportStatOnlyAbsent
    ignored = args5.split(",")
AttributeError: 'NoneType' object has no attribute 'split'
172.18.0.1 - - [23/May/2022 18:05:46] "GET /api/stat/onlyabsent?reports=wellness,participation,srpe
 HTTP/1.1" 500 -
```

Figur 48: Skjermbilde av konsollen som kjører Flask under et mislykket API kall

Den vil da gjøre konsoll meldingene så langt den kommer, som her ble det skrevet ut at datatypen til ignore listen var “none”. Og blant de nederste linjene vil det også bli nevnt hvor i Flask applikasjonen feilen skjer, som her er linje 99 der den ikke klarer å omformatere ignore listen siden den variablene er av “NoneType”, eller med andre ord rett og slett ikke finnes. Den vil så sende status koden 500 som står for “Internal Server Error” som er en generell response for at server funksjonen crashet i løpet av kallet.

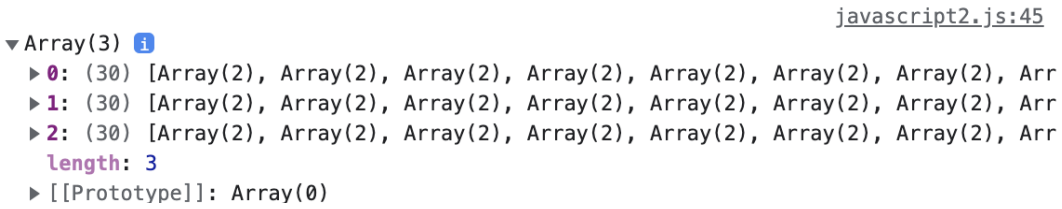
Programmet i seg selv der i mot crasher ikke i tilfelle et API kall ikke funker. I en build jeg hadde på serveren en periode funket ikke “Show Reported” knappen pga en feil i JavaScript hvor versjon 1 ikke var oppdatert til for å bruke et nytt format knyttet til versjon 2. Uansett hvor mange ganger knappen ble trykket på vil ikke siden gå ned.

4.6.2 Frontend

For frontend delen av siden gikk mye av debugging ut på å fin pusse på struktureringen av dataen som skulle vises. C3.js har et spesifikt format som skal leses inn. Tabellen må ha riktig begynelse og avsluttning av tags rundt dataen ment for cellen den skal puttes inn i for å bli presentert riktig. Og som nevnt i et tidligere kapittel er ikke dataen ferdig strukturert for frontend. Det gjøres derfor mange løkker og mye logikk kan gå i surr i en løkke.

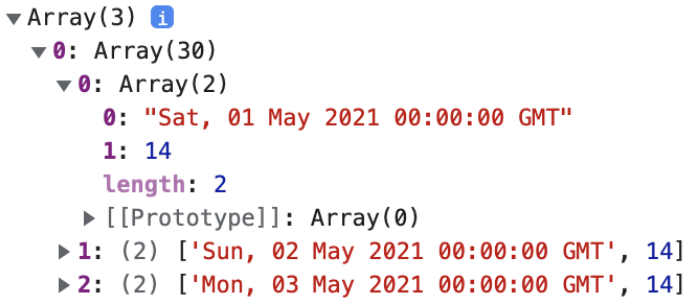
For disse utfordingene har Chrome DevTools mange nyttige funksjoner som gjør det enklere å få oversikt over hvilken data som jobbes med og for å finne feil i kode logikken.

Den ene funksjonaliteten deres er å kunne se konsollen til siden på en veldig detaljert måte. I JavaScript kan man bruke koden “console.log()” for sende noe til konsollen. Dette kan være hva som helst, enkle setninger, tabeller, objekter, etc. Etter et API kall kan det være nyttig å se om man fikk tilbake den dataen man forventet og også bekrefte strukturen den har. I figur 49 vises en konsoll utskrift av dataen som kommer inn når siden blir åpnet.



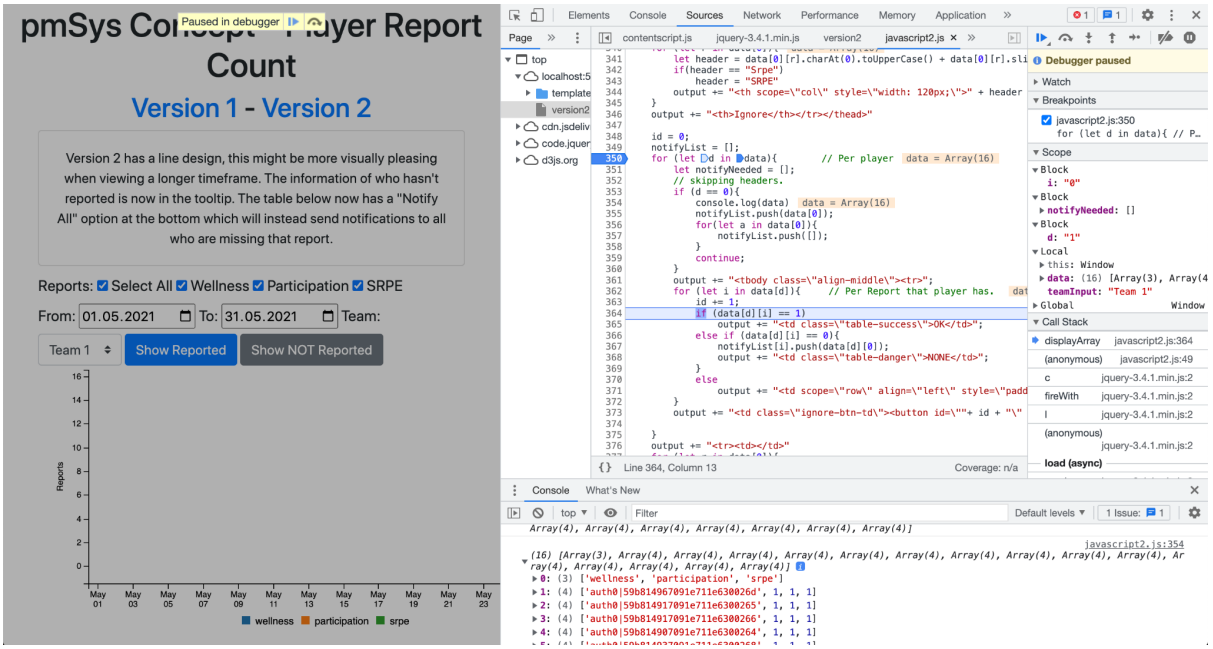
Figur 49: Skjermbilde for konsoll outputen sett i Chrome DevTools for dataen ved oppstart

Når mye data blir sendt på en gang vil den slå sammen tabellene, men ved å trykke på pilen ved siden av får man se alt innholdet som vist i figur 50. Den vil også skrive ut i høyre hjørne hvor i koden utskriften ble laget. Med så mange nestede tabeller som dataen min er blitt sortert inn er det veldig nyttig å kunne gå inn i tabellen og se hvordan all dataen ligger i forhold til hverandre. En nyttig egenskap er indeks numrene. For å få tak i datoen 1. mai står i linje 4 i figur 50 kan man se på alle indeks tallene som leder til den cellen, i dette eksempelet ville det blitt *navn_på_array*[0][0][0].



Figur 50: Skjermbilde av celler åpnet for mer detaljer i Chrome DevTools

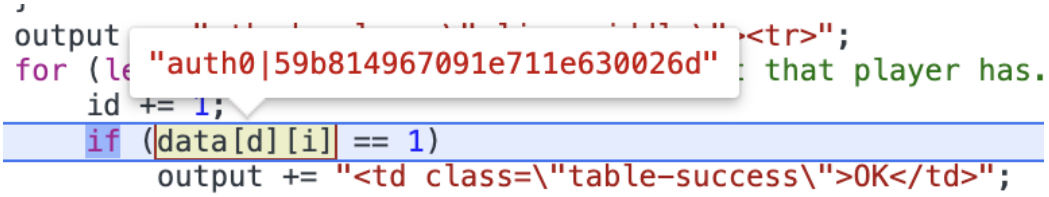
Den andre funksjonaliteten til DevTools brukt mye i debugging er breakpoints. I figur 51 vises et eksempel på et breakpoint i kodelinje 350 hvor generering av status tabellen skjer.



Figur 51: Skjermbilde av nettleseren under effekt av DevTools sitt breakpoint

Her kan man kjøre koden linje for linje ved å klikke “Step” øverst i høyre hjørne (eller taste F9).

Ved å hovere over en variabel vil man kunne sjekke hva den inneholder i det steget av koden. Som hvis man vil sjekke hvilken data som var vurdert om en if-statement ikke gikk gjennom vist i figur 52.



Figur 52: Skjermbilde av konsollen til DevTools når man hoverer over en variabel

Dette er veldig nyttig for å kunne enkelt feilsøke seg til problemene som oppstår når du får en HTML output som ikke gir mening. Ved å kjøre steg for steg på denne måte finner man nøyaktig ut hvor noe gikk galt, og man kan inspisere variabler nøye for å få mer innsikt.

4.6.3 Logging

Det er ingen permanent logging som foregår. Siden jeg kjører løsningen stort sett lokalt så har jeg alle konsollene åpne når noen som helst debugging er utført. Derfor var det ikke

noen prioritering for meg å lage veldig sofistikert loggføring hvor man overfører feilmeldingene Flask skriver ut til en egen loggfil for å kunne se over alle feilene ved en senere tid. Det hadde sikkert vært nyttig med mer omfattende loggføring med tanke på når siden ble lagt ut på nett, men det var ikke noe jeg fikk implementert. Siden min gikk ned et par ganger hvor om jeg hadde drevet med loggføring kunne jeg kanskje ha fanget opp hva som hadde gått galt.

5 Testing

Testing av applikasjonen er en avgjørende del av prosessen for å kunne være sikker på at det som blir produsert både funker og har kvaliteten man er ute etter. Mye av testing i forbindelse med koden var dekket i Debugging kapittelet, men her skal prosessen i seg selv beskrives litt nærmere. I denne delen blir det også gått over brukertesten som ble utført. Uheldigvis stoppet serveren å funke rett før brukertesten ble utgitt og brukerne måtte derfor forholde seg til skjermbilder i stedet for å kunne utforske siden selv.

Hele nettsiden min er en plattform for testing før den kan bli overført til ForzaSys sin egen plattform. Dataen jeg tester mot er derfor et utsnitt av ForzaSys sin database, hvor jeg har gjort noen endringer i dataen for å kunne skape situasjoner i dataen som er mer relevante å se på enn det som var tildelt.

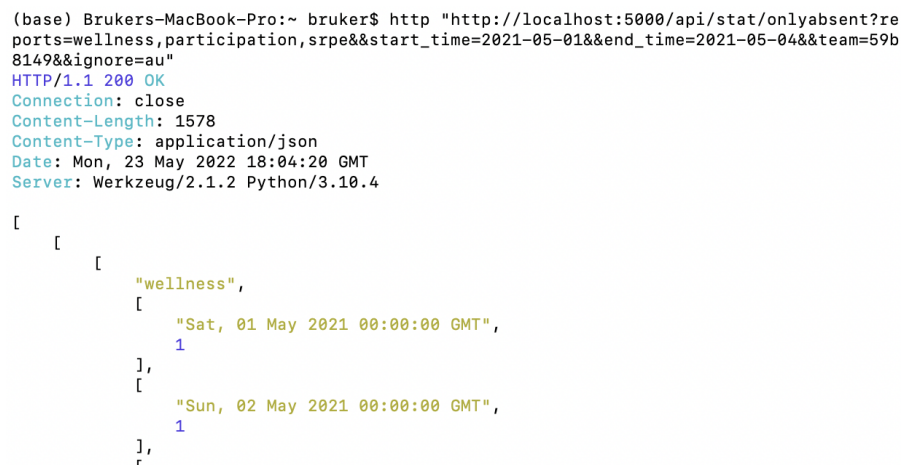
5.1 Enhetstesting

