Et bachelorprosjekt for Oslomet - Storbyuniversitetet

I samarbeid med Innovasjon Norge

Et planleggingsverktøy for prosjekter

Lars Brekkå, Kristoffer Farstad, Håkon Åreskjold
veiledet av Trym Lindell

25.05.2020

Institutt for Informasjonsteknologi
Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo
Telefon: 22 45 32 00

PROSJEKT NR.
20-39

TILGJENGELIGHET
Offentlig

BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL	DATO
Et planleggingsverktøy for prosjekter	24.05.2020
	ANTALL SIDER / BILAG 137
PROSJEKTDELTAKERE Håkon Åreskjold s326169 Lars Brekkå s318930 Kristoffer Farstad s319467	INTERN VEILEDER Trym Lindell

OPPDRAGSGIVER	KONTAKTPERSON
Innovasjon Norge	Trevo Ledrick

SAMMENDRAG
Rapporten er sluttproduktet for Bachelorstudenter ved Oslomet - Storbyuniversitetet i faget DATS3900 Bacheloroppgave. Prosjektet er utarbeidet for Innovasjon Norge hvor sluttproduktet ble en responsiv webapplikasjon for prosjektplanlegging.

Applikasjonen skal bidra til å lette planlegging av fremtidige prosjekter i IT-avdelingen til Innovasjon Norge, og skal basere seg på data fra tidligere gjennomførte prosjekter. Ved å bruke data fra tidligere prosjekter ønsker Innovasjon Norge å se hvordan man kan planlegge fremtidige prosjekter, og om man mer nøyaktig kan fordele ressurser på denne måten.

3 STIKKORD
Webapplikasjon
Azure DevOps
Planlegging

Forord

Denne rapporten er sluttproduktet for Bachelorstudenter ved OsloMet – Storbyuniversitetet i faget DATS3900 Bacheloroppgave. I en annerledes tid har vi gjennomført et prosjekt i samarbeid med OsloMet – Storbyuniversitetet og Innovasjon Norge som har resultert i denne oppgaven. Oppgaven gikk ut på å lage en webapplikasjon som skal tjene som et prosjektplanleggingsverktøy for Innovasjon Norge IT-avdeling ved fremtidige IT-prosjekter.

Gjennom året har vi opplevd hvordan samfunnet har endret seg, og hvordan vi har måttet tilpasse oss en ny hverdag hvor hjemmekontor og digitalt samhandling står sterkt.

Vi ønsker å rette en stor takk til vår interne veileder ved OsloMet – Storbyuniversitetet, Trym Lindell, for gode samtaler og innspill underveis i prosessen. Hans innsats gjennom året har vært en inspirasjon, og vi har sett frem til møtene hvor vi har fått konstruktive tilbakemeldinger og motivasjon for videre arbeid. En takk rettes også til Innovasjon Norge for at de ønsket å stille som oppdragsgiver, ga oss teknisk veiledning og stilte med kontorlokaler en dag i uken.

Introduksjon

Introduksjonene vil gi en oversikt over de ulike aktørene i bacheloroppgaven og deres rolle i gjennomføringen.

Aktører

Gruppen

Gruppen består av tre studenter ved OsloMet – Storbyuniversitetet, Lars Brekkå, Håkon Åreskjold og Kristoffer Farstad. Gruppen ble etablert i 5. semester, og vi startet tidlig jakten på en bacheloroppgave for å ha denne på plass i god tid før eksamensperioden. Vi kjente hverandre godt fra før gjennom studiet, noe som gjorde kommunikasjon og samarbeid veldig enkelt. Vi kunne tidlig holde en tydelig og ærlig kommunikasjon, noe som fremmet samarbeid uten at vi måtte bekymre oss for gruppetenkning. Ved oppstart av nye grupper er sosial konformitet noe som ofte preger gruppens fremgang, og man er mer opptatt av at alle skal ha det bra heller enn fremgang og løsning av oppdrag (Janis, 1971). Dette fører noen ganger til at gruppen sitter fast i det som kjennetegnes som rosenrød idyll. Solomon Asch har også demonstrert enkeltmenneskers ønske om å passe inn ved å utsette uvitende mennesker for gruppepress og bedømme responsen. Asch viste hvordan mennesker er villige til å gå på akkord med seg selv for å ikke stikke seg ut og gå imot gruppens meninger (Asch, 1956). Nettopp denne fasen har vi kunnet unngå grunnet tidligere samarbeidsprosjekter, noe som har gjort oss i stand til å jobbe sammen på en effektiv og produktiv måte.

Alle gruppemedlemmene har erfaring både fra frontend og backend, og har kjennskap til C#, .NET og React fra skole i tillegg til andre webteknologier og programmeringsspråk.

Håkon Åreskjold

Håkon studerer informasjonsteknologi og har stor interesse for frontend, ny funksjonalitet, og er gruppens early adopter når det kommer til å prøve ut nye løsninger innen teknologi. Håkon har tidligere hatt eget firma innen foto og grafisk design, så det kreative var tidlig en del av hans hverdag. Håkon er godt kjent med både backend og frontend gjennom både skole og jobb.

Lars Brekkå

Lars studerer dataingeniør og er gruppens primus motor når det kommer til backend. Lars jobber deltid som systemutvikler hos Omega AS, og har god kunnskap om programutvikling og oppsett av utviklingsmiljøer.

Kristoffer Farstad

Kristoffer studerer også dataingeniør og har valgt å gå i retning av datasikkerhet. Han jobber deltid hos mnemonic AS som nettverksanalytiker innen it-sikkerhet, og har derfor sikkerhet som tilleggsfokus under utviklingen.

Innovasjon Norge

Innovasjon Norge er en virksomhet som, gjennom deres tilstedeværelse i det norske næringsliv, skal bidra til nyskaping, utvikling av konkurransedyktige norske bedrifter og utvikling av distriktene (Innovasjon Norge, 2019). Nærings- og fiskeridepartementet, og fylkeskommunene som eiere. Deres mandat er å bidra til vekst i norsk næringsliv gjennom kapital og kompetanse.
Innovasjon Norge har de seneste årene lagt merke til den store digitaliseringen som skjer i Norge, og som en aktør med mange prosesser og brukerinteraksjoner er de ikke immun mot slike endringer. De har derfor hatt en betydelig økning i ansatte i sin egen IT-avdeling de siste årene for å kunne møte de fremtidige utfordringene.

Vi ønsket å jobbe med Innovasjon Norge slik at vi fikk oppleve hvordan forholdene er på en faktisk arbeidsplass, med de utfordringer det bringer, samt å jobbe for en bedrift hvor potensialet for at mange benytter løsningen man lager er stor.

Det var totalt åtte bachelorgrupper som skrev oppgaven sin for Innovasjon Norge og de stilte med et mentorteam på seks personer. Oppstarten hos Innovasjon Norge bestod i å bli kjent med selskapet, lokalene, få utdelt adgangskort, og få satt opp alle tilganger som vi hadde behov for i arbeidet med Azure DevOps og 7Pace TimeTracker. Det var første gang Innovasjon Norge hadde gjennomført bacheloroppgave for et universitet, og de la til rette for at alle gruppene skulle få det de trengte fra start av.

OsloMet – Storbyuniversitetet

OsloMet – Storbyuniversitetet er Norges tredje største universitet med rundt 20 000 studenter og 2100 ansatte. Universitetets visjon er å "Leverer kunnskap som løser samfunnets utfordringer" og i så måte går vår oppgave hånd i hånd med denne visjonen. OsloMet – Storbyuniversitetet tilbyr bacheloroppgaven som en avsluttende oppgave for bachelorstudentene ved fakultetet for teknologi, kunst og design. Universitetet er ansvarlig for det faglige opplegget rundt utarbeiding av bacheloroppgave, og de er ansvarlig for å finne veileder til hver enkelt gruppe. Skolen tilbyr tre studieretninger for IT-studier på bachelornivå, herunder Dataingeniør, Informasjonsteknologi og Anvendt data.

Trym Lindell er gruppens veileder og har stilling som stipendiat ved Fakultet for teknologi, kunst og design som ligger under Institutt for informasjonsteknologi. Hans nåværende forskning ligger i segmentet Kunstig intelligens, og han er en del av forskningsgruppen Applied Artificial Intelligence som jobber med, blant annet, deep learning, recurrent neural network og reservoir computing.