Enhetstesting går ut på å sjekke om de forskjellige enhetene og modulene av programmet oppfører seg på den måten den skal (Sawakinome, u.å.). Det er viktig med enhetstesting for å kunne forstå om hver enkel del som er laget fungerer som den skal. Om man tester større funksjonaliteter på en gang er det vanskeligere å fange opp hvilken del av programmet det er feil i. Så ved å bruke forutsigbare input i enhetstester kan man raskere fange opp hvor endringer trengs.

Under utviklingen av webapplikasjonen testet jeg koden etter hvert som hver enkel funksjon ble laget. For å kunne gjøre denne prosessen enklere og raskere brukte jeg et program som heter HTTPie.

Ved hjelp av dette verktøyet kunne jeg utføre API kall i konsollen uten å måtte gå gjennom brukergrensesnittet på nettsiden. Ved bruk av en linje i konsollen kan jeg få bekreftet om kommunikasjon med serveren og serveren sin spørring mot databasen var vellykket eller ikke, og om den har strukturen jeg er ute etter.

HTTPie input og output er vist i figur 53. Her vises et kall mot siden kjørt lokalt, hvor alt man trenger er URL-en til ressursen som skal lese for testing. HTTPie vil så skrive ut hvilken status som den mottar fra serveren, og formatet på JSON objekt den mottok.



Figur 53: Skjermbilde av konsollen hvor et API kall ble brukt gjennom HTTPie

5.2 System testing

Systemtesting går ut på å sjekke all funksjonaliteten til webapplikasjonen i sin helhet. Formålet med system testing er å evaluere om systemet har funksjonaliteten det trenger, at all mulig input gir outputen man ønsker, og hvilken opplevelse brukerne hadde ved bruk av siden. (Hamilton, 2022)

Denne typen testing var gjort av meg selv og ForzaSys. Når ny funksjonalitet eller oppdateringer var fullført ville det bli lastet opp til serveren til ForzaSys for å kunne testes. I det neste møte med ForzaSys ville Pål og Tomas åpne siden for å sjekke at den funksjonaliteten som var etterspurt var implementert på en akseptabel måte.

Det var også planlagt å ha en brukerundersøkelse hvor brukere av pmSys skulle kunne gjøre testing av nettsiden, men på grunn av at serveren stoppet å funke ble det denne testen gjort gjennom skjermbilder av løsningen.

5.3 Brukerundersøkelse

Et fokus fra ForzaSys var at visualiseringen av dataen skulle være presentert på en ryddig, oversiktlig og fin måte. For å kunne avgjøre hvilke design valg som ville være mest hensiktsmessig ble det foreslått å gjøre en brukerundersøkelse for å forstå bedre hva som funker og ikke.

På grunn av den treige utførelsen min av prosjektet var det ikke lang tid som kunne gå til denne undersøkelsen som medførte få tilbakemeldinger. Likevel er informasjonen som jeg fikk fra de tilbakemeldingene som kom fortsatt veldig verdifulle for å kunne forstå bedre hva som er foretrukket. Siden jeg har jobbet med siden over lengre tid er det vanskeligere for meg å forstå hva som vil virke unødvendig eller komplisert etter å ha sett på det samme flere ganger. Derfor er eksterne tilbakemeldinger ekstremt nyttig for å få et friskt innblikk til applikasjonen.

5.3.1 Google Forms

Undersøkelsen ble laget i Google Forms. Dette er en webapplikasjon fra Google hvor man kan legge inn spørsmål av mange forskjellige typer, flervalg, avmerking, og lineær skala var brukt for dette formålet. Det er også mulighet for å lage et valgalternativ for “Annet”. Dette er også veldig nyttig for det kan alltids være flere muligheter enn det man kommer på selv.

Når svar kommer inn vil applikasjonen generere kakediagrammer for flervalgsspørsmål, og søylediagrammer for avmerking og lineær skala for å kunne se over svarene som kom.

5.3.2 Oppsett

Med serveren som gikk ned ble det kun skjermbilder som var tilgjengelig for brukerne, noe som Google Form støtter. Siden det ikke var mulig for brukerne å teste siden la jeg inn mine egne erfaringer i noen av spørsmålene med hva som kan være relevant å vurdere for de ulike valgene for at det skulle være lettere for dem å visualisere bruken.

Skjemaet består av 7 spørsmål og en innledning. Hvert spørsmål hadde en overskrift med emne navn på funksjonaliteten som for eksempel “Tooltip”, og en beskrivelse hvor det ble gått over hva som skal vurderes, og om nødvendig en liten forklaring på hvorfor de forskjellige valgene kan være relevante.

Jeg og arbeidsgiver gikk over alle spørsmålene jeg hadde skrevet flere ganger for å være sikker på at det var tydelig forklart hva som skulle vurderes, og hvorfor det som vurderes kunne være nyttig å vite. Etter en mange endringer sendte jeg også undersøkelsen til broren min for å ha en ekstern input for å kunne vite om det som er skrevet er forståelig, selv for en som ikke har noen innsikt i det som blir spurt etter.

5.3.3 Spørsmål og Svar fra undersøkelsen

Jeg vil her gå over alle spørsmålene og svarene jeg fikk fra undersøkelsen og kort reflektering over hva som var forventet og de svarene som kom. Det ble totalt sendt inn 2

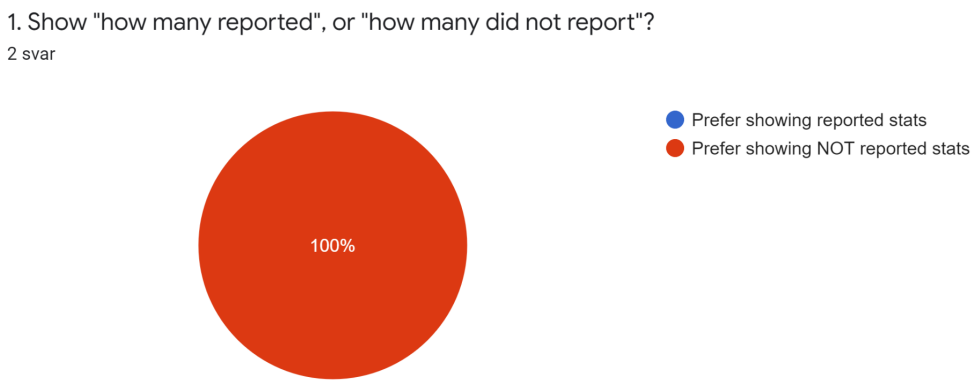
svar på undersøkelsen på grunn av knapp tid mot slutten av prosjektet som førte til å jeg ikke hadde tid til å nå ut til like mange som jeg hadde håpet.

5.3.3.1 Rapportert vs Ikke rapportert

I dette spørsmålet ble det spurt etter hvilken type statistikk som brukeren helst ønsker å se for en graf som viser rapporterings antall. I Google Form var spørsmålet presentert slik:

Figur 54: Skjermbilde av det spørsmålet i Google Form for statistikk valg

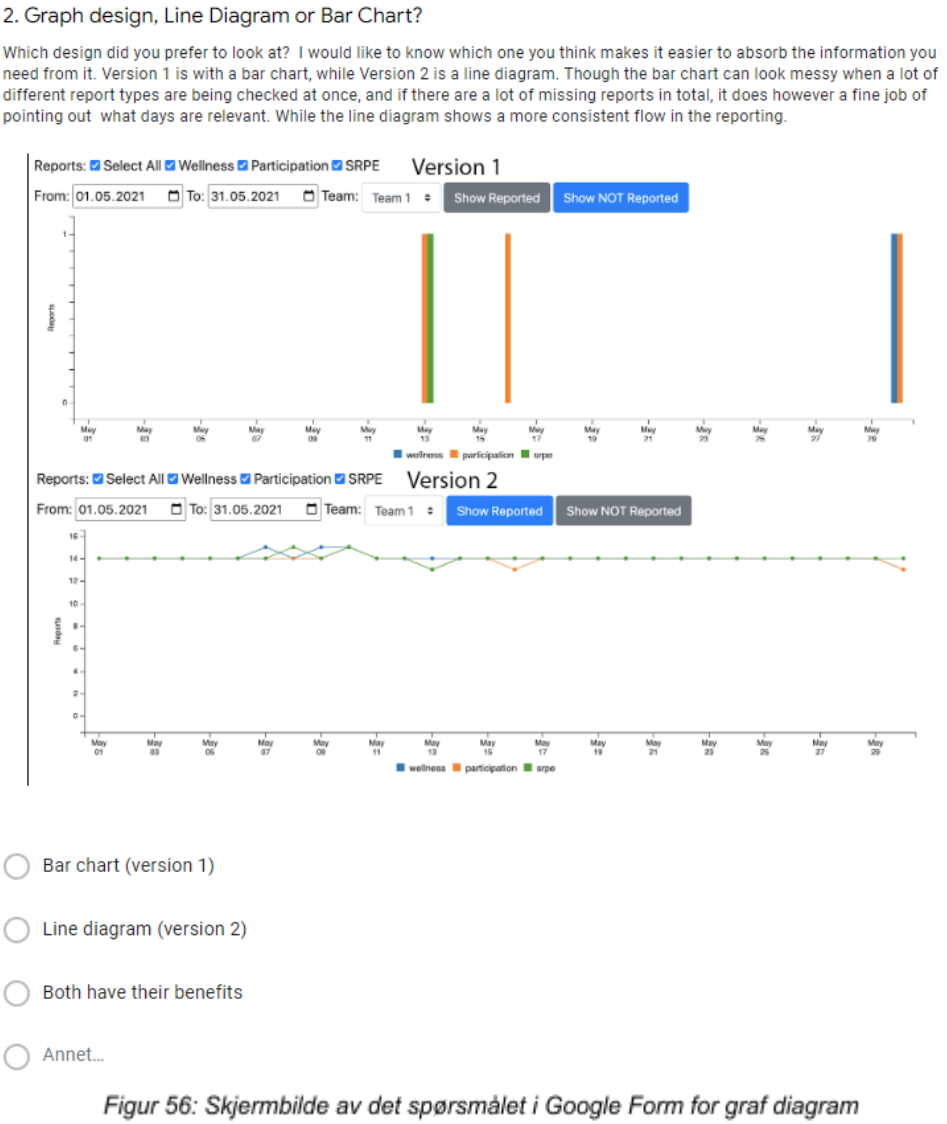
Jeg forventet her at statistikken som var mest relevant å vise var de som ikke hadde rapportert for i denne typen statistikk er det viktigst å få øye på avvikene man er ute etter og derfor ved å fjerne de som har rapportert vil man bare vise avvikene man er interessert i. Dette stemmer overens med tilbakemeldingene som kom inn hvor alle mente det var mest fornuftig å vise de som ikke rapporterer. Figur 55 viser utsnitt fra Google Form med svarene.



Figur 55: Skjermbilde av svar distribusjonen fremvist av Google Form for statistikk valg

5.3.3.2 Linjediagram vs Søylediagram

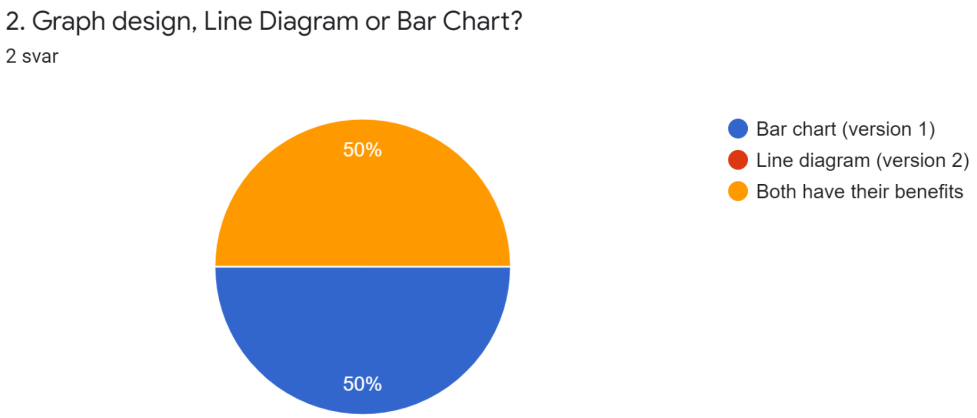
Her spørres det om hvilken type visualisering som er mest passende for denne statistikken for å kunne enklest ta til seg dataen som er vist.



Figur 56: Skjermbilde av det spørsmålet i Google Form for graf diagram

Her var jeg usikker på om hvilken type faktisk fremstilte dataen best, for søylediagrammet viser veldig enkelt interessepunktene i grafen for å kunne raskt skille ut avvikene, mens linjediagrammet gjør dette også i kanskje en litt svakere grad, men viser også en generell flyt i rapporterings statistikken.

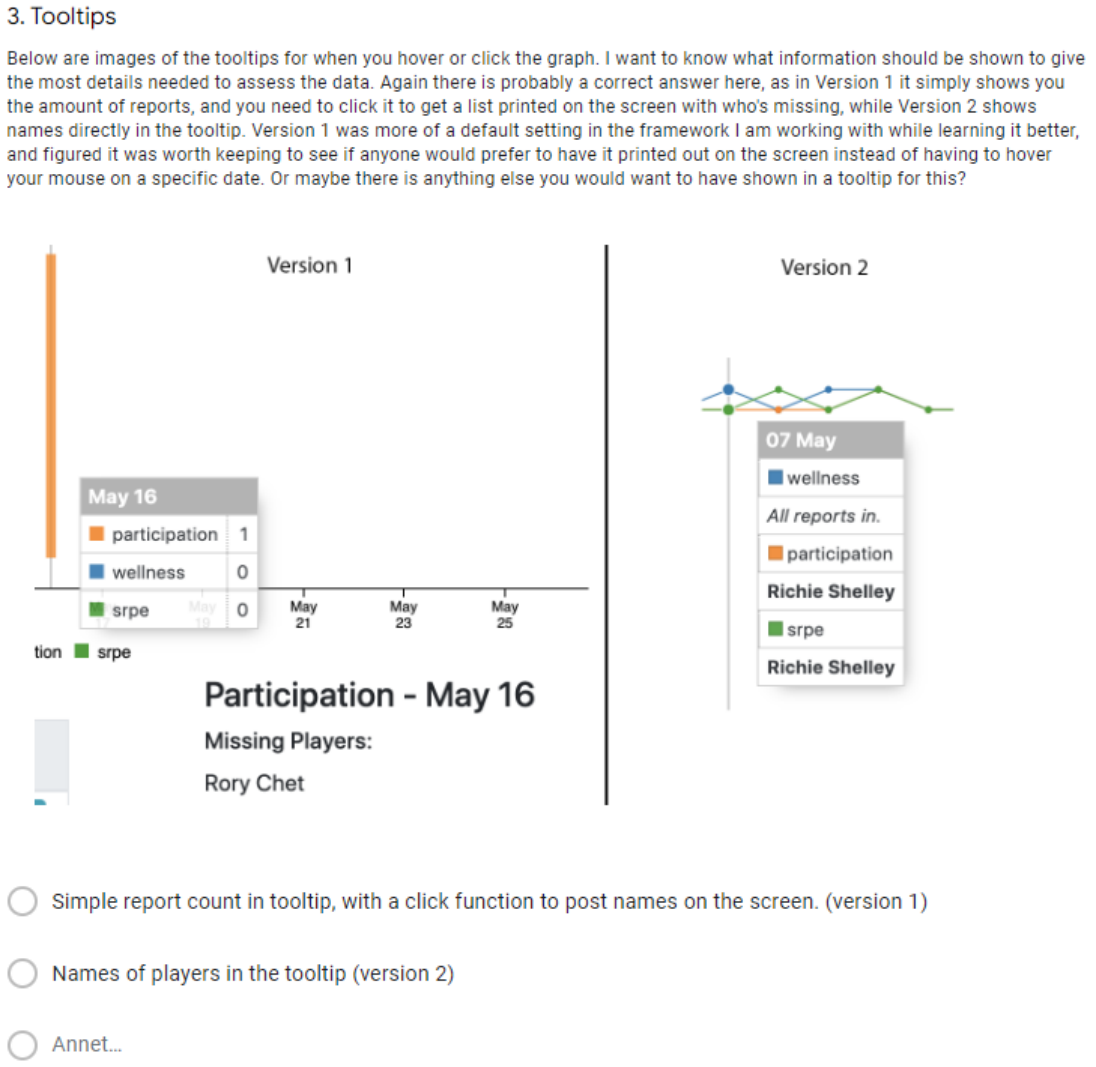
Tilbakemeldingene viser at søylediagrammet kan virke best, men at linjediagram er noe som kunne ha fungert.



Figur 57: Skjermbilde av det svar distribusjonen fremvist av Google Form for graf diagram

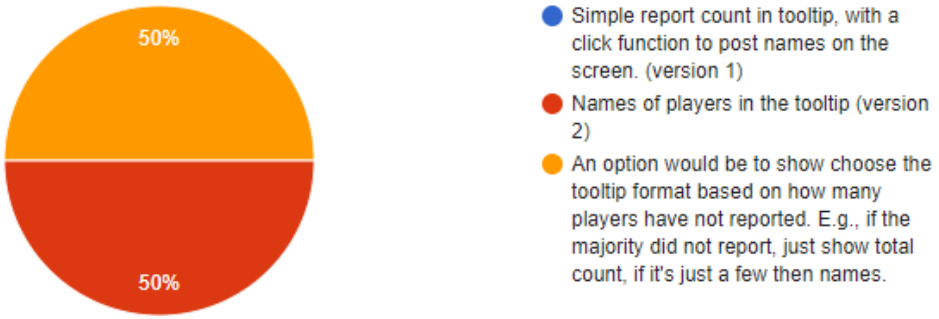
5.3.3.3 Tooltips

Her spørres det etter hva slags informasjon som de vil helst ha tilgang til i et tooltip for grafen for å kunne enklest vurdere dataen som er relevant.



Figur 58: Skjermbilde av det spørsmålet i Google Form for tooltip detaljer

Her forventet jeg at det mest relevante ville være å ha navnene i tooltip-et, for antall rapporter som mangler er noe som kan enkelt sees ut i fra grafen, eller ved å bare telle de som er i listen med navn. Det var i stedet et interessant forslag som kom inn under “Annet” valget som var tilbudt.

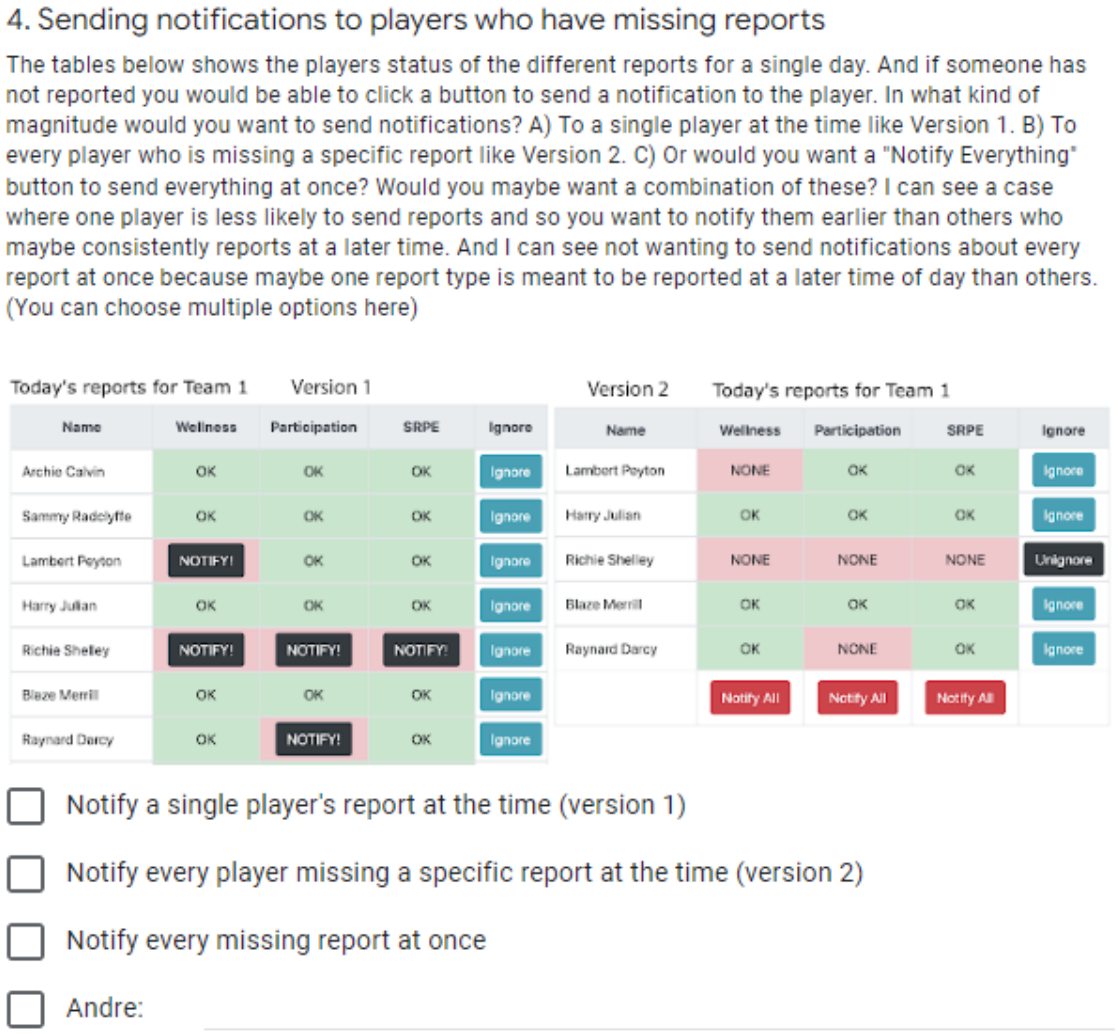


Figur 59: Skjermbilde av svar distribusjonen fremvist av Google Form for tooltip detaljer

Jeg hadde kun testet dataen med få avvik for jeg hadde en oppfatning at de fleste på et lag som bruker pmSys vil rapportere. Derfor var det ikke testet mye for når listen med navn blir lengre og mer uoversiktlig etter som antallet øker. En kombinasjon av begge var vurdert hvor det ved siden av rapport navnet vil også vise verdien til grafen, men jeg hadde aldri tenkt over å bruke de to visningene nøyaktig i de tilfellene de kunne ha en fordel. Så ved å legge inn at ved flere manglende rapporter enn for eksempel 5-7 ville tooltip-et går over til å helle vise bare et tall som versjon 1.

5.3.3.4 Varslings omfang

Jeg ville også vite i hvilken grad brukerne ville kunne sende ut varslinger til spillere. Så her sammenlignes versjon 1 og 2 for en spiller av gangen eller alle for en rapport, og i tillegg et alternativ for å sende alt på en gang som ikke ble implementert, men som kunne også ha sine fordeler.



Figur 60: Skjermbilde av det spørsmålet i Google Form for omfang av varslinger

Jeg valgte her å bruke avmerkingsbokser for det kan hende at en kombinasjon av alle disse ville ha sine bruksområder, og ikke at det er nødvendigvis et korrekt svar.

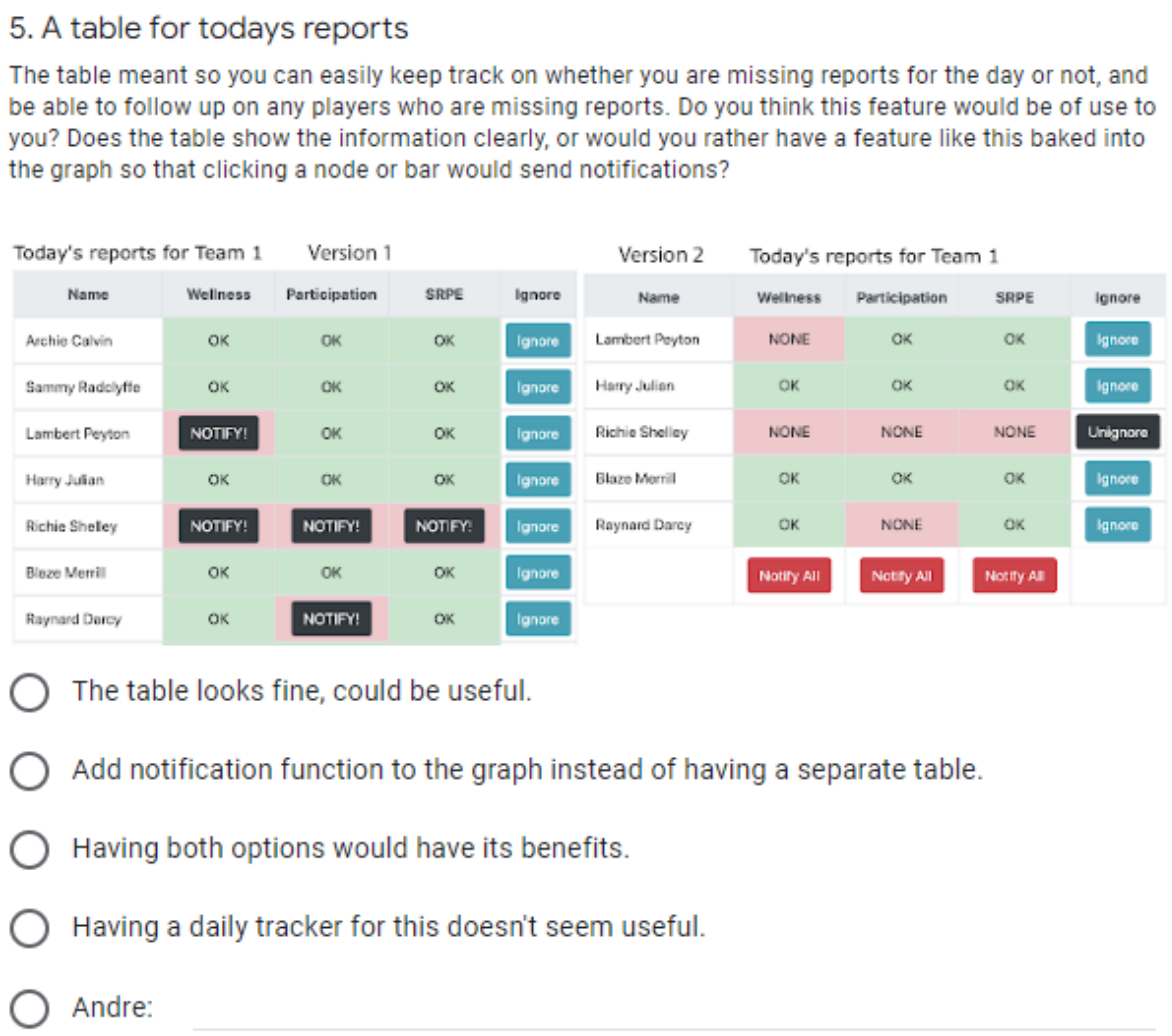
Figur 61: Skjermbilde av svar distribusjonen fremvist av Google Form for omfang av varslinger

Fra svarene kan jeg se at alle forslagene kunne være nyttige for brukeren. Det er altså ikke en spesifikk funksjon som er ute etter for denne typen funksjonalitet, men heller en fleksibilitet for å kunne gjøre som man vil for å kunne ha effektiviteten man ønsker selv.

Jeg trodde selv at en knapp som varsler alle spillere for en rapport ville være mest hensiktsmessige. Brukerhistoren jeg ville sett for meg var at en rapport ville ha et omtrent tidspunkt som den skulle helst vært levert, som for eksempel “participation” blir sent kort tid etter trening. Og i dette tilfellet ville en knapp som varsler alle som mangler den rapporten bli brukt etter at den forventede rapporterings tiden hadde gått. Fra undersøkelsen vises det derimot at flere er enig i at en enkel spiller sin oppfølging kan være mer nyttig enn flere på en gang.

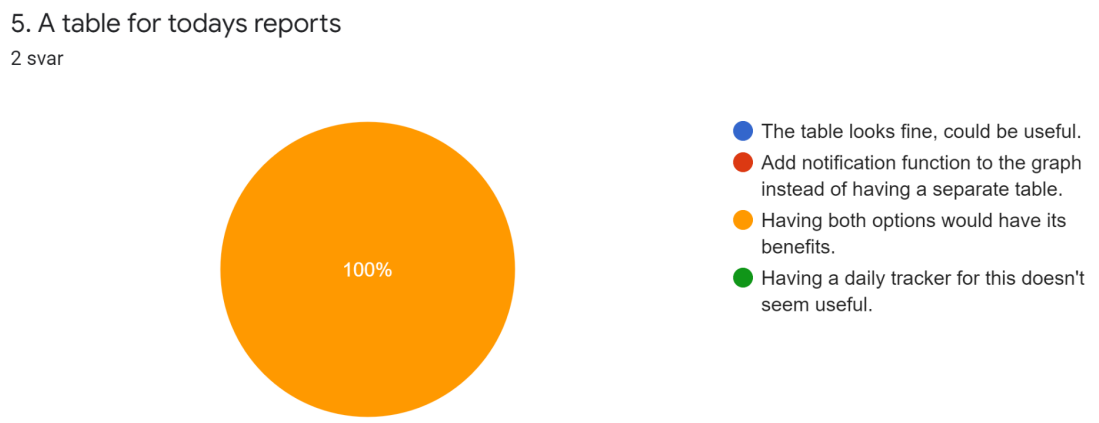
5.3.3.5 Tabell for rapport status

Dette spørsmålet var for å se hvor nødvendig det egentlig ville vært å ha en annen type visualiseringer for et mer spesifikt tilfelle som er dagens status. Jeg ville vite om en tabell kunne være nyttig for å se individuelle spilleres status, eller om dette er kun en litt mer komplisert versjon av grafen og at varslings funksjonen kunne heller vært en del av grafen.



Figur 62: Skjermbilde av det spørsmålet i Google Form for tabell design

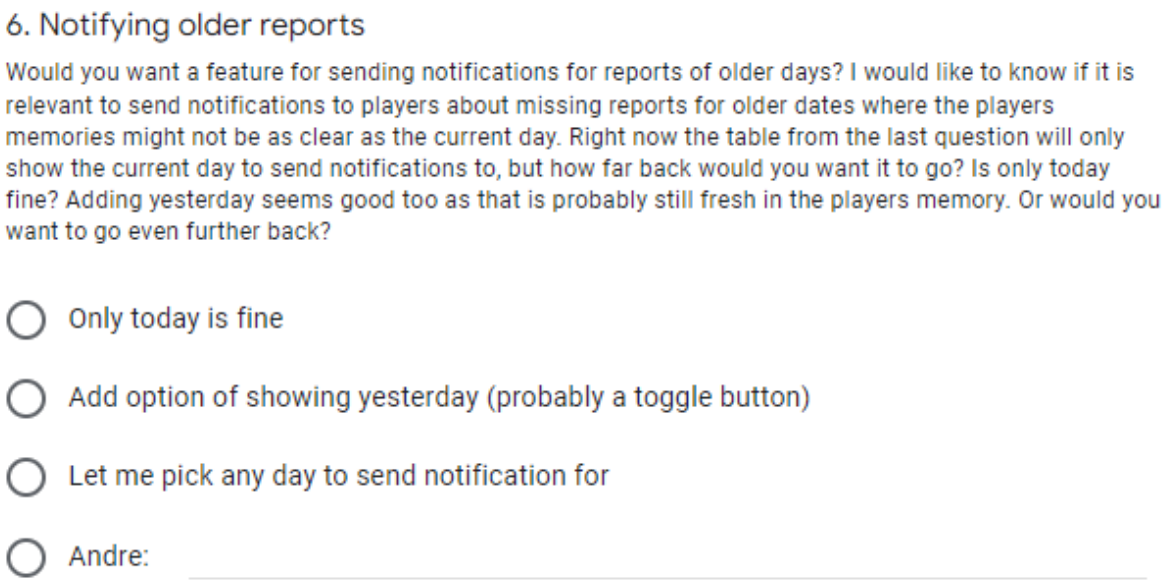
Svarene her var enige om at tabellen kan være et godt tillegg til grafen, men at varslinger kunne også være fornuftig å legge til i grafen for å da kanskje kunne sende grunnleggende varslinger gjennom den, mens hvis man ønsket å se flere detaljer bruke en tabell.



Figur 63: Skjermbilde av svar distribusjonen fremvist av Google Form for tabell design

5.3.3.6 Eldre rapporter

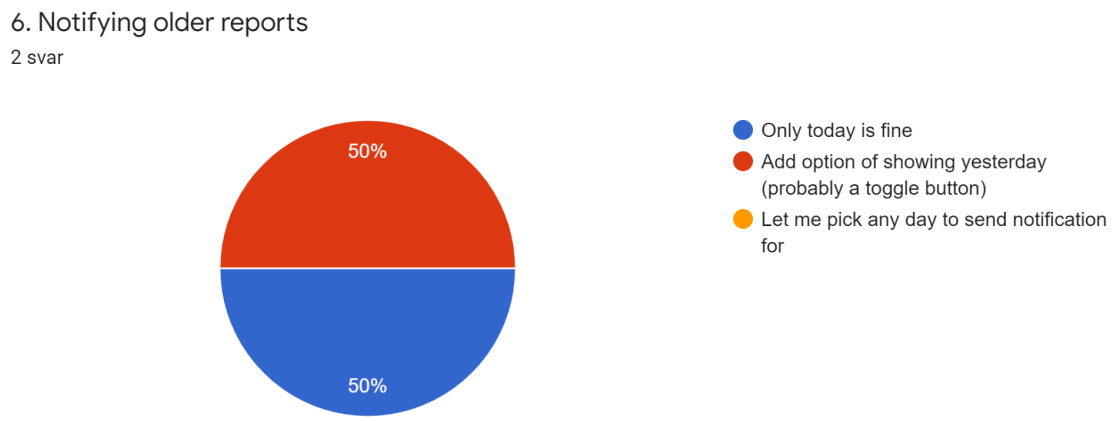
Her spør jeg om hvor langt tilbake det ville verdt å sende rapporter. Dette var noe som ble diskutert noen ganger i møter med ForzaSys. Vi var alle usikre på hvor relevant det ville være å sende varsler om eldre dager for da er ikke informasjonen like friskt i minne og dataen som blir samlet er da ikke like nøyaktig. Jeg ønsket derfor å sjekke hvor langt tilbake i tid brukerne mener det er vil være relevant for å purre på rapporter.



Figur 64: Skjermbilde av det spørsmålet i Google Form for tidsrom for varsling

Jeg visste ikke hva jeg ville forvente på dette spørsmålet. Funksjonaliteten jeg lagde var kun for en dag, men i utviklingsfasen var den også satt opp til å ha en datovelger, men som ble fjernet for å fokusere mer på at det var dagens rapport som var i fokus. Det er alltid lurt å kunne gi brukere så mye frihet de skulle ønske, om det ikke har stor bekostning av brukervennligheten. Derfor kunne en datovelger med default verdi for dagens dato fortsatt være en enkel endring om det viser seg å være interesse for en slik funksjonalitet.

Svarene endte opp slik:

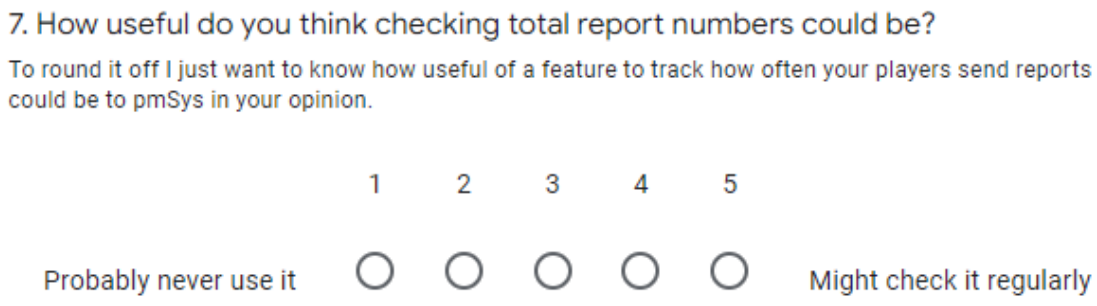


Figur 65: Skjermbilde av svar distribusjonen fremvist av Google Form for tidsrom for varsling