Innhold

Figurer

Tabeller

Kodelister

Kapittel 1

Innledning

Rapporten vil ta for seg prosessen knyttet til planlegging, utvikling og avlevering av en webapplikasjon for Innovasjon Norge som skal tjene som planleggingsverktøy for prosjektplanlegging i ledergruppen. Rapporten deles opp i prosess- og produktdokumentasjon, og inneholder også en brukerveiledning for å kunne ta i bruk applikasjonen når den er ferdig utviklet og testet. Testdokumentasjon for applikasjonen er inkludert i prosessdokumentasjonen, siden omfanget ikke kvalifiserte til egen del av rapporten. Det tekniske språket er forsøkt tilpasset lesere med ikke-teknisk utviklingskompetanse, men det vil nok i noen deler av rapporten kreve innsikt i programutvikling for å skjønne innholdet. Dette gjelder spesielt i Produktdokumentasjon, men det er laget en ordliste som i noen grad vil kunne gi leseren en bedre forståelse. Det er også lagt vekt på å gi leseren en grundig innføring i metodikk og teknologier som nevnes og som er brukt i arbeidet. Av teknisk innsikt regnes her kunnskaper om objektorientert programmering, webutvikling med JavaScript, og tekniske begreper innen systemutvikling.

1.1 Problemstilling

Med et økende antall ansatte hos Innovasjon Norge blir dagens prosjekter stadig mer omfattende og ressurskrevende, og det kan være vanskelig å estimere riktig i forhold til hvor lang tid et prosjekt vil ta, hvor mange personer det vil kreve, og hvordan man jobber mest mulig effektivt. Feilestimering er både kostnadskrevende og sløsing med ressurser. Bedrifter har derfor mye å tjene på å estimere prosjekter riktig. Som en følge av denne økende kompleksitet i prosjektene, samt en stadig økende masse av data ønsket Innovasjon Norge å utforske mulighetene for å bruke data fra tidligere prosjekter til å kunne planlegge fremtidige prosjekter.

Måten det løses på i dag er gjennom et excel-ark, noe som er veldig begrensende for at flere kan benytte det som et verktøy, samt at det ikke er bærekraftig om den ansatte som eier dokumentet skulle slutte. I tillegg ga ikke nåværende rapporteringsløsning i Azure DevOps tydelig nok informasjon til utviklerne i form av

metadata. Dermed ønsket utviklerne mer informasjon knyttet til estimeringen de gjør på sine arbeidsoppgaver, og at dette skal tydelig fremstilles i den nye applikasjonen. Ledelsen så dette som en mulighet for mer helhetlig prosjektplanlegging hvor data som de trengte var tilgjengelig slik at man ikke var avhengig av like mange systemer for å samhandle.

1.2 Vårt bidrag

Gjennom vårt arbeid vil vi bidra til at planleggingen av nye prosjekter blir enklere å utføre for ledelsen i Innovasjon Norge og prosjektledere. Ved å studere tidligere prosjekter, skal lederne bedre kunne estimere fremtidige prosjekter basert på timer og personell som behøves. Det er ønskelig at denne informasjonen visualiseres for bruker gjennom grafer eller andre egnede presentasjonskomponenter. Det er viktig at komponentene beskrives på en slik måte at de er enkle å forstå for bruker, da dagens løsning er tungvint. Det vil i utgangspunktet være et planleggingsverktøy for ledere som også kan benyttes i arbeidet med å utvikle utviklerne i avdelingen slik at de blir bedre på estimering av oppgaver. I tillegg skal den kunne hjelpe den enkelte utvikler til å estimere sin egen tid i forhold til hvordan hun eller han arbeider mest effektivt. Ved å undersøke utviklerens estimat opp mot faktisk brukt tid, og hvor mange brukerhistorier og oppgaver utvikleren jobber med samtidig, ønsker man å se om dette kan være et verktøy for optimalisering og effektivisering.

Vårt bidrag vil dermed være:

- Webapplikasjon - prosjektplanleggingsverktøy
- Visualisering av timeestimat for utviklere
- Visualisering av tidligere prosjekter
- Visualisering av tid brukt på løsning av feil
- Visualisering av bruk av konsulenter
- En god plattform som kan videreutvikles på når den er avlevert

Kapittel 2

Prosessdokumentasjon

2.1 Forord

Prosessdokumentasjon vil inneholde de ulike fasene planlegging, utvikling og testing.

2.2 Planleggingsfasen

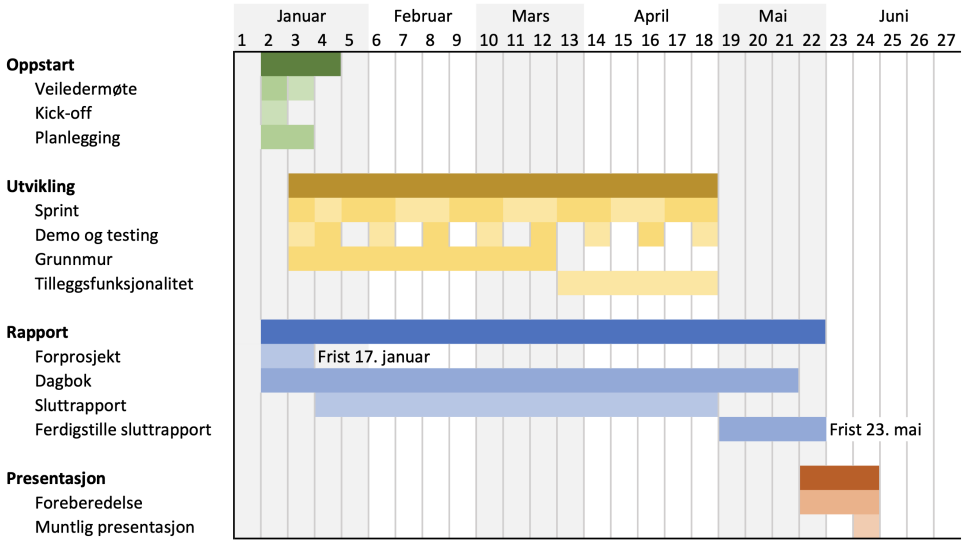
Gruppen startet planlegging av bacheloroppgaven allerede i oktober når tildelingen av oppgaver fra Innovasjon Norge skjedde. Gruppen strukturerte den initielle planleggingsfasen basert på Currie og Ravencross sin forskning på effekten mental forberedelse har på prestasjonen (Currie & Ravenscroft, 1997). Pensgård og Hollingen nevner visualisering av ønsket resultat, og gjennom diskusjoner og tankeprosesser begynte gruppen å forme seg et bilde av hvordan oppgaven kunne løses, og hva vi måtte forberede oss på (Pensgård & Hollingen, 2006). Gruppen diskuterte ulike arbeidsmetodikker, der "Smidig" skilte seg ut og virket som et fornuftig valg for gruppen.

2.2.1 Planlegging og metodikk

For å jobbe på en effektiv måte valgte gruppen å jobbe smidig og ved bruk av Scrum som rammeverk. Det ble avklart at arbeidsdagen hadde kjernetid fra 9-15 hver mandag, tirsdag og fredag med statusmøter klokken 9. Innovasjon Norge hadde lagt til rette for at studentene kunne jobbe i Innovasjon Norge sine lokaler én gang i uken og dette valgte gruppen å benytte seg av. Gruppen møttes hos Innovasjon Norge hver tirsdag, mens de andre dagene ble det booket grupperom på OsloMet for å kunne møtes for å jobbe. Gruppen har gjennom utdanningen erfart at det er mer effektivt å jobbe når man drar på skolen kontra å sitte hjemme, så derfor var det viktig og tidlig lage gode rutiner for arbeidshverdagen.