Så en datovelger er kanskje ikke nødvendig, men interesse for å lage en knapp for å bytte mellom dagens dato og gårsdagen ble påvist og noe som kanskje burde bli vurdert nærmere.

5.3.3.7 Nytteverdi

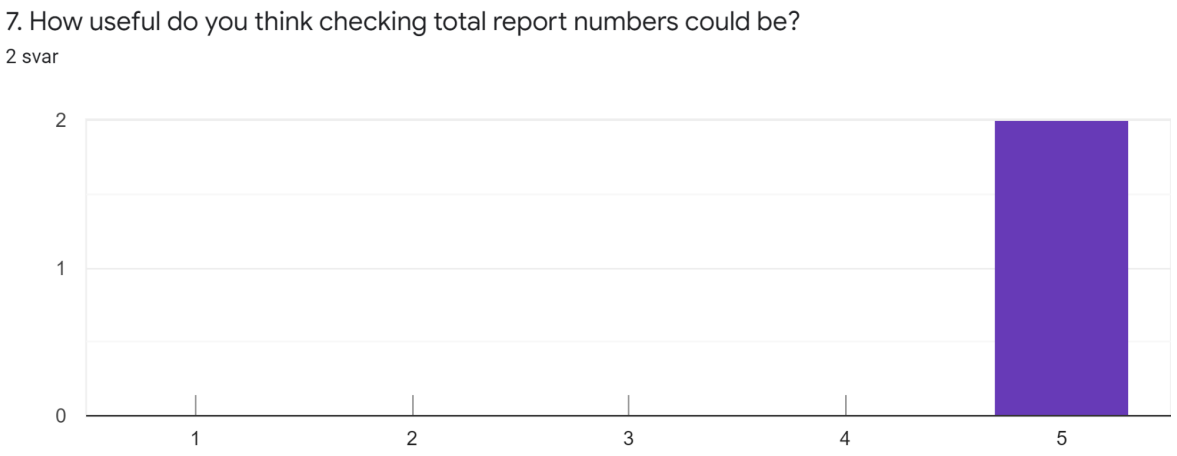
I det siste spørsmålet ville jeg vite om dette er en funksjonalitet som brukerne ville tro kunne være nyttig for dem å ha i pmSys. En slik abstrakt tanke er kanskje ikke helt enkel å ha et binært ja/nei svar på og ble derfor brukt en lineær skala for å sjekke interessen.



Figur 66: Skjermbilde av det spørsmålet i Google Form for nytteverdi

ForzaSys hadde allerede fått tilbakemeldinger før dette prosjektet startet at dette var en funksjon som kunne være interessant å ha i pmSys som var bakgrunnen for denne oppgaven. Likevel ville det være interessant å se om det fortsatt er noe som er relevant, og i hvilken grad det kanskje burde prioriteres.

Svarene som kom inn var enige i at dette ville være i stor grad en nyttig funksjon som de ville ha i pmSys.



Figur 67: Skjermbilde av svar distribusjonen fremvist av Google Form for nytteverdi

5.3.4 Refleksjoner over undersøkelsen

Undersøkelsen fremhevet godt hvor en funksjonalitet kan være universelt bra og som burde bli brukt videre, som med tanke på det første spørsmålet om hvilken statistikk som burde bli brukt. I tillegg påpekte den også flere ting som kan bli iterert på videre.

Noen funksjonaliteter som det ikke var tid til å implementere ble her påvist at kunne fortsatt ha nytte for seg. Og det kom også fram forslag til endringer som kan gjøres for å vise informasjonen på kanskje en enda bedre måte.

Dette er uvurderlig data for å kunne fortsette en utvikling av en funksjonalitet som dette videre. Det viktigste å tenke over er at brukeren er fornøyd og hva de mener kan være viktig og nytte å ha tilgang til. Uten tilbakemelding fra brukere kan det alltids hende at utviklerne sine meninger ikke stemmer overens med brukerne og produkte vil ble av lavere kvalitet enn det man først forventet.

Totalt sett virker det som at funksjonaliteten var godt mottatt. Jeg hadde lagt ved måter å vise misnøye på gjennom dedikerte svar eller å kunne legge til en kommentar i “Annet” valget. Det var også et ekstra spørsmål helt på slutten av undersøkelsen hvor man kunne legge inn en kommentar, men her kom det bare inn en kommentar som var “The added functions in pmsys would be of great value.” som videre påviser at dette kan være nyttig.

6 Oppdragsgivers vurdering

Pål Halvorsen sa at jeg har møtt kravene funksjonaliteten skulle ha for det som ble laget og at det vil sannsynligvis blir brukt videre av dem for å legge til denne funksjonaliteten på deres plattform. ForzaSys bruker bachelorprosjekter for å lage et “proof of concept” for en funksjonalitet de er ute etter og ikke nødvendigvis et program som kan direkte overføres til deres systemer. De vil kunne bruke noen kodesnutter som jeg har laget, eller lære av dem, og se på undersøkelsen om hva som burde bli behold og hva som burde bli endret på. Deretter integrerer de som regel alt inn i deres egen kodebase selv.

Det som er forventet å bli brukt videre er for eksempel hva slags SQL spørringer jeg gjør til databasen for å få informasjonen som trengs til diagrammene. Og ut i fra undersøkelsens blir det kanskje også bestemt hva man skal gå for og se på koden jeg har brukt for å lage visualiseringen og iterere til den fungerer på deres plattform.

7 Refleksjoner

Jeg har lært mye gjennom dette prosjektet av å kunne jobbe tett med et firma og kunne hjelpe å lage noe av verdi for dem. Jeg har allerede gjort noen refleksjoner rundt prosessens gjennomføring og viktigheten av en god gjennomføringsstrategi i Prosessdokumentasjon kapittelet, så her blir det mer reflektert over produktet og prosessen til utviklingen av produktet i seg selv.

Jeg er generelt fornøyd med det som ble laget. De forskjellige versjonene ga en god presentasjon til ForzaSys for å vise dem de forskjellige mulighetene som kunne bli brukt for å lage funksjonaliteten de var ute etter.

Det er også mange forbedringer jeg burde gjøre i koden som ikke ble fikset. Jeg burde ha et JavaScript dokument som begge versjoner deler for å ikke kopiere for mye kode. Jeg trodde når jeg begynte å skrive versjon 2 at den kom til å bli såpass annerledes at det ikke ville være en grunn til det. Det viste seg i stedet at strukturen av det som blir gjort er fortsatt lik mellom versjonene. Og etter hvert som jeg lagde versjon 2 var det mye funksjonalitet i den som jeg tenkte kunne være nyttig å legge til i versjon 1 og derfor ble det heller at jeg kopierte kode over i stedet for å ha en måte å dele dem på. Så jeg har lært for neste gang at med en gang kode blir kopiert er det nok nødvendig å samle alt på et sted med en gang for at det ikke bygger seg opp mange likheter over tid. Dette samme gjelder for HTML delen av siden. Jeg fant senere en nyttig del av Flask sin template funksjonalitet som jeg kunne tatt bedre fordel av for å kunne ha mer dynamiske sider enn det jeg allerede hadde.

Jeg er fornøyd med verktøyene, rammeverkene og bibliotekene jeg brukte for denne løsningen. Jeg følte at jeg hadde den funksjonaliteten jeg trengte gjennom hele prosjektet, men at det er mye i disse verktøyene som jeg fortsatt ikke har full oversikt over. Jeg skulle gjerne hatt en bedre forståelse for disse så jeg kunne lagd løsningen på en mer effektiv måte. Jeg leste mye gjennom dokumentasjonen til de forskjellige verktøyene jeg brukte, men jeg ser i etterkant at jeg burde nok gått mer i dybden for å finne alt jeg kunne få bruk for. Et eksempel her er i C3.js hvor jeg sier selv i dokumentasjonen her at jeg ikke finner noe sted hvordan jeg kan lese ut verdien til x-aksen. Jeg har gjort mye testing med objektene jeg blir tildelt av rammeverket, men jeg finner den ingen steder. Det som gjør at jeg tror at det må finnes der et sted derimot er at et default tooltip klarer å lese informasjonen som ligger der. Jeg i løsningen bruker datovelgeren min for å regne meg fram til hva som burde være på x-aksen, men C3.js klarer dette uten min datovelger, så om den finner det må det jo være mulig å få tak i det manuelt også.

Det er mange forbedringer som jeg skulle ønske jeg hadde tatt meg tid til. Jeg fikk ikke utført brukerundersøkelsen før utviklingsprosessen var mer eller mindre ferdig. Dette gjorde at jeg ikke kunne gjøre forbedringer ut i fra tilbakemeldingene jeg fikk. Nå i etterkant er det noen funksjonaliteter som jeg tenker kan være nyttige, eller andre måter å sette opp det jeg allerede har laget.

For eksempel en måte å gjøre varslinger på er å gjøre alle cellene i tabellen for manglende rapporter til å inkludere en checkbox og så kunne ha en knapp som sier “Notify Marked Players”. Det ville kunne vært en mulighet der for å kunne være mulig å lage en måte å markere alle spillere som mangler en rapport på en gang, eller individuelle spillere sine rapporter. Fra undersøkelsen ser det ut til at en bruker ville ha mest mulig fleksibilitet for hvor mange de vil sende rapporter til på en gang. Så ved å heller ha et checkbox system kan man gjøre det så fleksibelt som mulig. Det ville i hvert fall vært interessant å prøve.

Det er også mange forbedringer jeg må gjøre i min egen kode vaner. Når man ser nøyere på skjermbildene av koden vil man merke at variabelnavn er noen ganger veldig ubeskrivende. Som for eksempel “args3”, jeg skriver mye kode som jeg tror jeg kommer til å finpusse senere som bare ender opp med å bli igjen for etter 20 linjer er kanskje “args3” brukt flere ganger å det blir stående. Dette fungerer med tanke på at jeg er den eneste som trenger å forstå koden for å få den til å funke. I dette prosjektet det er ikke et team med utviklere hvor klarhet i koden er viktig for at flere skal kunne iterere på den. Koden skal derimot deles med ForzaSys, så dette er mer viktig enn jeg forsto i begynnelsen av kodingen. Om de skal kunne finne de delene som de er ute etter burde strukturen og klarheten i koden være et punkt jeg skulle ha fokusert mer på enn å tenke “dette funker for meg”.

Jeg er selvfølgelig misfornøyd med mengden funksjonalitet jeg fikk lagd. Det var diskutert flere oppgaver i starten av prosjektet som skulle utføres. På grunn av dårlig tidsbruk fikk jeg bare laget 2 av de tingene som ble foreslått i starten av prosjektet. Jeg skulle ønsket jeg hadde jobbet raskere for å kunne utforsket flere muligheter og løsninger. Dette har vært en veldig god læringsprosess for meg på mange måter, og hvis jeg hadde lagt inn mer innsats hadde jeg kunne lært mye, mye mer.

8 Oppsummering

Prosessen startet som et prosjekt for å lage flere funksjonaliteter for ForzaSys, men etter hvert som tiden gikk uten at produktiviteten ble opprettholdt til den graden som trengtes for å kunne fullføre alt vil jeg nok si at prosjektet ikke levde helt opp til forventningene. Jeg hadde håpet å kunne få utført flere oppgaver for arbeidsgiver enn det jeg gjorde. Av det som ble gjort var det godkjent, så på den siden er det ikke helt mislykket, men omfanget av oppgaven skulle vært mye større enn det den endte opp som.

Arbeidet som ble gjort var en suksess til en hvis grad. Jeg fikk laget noe av verdi til arbeidsgiver som de har godkjent at kan være nyttig for dem. Undersøkelsen som ble utført påviser at det som er gjort ville kunne vært nyttig for dem som brukere, og jeg har vekket oppmerksomhet for ForzaSys om hva som kan være interessant å se nærmere på.

Prosessens gjennomføring derimot er hvor mye mislyktes. Jeg hadde ingen plan ved inngangen av prosjektet som kunne ha hatt stor betydning for produktiviteten gjennom prosjektet. Jeg hadde ikke lagd en konkret måte jeg skulle jobbe på eller på hvilken måte jeg skulle forholde meg til ForzaSys, eller en måte å kunne garantere dem resultater.

Jeg har lært viktigheten bak prosess metodikk, gjennomføringsstrategier og planlegging, selv når man bare er 1 som jobber på et prosjekt. Jeg har lært viktigheten av å gå i dybden på rammeverk og verktøy som er i bruk for å kunne utnytte dem på en mer effektiv måte uten å måtte ty til workarounds.

9 Referanseliste

Martinez, G (2018, 6. februar) *How to create your own little Restful Web API without getting lost in the process — Part 1* [Illustrasjon]. Medium: Gabry Martinez. https://gabrymartinez.medium.com/how-to-create-your-own-little-restful-web-api-and-do-not-get-lost-in-the-process-part-1-cf8db6833ae4

Hamilton, T. (2022, 30. april) *What is System Testing? Types & Definition with Example*. Guru 99. https://www.guru99.com/system-testing.html#2

Pallets Projects. (u.å.). *Flask*. Hentet 24. mai. 2022 fra https://palletsprojects.com/p/flask/

PostgreSQL. (u.å.) *New to PostgreSQL?* Hentet 24. mai 2022 fra https://www.postgresql.org/

GetBootstrap. (u.å.) *Bootstrap, from Twitter*. Hentet 24. mai 2022 fra https://getbootstrap.com/2.0.2/

Meltzer, R. (2020, 3. desember). *What is JavaScript Used For?* Lighthouse Labs. https://www.lighthouselabs.ca/en/blog/what-is-javascript-used-for

D3js (u.å.) *Data Driven Documents*. Hentet 24 mai 2022 fra https://d3js.org/

C3js (u.å.) *Why C3?* Hentet 24 mai 2022 fra https://c3js.org/

IBM (2021, 23. juni). *Containerization* https://www.ibm.com/cloud/learn/containerization

Microsoft (u.å.) *Visual Studio Code FAQ.* Hentet 24 mai 2022 fra https://code.visualstudio.com/docs/supporting/FAQ

Datavizcatalogue (u.å.) *Line Graph*. Hentet 24 mai 2022 fra https://datavizcatalogue.com/methods/line_graph.html

Datavizcatalogue (u.å.) *Bar Chart.* Hentet 24 mai 2022 fra https://datavizcatalogue.com/methods/bar_chart.html

Sawakinome (u. å.) *Hva er forskjellen mellom enhetstesting og funksjonstesting.* Hentet 24 mai 2022 fra https://no.sawakinome.com/articles/technology/what-is-the-difference-between-unit-testing-and-functional-testing.html

10 Vedlegg

Vedlegg 1 - getReportStatOnlyAbsent() kode skjermbilde