Fag fra studiet

Gruppen analyserte tidligere fag fra studiet for å identifisere hvilke som ville være fornuftig å støtte seg på gjennom prosjektet. Det var spesielt fag som Webapplikasjoner, Systemutvikling, Datasikkerhet og Programutvikling. Webapplikasjoner er det faget som nærmest ligner på bacheloroppgaven da det også her skal utvikles en webapplikasjon ved hjelp av C#, ASP.NET Core og React. Faget hadde god oppbygning og dokumentasjon som gruppen tok med seg i utviklingen av prosjektet. Videre kunne gruppen dra nytte av faget Datasikkerhet som ga innføring i trusselbildet, samt en forklaring av blant annet JSON Web Token og OAuth autentisering som blir brukt i oppgaven. Programvareutvikling gjorde gruppen i stand til å gjennomføre større programutviklingsoppgaver på egenhånd, samt å rapportere utviklingsarbeidet i etterkant. Fra faget Systemutvikling var gruppen godt kjent med prosjektarbeid knyttet til utvikling av programvare, og ved oppstart av prosjektet laget vi derfor et Gantt-skjema (figur 2.1) for hvordan vi så for oss fremdriften gjennom ukene frem til innlevering.



Figur 2.1: Planlagt fremdriftsplan laget 10. januar

Gruppen laget også en samarbeidskontrakt som skulle håndtere uenigheter og avvik, kontraktsfeste arbeidsdager, -tider og -lokasjon, og danne grunnlaget for planleggingsfasen. Samarbeidskontrakten er vedlagt som vedlegg B. Faget ga også en grundig innføring i metodikken "smidig" og hvordan man kunne jobbe med rammeverket Scrum.

Scrum

Scrum er et rammeverk for adressering av komplekse adaptive problemer, samtidig som man på en produktiv og kreativ måte leverer produkter av høyest mulig verdi (Schwaber & Sutherland, 2017). Dette rammeverket er regnet som lettvekt i forhold til det tyngre og mer dokumentasjonskrevende rammeverket Fossefall. Scrum ble utviklet av Ken Schwaber og Jeff Sutherland på 90-tallet.

Scrum utføres i såkalte sprinter som er iterasjoner på 2-4 uker, og for å gjennomføre en sprint trenger man en Scrum master, et utviklingsteam og en Produkteier. Teamet som utfører oppgavene opererer som en enhet og blir ikke fortalt hvordan den skal omforme brukerhistorier til inkrementer av potensielt ny funksjonalitet. Teamet organiserer seg selv og alle har ansvar for resultatet på slutten av sprinten.

Gruppen valgte å gjennomføre sprinter på 1-2 uker siden det aktuelle tidsrommet for bachelorskriving er kort, og ingen av gruppemedlemmene har stor erfaring med ledelse av Scrumteam fra tidligere. Det ble avgjort at gruppemedlemmene skulle dele på de forskjellige oppgavene og at alle skulle prøve ut rollen som Scrum master i løpet av arbeidsperioden. En Scrum master har ansvaret for arbeidet i gruppen, og er den som koordinerer jobben som skal gjøres. Scrum master skal også sørge for at gruppen arbeider etter prinsippene som gjelder i Scrum guiden. Sammen med Produkteier, herunder en representant fra Innovasjon Norge, prioriterer de hvilke arbeidsoppgaver som har størst prioritet for den kommende sprinten, og dermed hva utviklingsteamet skal jobbe med. Produkteier er kundens representant, og jobber gjerne hos kunden. Dens oppgave er å se brukerhistorier i sammenheng med hva kunden trenger, og dermed prioritere i forhold til hva som er viktig hos kunden ved start av sprint. En Scrum master skal i teorien ikke delta som en del av utviklingsteamet, men på grunn av mangel på ressurser kunne vi ikke avse en person per sprint til den rollen alene. Det ble derfor avgjort at Scrum master også måtte ta del i arbeidsoppgavene for hver iterasjon (Sommerville, 2016).

Siden gruppen hadde arbeidsdag hos Innovasjon Norge hver tirsdag ble det besluttet å arrangere sprintene slik at de startet på tirsdag for å kunne innhente produkteiers tilbakemeldinger og vurderinger av backlog. Dermed kunne sprint start-up-møte holdes i Innovasjon Norge sine lokaler, og man hadde god tilgang på representanter for Innovasjon Norge. Ved oppstartsmøtene hadde Scrum master ansvar for å passe på at Scrum-metodikk ble fulgt, samt fordele oppgaver fra backlog til den nye sprinten. Azure DevOps ble benyttet for sprintplanlegging, samt at gruppen også hadde tilgang på whiteboard for å gi alle gruppemedlemmer en samlet forståelse. Det var viktig å tilse at gruppen hadde en felles forståelse for oppgavene som ble introdusert til sprinten slik at resultatet ble tilnærmet det som var forventet ved oppstart av sprint (Sommerville, 2016).

Ved sprintens slutt har man et møte kalt “Sprint retrospekt” hvor man ser tilbake på sprinten som har vært og vurderer hva som fungerte og hva som ikke gikk

som det skulle. Videre vil man presentere det arbeidet som sprinten har gjort, og hvilke brukerhistorier man har klart å fullføre. De som er fullført kan markeres som ferdig, mens de som ikke er fullført må tilbake i backloggen. Dermed kan Scrum master og produkteier på nytt gjennomgå backlog og se hva som skal prioriteres til neste sprint. Når slike brukerhistorier går tilbake i backlog uten å være ferdig kaller man det for teknisk gjeld. Denne gjelden forsøker man å holde lav gjennom hele prosjektet, og er i realiteten urealisert arbeid. Gjelden kan senkes ved å fullføre arbeid som er påbegynt, men hvis det ikke er den type arbeid som har prioritet hos kunden så vil en slik gjeld hope seg opp. Derfor er det viktig at man før sprinten starter, klarer å estimere hvor mye arbeid man kan få ferdig, og arbeider effektivt med det, kontra å starte på for mye arbeid som man ikke får ferdigstilt og som ender opp som gjeld (Sommerville, 2016).

Gruppen arrangerte sprint retrospekt-møter på mandager for å få en oversikt over burndown chart og sprintens leveranse. Det var viktig å gjennomføre slike møter for å ha en oversikt over estimatene fra sprintplanleggingsmøtet og om de var riktige. Gruppen ønsket å finne skjæringspunktet mellom over- og underestimering for å jobbe mest mulig effektivt. Siden Innovasjon Norge ikke faktureres for gruppens arbeid er det mindre sannsynlig at Produkteier vil overstyre gruppens planlegging og prioritering da det er mer vanlig når ressurser faktisk må prioriteres, men innspillene fra Produkteier er likevel konstruktive og gir et godt bilde på forventninger og kundens opplevelser av produksjonen. Gjennom sprint retrospektmøter kunne gruppen måle fremdrift i applikasjonen og se arbeidet opp mot oppgaven som man jobbet mot, og hvordan resultatet virket å bevege seg i retning av ønskelig sluttresultat.

2.2.2 Oppgave

Den opprinnelige oppgaven fra Innovasjon Norge var:

> Vi ønsker oss et nytt og mer brukervennlig timeføringssystem for de ansatte. Det er også mulig å se på muligheten til å utvide systemet med funksjonalitet som gjør det mulig for utviklere til å også logge timer direkte til 7Pace TimeTracker i Azure DevOps.

Etter innledende møter i november ønsket Innovasjon Norge å endre oppgaven for å gjøre det mulig å jobbe med en oppgave som faktisk var mulig å lage på den tilmålte tiden og som ville bli satt i produksjon. Etter mailutveksling og nye samtaler fikk gruppen en forståelse for hvordan den nye oppgaven skulle se ut, og hva Innovasjon Norge var ute etter. Fra mailtråden kommer det frem at den nye oppgaven blir å erstatte timeføringsverktøyet 7Pace TimeTracker siden dette var en tredjepartsløsning som Innovasjon Norge ikke hadde eierskap over.