```python
def getReportStatOnlyAbsent(called=0):
    args1 = flask.request.args.get("reports")
    args2 = flask.request.args.get("start_time")
    args4 = flask.request.args.get("end_time")
    args3 = flask.request.args.get("team")
    args5 = flask.request.args.get("ignore")
    print("Printing ignore input: {}".format(args5))
    ignored = args5.split(",")
    reports = args1.split(",")

    ignore_string = "("
    for i in ignored:
        ignore_string += "'" + i + "'"
        if i != ignored[-1]:
            ignore_string += ", "
    ignore_string += ")"
    print("SQL ready ignore list: {}".format(ignore_string))

    cur = conn.cursor()
    all_data = []
    cur.execute("SELECT count(DISTINCT owner) FROM datapoints WHERE owner LIKE '%{}%' AND owner NOT IN {} ".format(args3, ignore_string))
    team_size = cur.fetchall()
    size = team_size[0][0]

    for r in reports:
        cur.execute("""SELECT DATE(created), count(DISTINCT owner)
                       FROM datapoints
                       WHERE schema_name='{}' AND created>='{}' AND created<='{}' AND owner LIKE '%{}%' AND owner NOT IN {}
                       GROUP BY DATE(created) ORDER BY DATE(created);""".format(r, args2, args4, args3, ignore_string))
        db_query = cur.fetchall()