Ved oppstart i januar hadde gruppen nærmere samtaler med veileder og fikk nå beskjed om at det ikke var snakk om en erstatning av 7Pace TimeTracker, men

derimot en rapporteringsløsning ved å hente data gjennom API'et til 7Pace TimeTracker. Gruppen gikk straks i gang med å utforske API'ene knyttet til både Azure DevOps, som ble brukt for oversikt over arbeidsoppgaver, og løsningen de brukte for timeføring, 7Pace TimeTracker. Videre ble det også gjort research rundt TimeTracker-løsningen for å forstå hvordan den fungerte, og hvilke data vi kunne få ut fra denne. I arbeidet med tredjepartsløsningen oppdaget gruppen at et slikt rapporteringssystem allerede eksisterte i 7Pace TimeTracker, og det ble stilt spørsmål om hvorvidt gruppen skulle produsere en kopi av løsningen for å gå bort fra tredjepartsløsningen, eller om man måtte se på andre oppgaver. Svaret ble det siste, og gruppen måtte dermed omstille seg og gå i dialog med Innovasjon Norge for å finne en ny oppgave. Gruppen hadde i denne perioden startet med sprinter, så for å ikke miste for mye tid forsøkte gruppen å få skrevet en del på sluttrapporten frem til den nye oppgaven var klar. Sprintene vil bli nærmere gjennomgått i seksjon 2.5.

2.2.3 Covid-19

11. mars 2020 endret verden seg slik man kjenner den for en lang periode fremover. Etter en måneds arbeid med den nye oppgaven kom beskjeden om at Korona, eller Covid-19, hadde nådd Norge og mange bedrifter og institusjoner så seg nødt til å stenge dørene for å bekjempe spredningen av smitte. Innovasjon Norge var en av bedriftene som innførte hjemmekontor, og det samme gjorde OsloMet. Gruppen måtte nå omstille seg i forhold til både lokasjon og metodikk. Fra å være vant til å møtes fysisk, måtte nå gruppens medlemmer forberede seg på at videre kommunikasjon og interaksjon kom til å bli digital i uviss fremtid. I en periode preget av mye uvisshet, ble det desto viktigere å holde på rutiner som gruppen var vant til. Vi var på det tidspunktet ikke klar over hvordan viruset ville definere resten av tiden med prosjektet. Gruppen måtte derfor legge om planene for arbeidet og finne nye måter for samhandling. Gruppen fortsatte med statusmøter hver morgen klokken 9, men nå var det digital samhandling som sørget for kommunikasjon og interaksjon. Teams er en samhandlingsløsning fra Microsoft, som Innovasjon Norge benytter seg av, og denne hadde gruppen allerede god erfaring med å bruke. Det ble derfor gjennomført statusmøter med video gjennom Teams, og gruppen holdt på de faste arbeidsdagene for å opprettholde rutinen. På tross av omfattende endringer i samfunnet klarte gruppen å omstille seg slik at arbeidet kunne fortsette.

Hjemmekontor

Et annet aspekt ved nedstengning av samfunnet var at gruppemedlemmene måtte finne alternative løsninger for arbeidssted. Gruppen ble enig om å arbeide hjemmefra, siden man ikke hadde andre alternativer for å møtes. Gruppemedlemmene bor i små leiligheter som de deler med venner eller samboer, noe som gjør det utfordrende å ha plass til å kunne både jobbe og leve. Små leiligheter kan også være

utfordrende når det kommer til psykisk helse nettopp ved at man opplever å bli isolert på et lite område, på samme måte som ved en fengsling. Marte Rua finner at fengselsleger har erfaring med at fanger i isolasjon utvikler plager eller helseproblemer som følge av situasjonen de er i (Rua, 2012). Det er et kjent problem at isolasjon kan være en tøff påkjenning for kroppen. Peter Smith deler de ulike symptomene i fem kategorier (P. S. Smith, 2006);

- Fysiske plager som bl. a. hodepine, uregelmessig hjerterytme, fordøyelsesbesvær og muskerplager i nakke og rygg
- Forvirring og svekket konsentrasjon
- Hallusinasjoner, illusjoner og paranoide ideer.
- Følelsesmessige reaksjoner og impulsive handlinger
- Kronisk tretthet, reduserte kognitive funksjoner og søvnvansker

Derfor besluttet to av gruppemedlemmene at det ikke ville være overkommelig for de å bli værende i Oslo, og reiste til sine foreldre for å dra nytte av større frihet og fasiliteter. Det siste gruppemedlemmet ble værende som følge av deltidsjobb i samfunnskritisk bedrift, men ble tvunget til å gå til innkjøp av kontormøbler for å gjøre leiligheten levelig gjennom denne perioden. På tross av den geografiske avstanden klarte gruppen gjennom digital samhandling å fortsette arbeidet med oppgaven og sikre fremdrift. Dagene startet med at gruppen sjekket inn på Teams klokken 09.00 og avklarte hvilke oppgaver hver enkelt hadde for dagen. Om det dukket opp utfordringer, eller at noen stod fast med et problem hadde gruppen lav terskel for å ringe hverandre. Mindre spørsmål ble som regel løst gjennom meldingsutveksling. Hjemmekontor og restriksjoner tillot oss også å jobbe utenfor kjernetiden og på den måten utnytte at vi kunne jobbe når vi var på vårt mest effektive.

2.2.4 Teknologi

Det var viktig å velge teknologier som gruppen var kjent med fra før slik at vi kunne fokusere på å lage et produkt som ble omfattende, og ikke bare skrapte overflaten fordi kunnskap først måtte tilegnes, eller at gruppen måtte lære nye teknologier. Det er mange forskjellige alternativ der ute og alle har sine styrker og svakheter. Å velge kan være vanskelig da teknologier forandrer seg, noen forsvinner og andre kommer til. En ting som er sikkert er at det er en stor fordel å velge teknologier som er godt etablert. Gruppen hadde på forhånd sett for seg hvilke teknologier de ønsket å benytte seg av og de samsvarte med kundens teknologier på alle punkt.

Server

ASP.NET Core/C# .NET-plattformen er Microsoft sin plattform for å bygge ulike typer dataapplikasjoner. Til .NET er det mange utvidelser og ASP.NET er en utvidelse for utvikling av webapplikasjoner. ASP.NET Core er open source versjonen

av ASP.NET og ble først lansert i 2016 (Microsoft, 2020b). Den brukes for å bygge moderne skybaserte løsninger som f.eks. en webapplikasjon (Roth, Anderson & Luttin, 2019). Den nyeste versjonen av rammeverket er ASP.NET Core 3.1 som ble lansert Desember 2019.

Det er flere grunner til at vi har valgt å bruke ASP.NET Core i utviklingen. Fra tidligere var det bare ASP.NET Core alle gruppemedlemmene var kjent med av teknologier til bruk på serversiden. Et alternativ ville vært Java med Spring framework, men dette var ukjent for to av tre gruppemedlemmer. ASP.NET Core er en teknologi vi føler vi er forholdsvis komfortable med og som vi har også brukt i faget Webapplikasjoner. I «Introduction to ASP.NET Core» (Roth mfl., 2019) skriver de at ASP.NET Core integreres godt med React, noe som er en stor fordel da det er React vi skal bruke på klientsiden.

Programmeringsspråket vi bruker i ASP.NET er C#, som er et moderne objektorientert, type-safe språk. C# har flere funksjoner som gjør det godt egnet til å lage robuste og varige applikasjoner. Garbage collection henter automatisk tilbake ledig minne som er okkupert av ubrukte objekter som ikke kan nås, exception handling håndterer feil på en strukturert måte, og at språket er type-safe gjør at vi unngår å prøve å lese fra uinitialiserte variabler (Schonning mfl., 2020).

Klient

React React er et JavaScript-bibliotek for å lage brukergrensesnitt. I 2010 introduserte Facebook «XHP: A New Way to Write PHP» (Laverdet, 2010) som en utvidelse til PHP-kodebasen for å forbedre syntaksen til språket og hjelpe mot skriptangrep på tvers av nettsider. Tre år senere, 29. mai 2013, lanserte de React versjon 0.3.0 for offentligheten, men det var først i 2015 at React ble regnet for å være stabilt og selskap som Netflix (Kwok, 2015) tok biblioteket i bruk. November 2019 ble versjon 16.12.0 lansert og på GitHub er det over 1 million prosjekt knyttet til biblioteket.

React er bygget opp av komponenter. Disse er bygget for gjenbruk, og hver av komponentene kan enkelt og raskt oppdateres, om innholdet endres. Dette gjør at bare de komponentene som er endret oppdateres. På den måten forblir brukeren av applikasjonen på samme side uten at den må lastes på nytt, noe som gjør nettsiden raskere, som igjen gir en bedre brukeropplevelse.