        db_list = [list(i) for i in db_query]  # Converts to list. I can not change a tuple which is what the queries gets me.

        for data in db_list:
            data[1] = int(size) - int(data[1])

        db_list.insert(0, r)
        all_data.append(db_list)

    if called == 1:
        return all_data
    else:
        return flask.jsonify(all_data)
```

Vedlegg 2 - getAbsenceName() kode skjermbilde

```python
@app.route("/api/stat/absence/name", methods=["GET"])
def getAbsenceNames():
    args1 = flask.request.args.get("team")
    args2 = flask.request.args.get("date")
    args3 = flask.request.args.get("schema")
    reports = args3.split(",")
    print("Query for Date: {}, Team: {}, Schema: {}".format(args2, args1, reports))

    cur = conn.cursor()
    all_data = []

    for r in reports:
        cur.execute("""SELECT DISTINCT owner
            FROM datapoints
            WHERE owner LIKE '%{}%'
            EXCEPT
            SELECT DISTINCT owner
            FROM datapoints
            WHERE schema_name='{}' AND owner LIKE '%{}%' AND to_date(CAST(created as char(10)), 'YYYY-MM-DD') ='{}'
            ORDER BY(owner);""".format(args1, r, args1, args2))

        db_query = cur.fetchall()
        all_data.append(db_query)

    return flask.jsonify(all_data)
```

Vedlegg 3 - onClick funksjon i C3.js for Versjon 1 skjermbilde

```javascript
var chart = c3.generate({
    bindto: '#chart',
    data: {
      columns: stats,
      type: 'bar',
      onclick: function (d, i) {

          let date = new Date($("#start_time").val());
          date.setDate(date.getDate() + d.index);
          let month_format = parseInt(date.getMonth()) + 1;
          let date_formated = date.getFullYear() + "-" + month_format + "-" + date.getDate();

          let url = "/api/stat/absence/name?team=" + $("#team").val() + "&&date=" + date_formated + "&&schema=" + d.id;

          $.get(url, function (data){

              let output = "";
              if(data[0].length > 0){
                  for (let i in data[0]){
                      output += convertName(data[0][i]) + "<br />";
                  }
              }else{
                  output = "No missing reports."
              }
              report_name = d.id.charAt(0).toUpperCase() + d.id.slice(1);
              $("#right-text").html("<h4>" + report_name + " - " + date.toDateString().split(' ').slice(1, 3).join(' ') +
                  "</h4><h6>Missing Players: </h6>" + output);

          });

      },
    },
```

Vedlegg 4 - showAttendance() kode skjermbilde

```
function showAttendance(){
    var btn = document.getElementById("show-rep-btn");
    btn.innerHTML = "Showing Reported";
    document.getElementById("show-rep-btn").classList.remove("btn-secondary");
    document.getElementById("show-rep-btn").classList.add("btn-primary");

    btn = document.getElementById("not-rep-btn");
    btn.innerHTML = "Show NOT Reported";
    document.getElementById("not-rep-btn").classList.remove("btn-primary");
    document.getElementById("not-rep-btn").classList.add("btn-secondary");


    // Getting the current values
    let reportInput = $("input[type=checkbox][name=reports]:checked").map(function(){
        return $(this).val();
    }).get();
    let starttimeInput = $("#start_time").val();
    let endtimeInput = $("#end_time").val();
    let teamInput = $("#team").val();

    // This URI will return how many reports where submitted between the selected dates.
    // Format: /api/stat?reports=XYZ&&start_time=XYZ&&end_time=XYZ&&team=XYZ&&ignore=XYZ
    url = "/api/stat?reports=" + reportInput + "&&start_time=" + starttimeInput + "&&end_time="
    + endtimeInput + "&&team=" + teamInput + "&&ignore=" + JSON.parse(localStorage.getItem("ignored_players"));


    $.get(url, function(data){
        displayStats(data[0]);
        makeAbsenceList(data[1]);
    });
}
```

Vedlegg 5 - convertName() kode skjermbilde