Et alternativ til React er Angular. AngularJS ble første gang utgitt av Google i 2010. Versjon to kom i 2016 og endret navnet til Angular. Angular er et komplett rammeverk, mens React er et bibliotek til JavaScript. Både React og Angular er gode teknologier, men Angular fremstår som mer komplekst å lære seg. I Angular er man låst til TypeScript, mens ved React kan man velge mellom å skrive JavaScript eller TypeScript. Les mer om TypeScript i neste avsnitt.

Hos oppdragsgiver er både React og Angular en del av teknologiene de bruker. Valget falt raskt på React da alle medlemmene har kjennskap til dette fra tidligere.

I tillegg til React bruker vi også Typescript. Dette gir oss flere fordeler, for eksempel at språket er strengt typet.

Styling For styling av html-dokumenter, nettsider, benyttes Cascading Style Sheets (CSS), som er et format av stilark, som brukes for web.

> Stilsett (stilark) er en enkel måte å definere stiler på, det vil si layoutegenskaper som skrifttyper, farger og linjeavstander, slik at alle dokumenter fra et gitt nettsted får samme design. ... Hovedprinsippet er at ulike elementer i web-dokumenter – som overskrifter, tabeller og hypertekst – omkranses av egne tagger («merkelapper»), og at definisjonen på elementets utseende (som skriftstørrelse, farge, avstand over og under linjen) hentes fra stilsettet. (Store norske leksikon, 2020)

Altså når vi definerer en stil i en CSS-fil vil alle html-dokumentene som importerer den CSS-filen, få den stilen.

Det finnes flere biblioteker for å gjøre styling enklere og mer effektivt. Vi har valgt å bruke Bootstrap til å gjøre dette. Bootstrap er et front-end-bibliotek som f.eks. tilbyr løsninger for enkelt å kunne gjøre nettsider responsive, altså at innholdet på siden tilpasser seg skjermstørrelsen til enheten det vises på. Bootstrap-biblioteket finner også for React under navnet 'reactstrap" der elementene er React-komponenter (se 4.4.1).

Versjonskontroll

Git ble laget av Linux-grunnlegger Linus Torvalds tilbake i 2005 (Brown, 2018). Git er et program brukt for versjonskontroll i programvareutvikling, for å forenkle utviklingsprosessen. Gjennom Git får utviklere mulighet til å spore endringer i koden og gå tilbake til tidligere versjoner hvis en senere utgivelse skulle føre til feil. Git kom som et svar på Bitkeeper sin utestengelse av Linux-utviklere fra tjenesten, etter at brukere forsøkte å rekonstruere oppbyggingen til Bitkeeper.

Git kjennes også under navnet Github som er et grafisk brukergrensesnitt (GUI) for kommandolinjeversjonen. Denne typen GUI har gjort det enklere for ferske utviklere å ta i bruk versjonskontroll, uten å ha full oversikt over filstruktur eller kommandolinje-klienter.

I tillegg til å være en plattform for versjonskontroll har Git også blitt sett i lys av dens evne til å bidra til sosial koding. Gjennom Git har man sett en betydelig økning i samarbeidsprosjekter innen utvikling av programvare (Antonio Lima, 2014), og denne økningen tilskrives git-plattformens samarbeidsmuligheter. Gjennom Git har man fått mulighet til å følge andre utviklere og dermed finne gode kodeløsninger som andre har laget.

For vår del er dette et nyttig verktøy da vi er tre personer som sitter på hver vår pc samtidig og programmerer. Når vi bruker versjonskontroll har vi en måte å jobbe

på et felles prosjekt, på en strukturert måte. I tillegg gjør dette at vi får lagret all kode vi skriver i skyen, noe som gir oss backup hvis dataen skulle bli ødelagt. Innovasjon Norge bruker også repositories gjennom Azure DevOps og ønsket å få prosjektet lastet opp i Git på deres område. Git samhandler også med Visual Studio slik at det er sømløst å samarbeide og bytte mellom hverandres branches hvis noen skulle trenge hjelp.

2.2.5 Prosessverktøy

Innovasjon Norge jobber i sine team smidig ved hjelp av Scrum. Vi ønsket å jobbe på denne måten og Innovasjon Norge la til rette for dette blant annet ved hjelp av sprintplanleggingsverktøyet i Azure DevOps (2.2.6) som vi fikk tilgang til. Vi fikk også innblikk i hvordan de holdt daglige møter de dagene vi satt på kontoret hos dem. I avsnittene som følger vil vi forklare hva det vil si å jobbe smidig og hva rammeverket Scrum gjør.

Smidig

Agile eller smidig metodikk er en motvekt til den tunge og dokumentdrevne fossefallsmetoden som lenge var ledende innen prosessverktøy. Det var på slutten av 1990-tallet behov for en raskere program- og prosessutvikling som kunne håndtere endringer i programvare og teknologi på en bedre måte enn Fossefallsmetoden. Hurtig programvareutvikling ble kjent som smidig utvikling eller smidige metoder, og dette tok virkelig av og metoder som for eksempel “Ekstrem programmering” (Beck, 1999), Scrum (Schwaber & Sutherland, 2017) og DSDM (Stapleton, 2003) ble utviklet. Smidige metoder har alle en rekke ting til felles (Sommerville, 2016):

- Det finnes ingen detaljert systemspesifikasjon som forteller alle steg på veien. Dette skapes mens man utvikler.
- Utviklingen gjøre i en serie av inkrementer hvor sluttbruker og andre interessenter er involvert gjennom spesifisering og evaluering av hvert inkrement.
- Verktøy som testing, systemintegrasjon og konfigurasjonsstyring brukes for å støtte utviklingsprosessen.

I smidige utviklingsprosesser skal utviklerne jobbe i korte tidsintervaller på noen uker, for så å få tilbakemelding fra kunde, slik at man hele tiden korrigerer retningen mot sluttproduktet og for å unngå å levere et produkt som kunden ikke ønsker.

Et resultat av denne smidige bevegelsen som vokste frem ble et møte mellom ledende representanter fra ulike smidige metoder, deriblant ekstrem programmering, Scrum, DSDM, Crystal og Adaptive Programvareutvikling (Beck mfl., 2001). Dette møtet resulterte i Det agile manifest, som fungerer som retningslinjer for smidig utvikling.

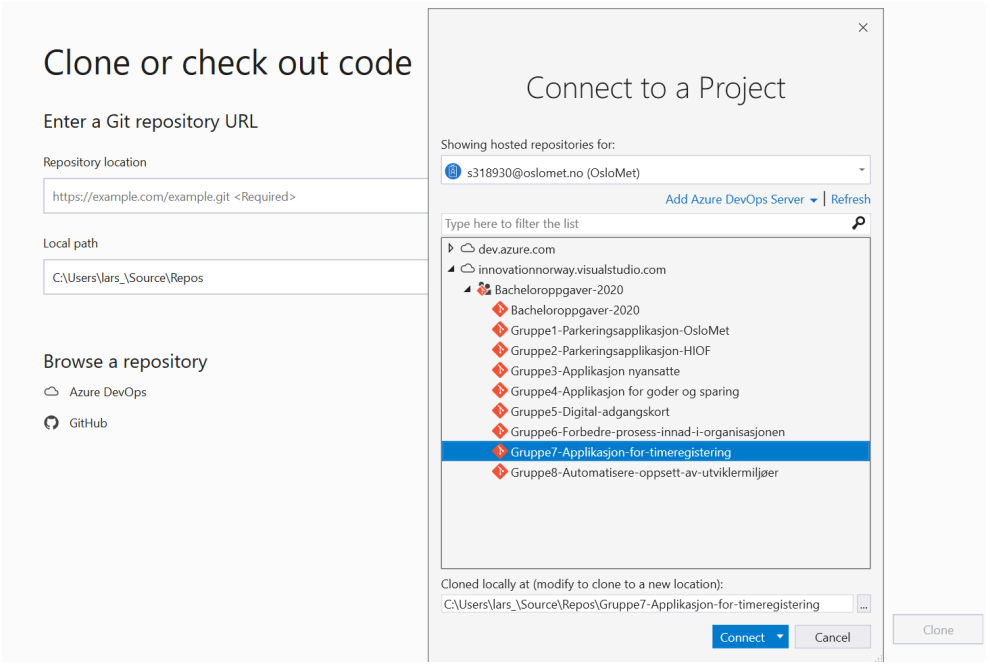
Tanken er at prosessen skal drives av gruppen og kunden i tett samarbeid, og at man på den måten skal gi kunden gode muligheter til å påvirke sluttresultatet slik at det blir så nært opptil det kunden ønsker som mulig. Når man oppdager at endringer kreves så er det viktigere å være adaptiv til endringen enn å følge en plan slavisk.

2.2.6 Andre programvarer og verktøy

Visual Studio

Visual Studio er et integrert utviklingsmiljø som vi bruker til å skrive kode og tester. Vi har valgt å bruke Visual Studio til å skrive backend-koden, som er C#. Grunnen til at vi bruker dette er at Visual Studio er laget av Microsoft og er da tett knyttet opp mot .NET-utvikling. Visual Studio gjør det lett å kjøre applikasjonen lokalt og har også innebygd funksjonalitet for testing. Visual Studio integrerer også med Azure DevOps repos (se 2.2.6), som gjør at vi lett kan hente repoer fra Azure.

Figur 2.2 viser hvordan vi enkelt kan klone prosjektene våre fra Azure DevOps repos i Visual Studio.



Figur 2.2: Eksempel på kloning av kode fra et Azure repo i Visual Studio

Visual Studio Code

Visual Studio Code er en lett tekstbehandler fra Microsoft, som vi hovedsakelig bruker til å skrive frontend-kode, da den har innebygd support for JavaScript.

Den er bygget ved hjelp av webteknologi og er laget på toppen av Chromium nettleser og node.js, ved hjelp av HTML, CSS og Javascript.

Azure DevOps

Azure DevOps er Microsoft sitt verktøy for å hjelpe utviklere planlegge og samarbeide i teams, og for å bygge og publisere applikasjoner. Innovasjon Norge bruker allerede dette verktøyet, så det var ønskelig at gruppen benyttet dette slik at de kunne organisere alle bacheloroppgavene i samme verktøy. Azure DevOps gir tilgang til blant annet Azure Repos, Azure Pipelines og Azure Boards (Sherer mfl., 2019).

Azure Boards Azure Boards er det vi bruker for å ha oversikt over systemutviklingen. Her har vi oversikt over user stories, backlog items, task, features og bugs. Dette forenkler arbeidsprosessen når det kommer til å bruke smidig-metodikk (se 2.2.5) og Scrum (se 2.2.1).

Azure Repos Azure Repos består av flere verktøy som vi bruker til versjonskontroll av koden vår. Versjonskontrollen vi bruker er Git (se 2.2.4), som er integrert i Azure Repos (Microsoft, 2020c). Et repository, eller repo, er et område som brukes for lagring av noe, i vårt tilfelle programvare. Slike repos brukes for å lettere kunne samarbeide om koden, og for å ha versjonskontroll underveis i utviklingsløpet. Man deler gjerne opp en repository i flere grener, eller branches. På den måten kan flere jobbe med utgangspunkt i samme kode uten at man påvirker hverandres branch. Når man er ferdig med iterasjonen, merger man grenene sammen igjen. På denne måten får man et nytt utgangspunkt for versjonskontroll og videre arbeid med koden. Samtidig kan man gå tilbake til tidligere grener skulle det vise seg at man har behov for det.

Azure Pipelines Azure Pipelines er verktøyet vi bruker for å automatisere byggingen og deployeringen av løsningen vår. Her kombineres continous integration og continous delivery (Danielson mfl., 2019). Vi kombinerer dette med Azure Repos ved at vi automatisk bygger prosjektet når vi pusher kode til Dev-branch og Release-branch.

Notion

I semesteret før bachelorprosjektet benyttet gruppen seg av Notion for å notere tanker og annen relevant informasjon til et kommende bachelorprosjekt. Notion er en gratis tjeneste med slagordet "The all-in-one workspace for your notes, tasks, wikis, and databases." – en tjeneste for notater og arbeidsoppgaver. Man har også mulighet til å samarbeide på samme prosjekt, på samme tid på en enkel måte da arbeidsområdet kan deles med flere brukere.

Etter prosjektstart fungerer Notion som en mellomstasjon før tekst eventuelt skrives inn i sluttrapporten. En av fordelene med å skrive i Notion er at teksten formateres som ren tekst. Dette betyr at teksten enkelt kan kopieres inn andre tekstbehandlingsprogrammer uten å tenke noe særlig på formatering.

Microsoft Teams

Innovasjon Norge arrangerer de fleste møter og seminar gjennom Teams, noe som gjør at man kan være med selv om man ikke er fysisk tilstede. Innovasjon Norge foretrekker kommunikasjon på Teams over e-post.

LaTeX

I arbeidet med rapporten besluttet gruppen tidlig å benytte seg av et kjent verktøy for behandling og klargjøring av rapporter, nemlig LaTeX. Systemet er laget av Leslie Lamport og er bygget på arbeidet til Donald Knuth som laget TeX på slutten av 70-tallet. I 1985 var første versjon av LaTeX ferdig, 2.09 som det mystisk ble kalt, og vokste seg gjennom større gjennom de neste årene før det på starten av 90-tallet var blitt en global instans i verdens forskermiljø (Lamport, 1994). Forskere brukte gjerne LaTeX-format når de sendte rapporter til hverandre, og på grunn av den store interessen for dette systemet ble derfor LaTeX 2_ε laget. Tanken var at forfatteren av en artikkel kun skulle måtte forholde seg til innholdet i sitt arbeid, og ikke måtte bruke tid på visuell fremstilling.

Det var Frank Middelbach som ledet arbeidet når LaTeX ble implementert i 1994, og med seg hadde han Johannes Braams, David Carlisle, Michael Downes, Alan Jeffrey, Sebastian Rahtz, Chris Rowley og Rainer Schöpf. Dette er den nåværende versjonen, og er oppgradert for å håndtere tilpasning av dokumentet bedre. Dette gjelder spesielt med tanke på forskjellige skrifttyper, grafikk og farger. Dette er bare noen få av de nye typene funksjoner som er implementert. Videre har også LaTeX blitt vanligere som en tekstbehandler på nett. Flere forskjellige nettsider tilbyr versjoner av LaTeX slik at man slipper å laste ned og sette opp sine egne versjoner lokalt.

Denne rapporten er utformet i LaTeX nettopp på grunn av LaTeX sin styrke når det kommer til formatering, robusthet, og omfattende litterære støtte. Malen som er brukt er basert på en mal fra Community of Practice for Computer Science Educators ved NTNU, som igjen er basert på tidligere maler av Simon McCallum, Ivar Farup og Kjetil Ørbekk. Gruppen var bevisst på at den ønsket å utforme rapporten på en måte som ville gi kvalitet i gjennomføringen, og LaTeX ble identifisert som

2.2.7 Kodestandard

Når man er flere på et prosjekt er det viktig med standarder for å holde orden. I avsnittet som følger vil vi gå gjennom ulike konvensjoner for å holde et ryddig prosjekt. Regler om navnekonvensjon satt for prosjektet er inspirert av «Design Guidelines for Class Library Developers» (Microsoft, 2006).

Varianter

- Pascal case
  - Første bokstav i hvert ord er stor, eksempel “FileStream”.
- Camel case
  - Første bokstav i hvert ord utenom første bokstav er stor, eksempel “fileStream”.
- Uppercase
  - Alle bokstaver er store, brukes bare når det er to eller færre bokstaver, eksempel “UI” og “IO”.

Server

Navnekonvensjonene i dette avsnittet beskriver serverapplikasjonen vår.