```
function convertName(autotokenid){
    if (autotokenid == "auth0|59b814967091e711e630026d"){
        return "Archie Calvin";
    }
    else if (autotokenid == "auth0|59b814917091e711e6300265"){
        return "Sammy Radclyffe";
    }else if (autotokenid == "auth0|59b814917091e711e6300266"){
        return "Shane Gladwyn";
    }else if (autotokenid == "auth0|59b814907091e711e6300264"){
        return "Rocky Dom";
    }else if (autotokenid == "auth0|59b814967091e711e6300268"){
        return "Albie Alastair";
    }else if (autotokenid == "auth0|59b814947091e711e630026a"){
        return "Braiden Ronnie";
    }else if (autotokenid == "auth0|59b814967091e711e6300266"){
        return "Ben Schuyler";
    }else if (autotokenid == "auth0|59b81492cbb73b1e63a1cbc5"){
        return "Lambert Peyton";
    }else if (autotokenid == "auth0|59b81494c71350685f02daa6"){
        return "Harry Julian";
    }else if (autotokenid == "auth0|59b81499cbb73b1e63a1cbc8"){
        return "Richie Shelley";
    }else if (autotokenid == "auth0|59b814937091e711e6300269"){
        return "Blaze Merrill";
    }else if (autotokenid == "auth0|59b814967091e711e630026c"){
        return "Raynard Darcy";
    }else if (autotokenid == "auth0|59b81495c71350685f02daa7"){
        return "Nikolas Maximilian";
    }else if (autotokenid == "auth0|59b81497cbb73b1e63a1cbc7"){
        return "Rodger Fearghas";
    }else if (autotokenid == "auth0|59b81490cbb73b1e63a1cbc4"){
        return "Rory Chet";
    }else if (autotokenid == "auth0|59b814937091e711e6300268"){
        return "Alpha Bodhi";
    }else if (autotokenid == "auth0|59b81497cbb73b1e63a1cbc6"){
        return "Norwood Pip";
    }
    else{
        console.log(autotokenid);
        return "Who is this?";
    }
}
```

Vedlegg 6 - ignorePlayer(name, id) kode skjermbilde

```
function ignorePlayer(name, id){

    // Checking for Ignore or Unignore.
    var btn = document.getElementById(id);
    let ignore_list = JSON.parse(localStorage.getItem("ignored_players"));

    if(btn.classList.contains("btn-info")){
        btn.innerHTML = "Unignore";
        document.getElementById(id).classList.remove("btn-info");
        document.getElementById(id).classList.add("btn-dark");
        ignore_list.push(name);
    }
    else{
        btn.innerHTML = "Ignore";
        document.getElementById(id).classList.remove("btn-dark");
        document.getElementById(id).classList.add("btn-info");
        ignore_list.splice(ignore_list.indexOf(name), 1)
    }



    localStorage.setItem("ignored_players", JSON.stringify(ignore_list));

    let report_display_check = document.getElementById("show-rep-btn");
    if (report_display_check.classList.contains("btn-primary")){
        showAttendance();
    }
    else{
        showAbsence();
    }
}
```

Vedlegg 7 - notify(name, report, id) kode skjermbilde

```
function notify(name, report, id){
    var btn = document.getElementById(id);
    btn.innerHTML = "Notifed";
    alert("Sent notification to: " + convertName(name) + "\nFor report: " + report);
}
```

Vedlegg 8 - notifyAll(repID) kode skjermbilde

```
function notifyAll(repID){
    id = "repID" + repID;
    var btn = document.getElementById(id);
    btn.innerHTML = "Notifed";
    report = notifyList[0][repID];
    let playerList = "";
    for (let p in notifyList[repID+1]){
        playerList += convertName(notifyList[repID+1][p]);
        playerList +=", "
    }
    playerList = playerList.slice(0, -2);
    alert("Sent notification to: " + playerList + "\nFor report: " + report);
}
```

Vedlegg 9 - Skjermbilde av kode bukt for “Select All” funksjonalitet

```
function checkall(source){
    var checkboxes = document.getElementsByName("reports");
    for (var cbox of checkboxes)
        cbox.checked = source.checked;
}

$('#checkboxlist input[type=checkbox]').click(function(){
    if (!$(this).is(':checked')) {
        $('.selectall').prop('checked', false);
    }
 });
```

Vedlegg 10 - Skjermbilde av nettsiden i sin helhet. Versjon 1

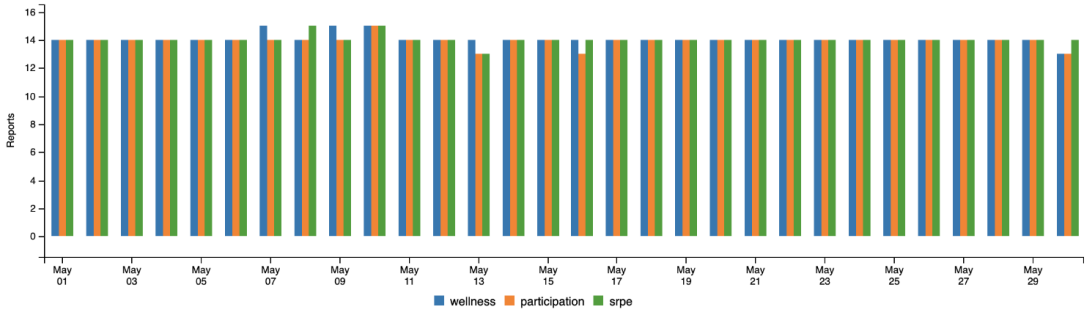
pmSys Concept - Player Report Count

Version 1 - Version 2

Version 1 has a bar design, grouping bars next to each other for each report type selected. You can click each individual bar and it will display a list of who hasn't reported that day. The table below shows a daily report for every player on the team and if they have sent in their reports for the day. If not, you can click the Notify button to send a notification, though of course this is just testing so no notification is sent, only an alert back to you to give some feedback that something happened. Like in this dataset, a specific player may have a long term injury and stopped reporting, in those cases it might be useful to not have them show up in the graph, and you can do that by clicking the "Ignore" button next to their name in the table.

Reports: ☑ Select All ☑ Wellness ☑ Participation ☑ SRPE

From: 01.05.2021 To: 31.05.2021 Team: Team 1 Show Reported Show NOT Reported



Today's reports for Team 1

Name	Wellness	Participation	SRPE	Ignore
Archie Calvin	OK	OK	OK	Ignore
Sammy Radclyffe	OK	OK	OK	Ignore
Shane Gladwyn	OK	OK	OK	Ignore
Rocky Dom	OK	OK	OK	Ignore
Alpha Bodhi	OK	OK	OK	Ignore
Braiden Ronnie	OK	OK	OK	Ignore
Norwood Pip	OK	OK	OK	Ignore
Lambert Peyton	NOTIFY!	OK	OK	Ignore
Harry Julian	OK	OK	OK	Ignore
Richie Shelley	NOTIFY!	NOTIFY!	NOTIFY!	Ignore
Blaze Merrill	OK	OK	OK	Ignore
Raynard Darcy	OK	NOTIFY!	OK	Ignore
Nikolas Maximilian	OK	OK	OK	Ignore
Rodger Fearghas	OK	OK	OK	Ignore
Rory Chet	OK	OK	OK	Ignore

Participation - May 16

Missing Players:

Rory Chet
Richie Shelley

Vedlegg 11 - Skjermbilde av nettsiden i sin helhet. Versjon 2

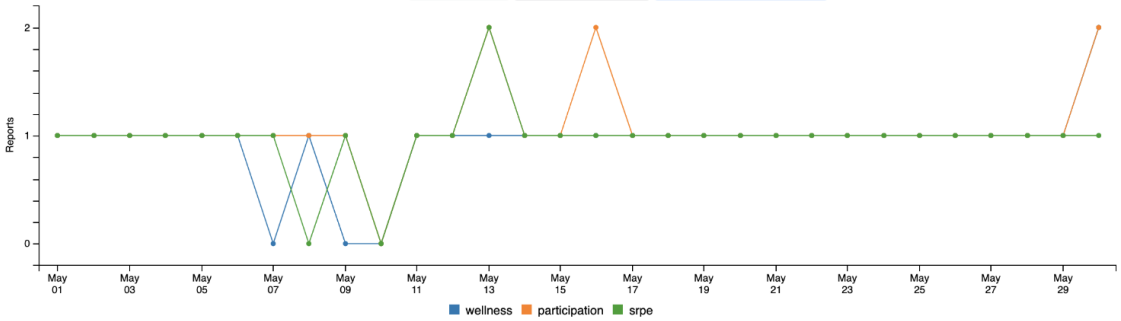
pmSys Concept - Player Report Count

Version 1 - Version 2

Version 2 has a line design, this might be more visually pleasing when viewing a longer timeframe. The information of who hasn't reported is now in the tooltip. The table below now has a "Notify All" option at the bottom which will instead send notifications to all who are missing that report.

Reports: Select All Wellness Participation SRPE

From: 01.05.2021 To: 31.05.2021 Team: Team 1 Show Reported Show NOT Reported



Today's reports for Team 1

Name	Wellness	Participation	SRPE	Ignore
Archie Calvin	OK	OK	OK	Ignore
Sammy Radclyffe	OK	OK	OK	Ignore
Shane Gladwyn	OK	OK	OK	Ignore
Rocky Dom	OK	OK	OK	Ignore
Alpha Bodhi	OK	OK	OK	Ignore
Braiden Ronnie	OK	OK	OK	Ignore
Norwood Pip	OK	OK	OK	Ignore
Lambert Peyton	NONE	OK	OK	Ignore
Harry Julian	OK	OK	OK	Ignore
Richie Shelley	NONE	NONE	NONE	Ignore
Blaze Merrill	OK	OK	OK	Ignore
Raynard Darcy	OK	NONE	OK	Ignore
Nikolas Maximilian	OK	OK	OK	Ignore
Rodger Fearghas	OK	OK	OK	Ignore
Rory Chet	OK	OK	OK	Ignore
	Notify All	Notify All	Notify All	