- Klasse
  - Skrives med pascal.
  - Beskrivende navn, unngå forkortelser.
  - Eksempel: “InputDecoder”.
- Datafelt
  - Skrives med pascal.
  - Om feltet er privat brukes understrek, “_Name”, for å få frem dette.
  - Beskrivende navn, unngå forkortelser.
  - Eksempel: ‘Name”,‘FirstName”.
- Metoder
  - Skrives med pascal.
  - Metodenavn skrives som verb, unngå forkortelser.
  - Eksempel: ‘RemoveAll”,‘GetAll”.
- Parameter
  - Skrives med camel.
  - Beskrivende navn, unngå forkortelser. Navnet skal beskrive hva det skal være enn hva slags type det er.
  - Eksempel: ‘typeName”,‘index”.

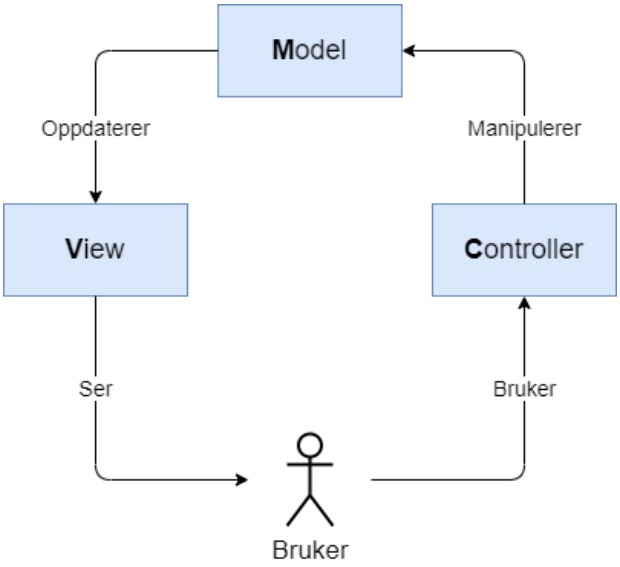
- Interface
    - Skrives med pascal.
    - Bokstaven ‘I” – for interface – i starten av navnet.
    - Beskrivende navn, unngå forkortelser.
    - Eksempel:‘IComponent”.

Designmønster

Designmønster er løsninger på generelle problemer utviklere støter på i utviklingen (Aroraa & Chilberto, 2019). Disse er bygget på erfaring om hva som funker og hva som ikke funker. Fordelen med å bruke et designmønster i utviklingen, er at vi ikke prøver å finne løsninger på ting som allerede er løst. I tillegg gjør det at applikasjonen vi utvikler får en forutsigbar struktur og kan skaleres.

MVC MVC står for model view controller og beskriver et designmønster der applikasjonen deles opp i de tre delene. I model definerer vi domenemodellen til applikasjonen vår. Modellene vi lager i model inneholder ingen referanser til noe data, og skal beskrive entitetene i applikasjonen. Views skal presentere dataen i HTML-filer. Siden vi kommer til å lage en web-service i form av et API, har vi ikke views, men viser heller serialisert data i JSON. I controllerene håndteres forespørsler fra bruker, og er ofte støttet av en tjeneste (Resca, 2019).

Figur 2.3 illustrerer hvordan MVC-arkitekturen henger sammen. En bruker benytter en nettside. Her oppretter, endrer eller vises brukeren data gjennom en controller. Controlleren oppdaterer gitte modeler som sendes til view som vises for brukeren. Med andre ord holder modeller på data, view viser frem data og kontrollere flytter på data.

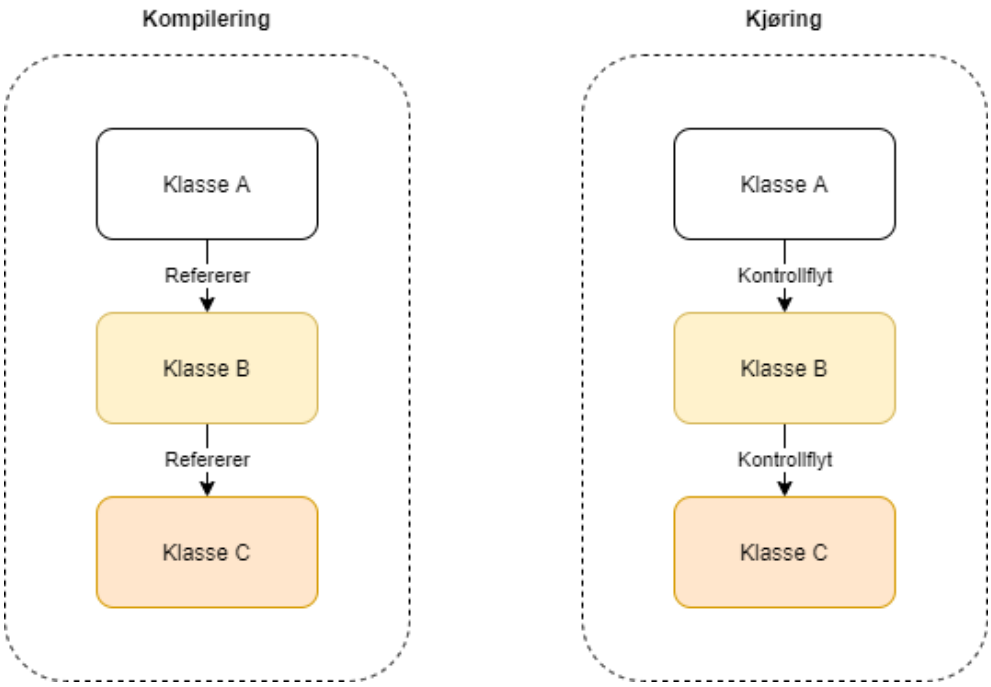


Figur 2.3: MVC-diagram

Dependency Inversion Siden applikasjonen vår kom til å ha flere avhengigheter, valgte vi å bruke dette prinsippet i deler av applikasjonen vår. Dette er et designmønster som er en del av SOLID-prinsippene, som er et av de mest innflytelsesrike designprinsippene i programvareutvikling. De består av følgende 5 prinsipper:

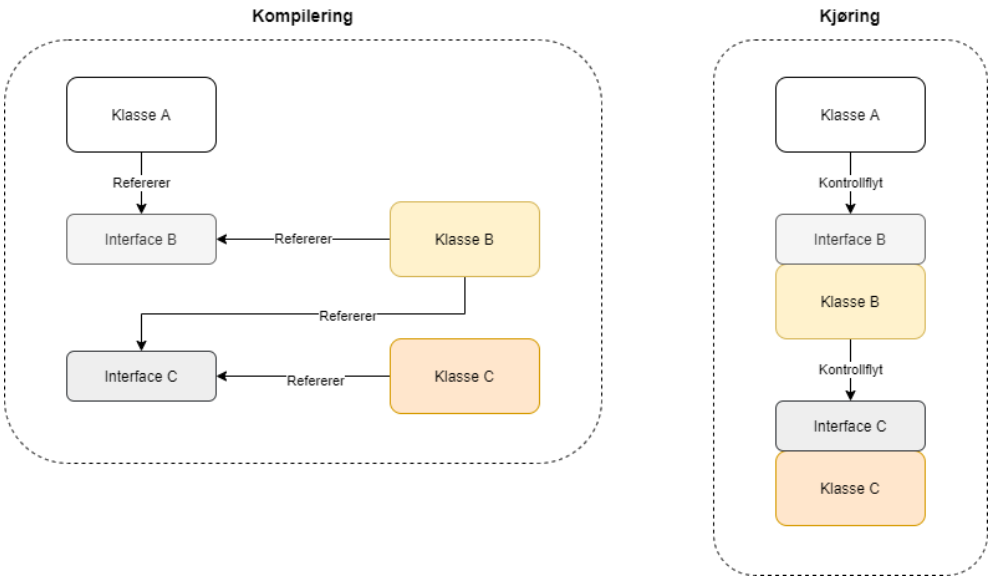
- **S**ingle responsibility principle
- **O**pen/Closed principle
- **L**iskov substitution principle
- **I**nterface segregation principle
- **D**ependency inversion principle

Dependency inversion handler om at retningen til en avhengighet i applikasjonen skal rettes mot abstraksjonen, ikke detaljer i implementasjonen. Med en avhengighet, menes en del av programvaren som avhenger av en annen. I vår applikasjon er f.eks. en kontroller avhengig av en tjeneste for å fungere som den skal. Tradisjonelt er de fleste applikasjoner bygd opp slik at avhengigheten av kompileringstiden flyter i samme retning som kjøringen av applikasjonen. Altså hvis modul A kaller på en funksjon i modul B, som igjen kaller på en funksjon i modul C, vil A avhenge av modul B som igjen vil avhenge av modul C, når applikasjonen kompileres, som vist i figur 2.4 (Wenzel mfl., 2019).



Figur 2.4: Direkte avhengighet

Hvis vi her anvender dependency inversion-prinsippet, lar vi A kalle metoder på abstraksjonen som B implementerer, altså at A kaller på metoder fra interface B, heller enn klasse B. Dette gjør det mulig for A og kalle på B under kjøring, men at B avhenger av et interface som er kontrollert av A under kompilering, som vi viser i figur 2.5. Under kjøring forblir flyten den samme, men implementeringen av interfacene gjør at andre implementasjoner av interfacene lett kan implementeres (Wenzel mfl., 2019).



Figur 2.5: Invertert avhengighet

Resultatet av dette er at applikasjonen vi bygger blir mer testbar, modulær og mulig å vedlikeholde. Dette gjør også at vi kan innføre dependency injection i applikasjonen (se 4.3.6).

Klient

Komponentene i React samles under mappen "components". Her har hver komponent sin egen mappe som inneholder selve komponenten og dens CSS-fil. React-komponenter skrives med pascal. For eksempel "Dashboard" og "TimeOverview".

For hver komponent i React opprettet vi egne CSS-filer. Ved å gjøre det på denne måten slipper vi en stor fil med all kode og får samlet koden som hører til den aktuelle komponenten, Siden CSS-klassene er tilgjengelige på tvers av komponentene, er det viktig med unike navn på klassene. For å løse dette benyttes navnet på komponenten i starten av klassenavnet, for eksempel "card-container", "card-title" og "popup-title". Om en CSS-klasse skal være tilgjengelig for alle komponenter legges den i "App.css".

2.2.8 DevOps

DevOps, development (utvikling) og Operations(operasjoner) er en forening av prosessen som skjer når man skal levere verdi for kunde kontinuerlig (Microsoft, 2020a). Bakgrunnen for DevOps er et ønske om å integrere utvikling og operasjoner ved bruk av automatisert utvikling, implementering og overvåkning av infrastruktur. I stedet for å dele opp utvikling og operasjoner i hver sine grupperinger

lar denne arbeidsmetodikken gruppene samarbeide for å øke problemløsning og minske kommunikasjonsproblemer. (Ebert, Gallardo, Hernantes & Serrano, 2016)

2.2.9 Sikkerhet i DevOps

En del av utviklingen som ofte glemmes i DevOps er sikkerhet. Dette skjer ofte fordi utviklere ønsker kontinuerlige endringer, og operasjon ønsker en stabil leveranse, og når de to skal integreres er det sjeldent plass til sikkerhet nettopp fordi det legger store begrensninger. Derfor kan en bedrift fort befinne seg i en situasjon hvor man ser bort fra sikkerhetsproblemer under utvikling og deployering, og heller tenker at dette skal fikses før løsningen rulles ut. Det er vanskelig å ta igjen sikkerhetsproblemer når man ikke har hatt den tankegangen fra start, og det er heller ikke sikkert det er tid i det hele tatt til å fikse dette. Dermed oppstår problemet hvor man egentlig må skylde på hele DevOps-prosessen for at man har sikkerhetsbrister. Dette er i bunn og grunn et problem som knyttes til kultur i selskapet hvor man jobber med DevOps, og noe som må endres for å kunne fortsette utviklingen av DevOps-prosessene. (Marstrander & Hovind, 2020)

Arbeidsflyt

For å holde kontroll på all koden benytter vi oss av versjonskontrollverktøyet Git (se 2.2.4). Git fungerer på den måten at man lager greiner av kodebasen der man for å tilfører ny kode. Gjennom oppdragsgivers infrastruktur i Azure DevOps (se DevOps) fikk vi tildelt oppbevaringssted for kodebasen til prosjektet. På denne måten skiller man nye versjoner fra fungerende kode og velger selv når man ønsker å flette disse sammen. Ved sprintstart oppretter gruppemedlemmene hver sin gren som de jobber på gjennom sprinten. Når sprinten er ferdig fletter man disse sammen til utviklingsgrenen, dev, og de ulike delene testes sammen. Før grenene flettes, tester eier av grenen at koden kjøres ved å kjøre den gjennom pipelinen som er satt opp (se 4.2.1).

Ved å kjøre koden gjennom pipelinen før sammenfletting vet man om programmet bygger og kjører, dette forhindrer å flette sammen kode som ikke kjører.

Når koden i dev er klar for å lanseres, fletter man denne med prod. Slik pipelinen er bygget opp havner koden først i et test-stadie. Dette er siste stopp før koden havner i produksjon og applikasjonen er “live”. For at koden skal komme i produksjon må testene gå gjennom og en annen person må godkjenne forespørselen.

2.3 7Pace Timetracker

2.3.1 Innledning

7Pace Timetracker er en tredjepartsløsning, integrert i Azure DevOps, som lar utviklere spore tid brukt på work items i Azure Devops (Copeland, 2019). Et work item er enten en business epic, epic, feature, user story eller task som er element

opprettet i Azure DevOps, backlog. Elementene har barn i gitt rekkefølge, altså har en Business Epic flere Epics, en Epic har flere Features og så videre. En worklog er en logg av endringer knyttet til et work item. For eksempel om man jobber med en task over flere dager vil man få en worklog for hver dag man registrerer timer på tasken. Et work item kan ha null eller flere workloger.

2.3.2 Reporting API

Med nyeste versjon av Timetracker tilbys Reporting v3.0 API-et som blant annet gir tilgang til arbeidsoppgaver og data knyttet til tid bruk på oppgavene. API-et er bygget på OData-rammeverket som er en standard protokoll for å opprette og konsumere data (se 2.3.3).

For å autorisere bruk av API-et benyttes OAuth 2.0. Token genereres i Azure DevOps, i instillingene til 7Pace TimeTracker. Mer om OAuth 2.0 i kapittel 2.3.4.

API-et har flere endepunkt, disse er dokumentert i «7pace Timetracker Overview» (Copeland, 2019). API-et er bygd opp slik at en velger endepunkt ut i fra hvor mye data en er interessert i, for å begrense unødvendig data. Endepunktet "workItemsHierarchyAllLevel", gir data om work items samt mulighet for å hente informasjon om barn- eller forelder-element, mens endepunktet "worklogsOnly" er det raskeste som bare gir data relatert til worklog uten å koble sammen med work items. I løsningen vil vi bruke flere av disse endepunktene ut fra hvilke data vi trenger. I navnet på de ulike endepunktene kommer det tydelig frem om det er work item eller worklog som er objektet som blir returnert.

Siden API-et er bygget med OData rammeverket, brukes OData sine parametere for filtrering og henting av data. "Select" brukes for å definere hvilke felter som ønskes i de returnerte objektene. "Filter" lar en gi regler for hvilke objekt som skal returneres. F.eks. kan vi legge til at id $= 1$ i filteret og da returneres bare objektet med id 1. Disse parameterne brukes for å bygge de spørringene som er ønskelig.

```
http://innovationnorway.timehub.7pace.com/api/odata/v3.0/workLogsWorkItems?
$filter=WorkItemId eq 22861
```

Kodeliste 2.1: Kall for å hente ut worklog-en til et gitt work item gjøres ved følgende URL

Et noe forkortet resultat ser da slik ut:

```
{
            "WorkItemId": 22861,
            "Timestamp": "2020-02-17T00:00:00.67Z",
            "PeriodLength": 1800,
            "EditedTimestamp": "2020-02-17T10:42:06.663Z",
            "CreatedTimestamp": "2020-02-17T10:42:06.663Z",
            "User": {
                "Name": "Håkon Åreskjold",
                "Email": "s326169@oslomet.no"
```

```
        },
        "ActivityType": {
            "Name": "IT Development"
        },
        "AddedByUser": {
            "Name": "Håkon Åreskjold",
            "Email": "s326169@oslomet.no"
        },
        "WorkItem": {
            "System_Id": 22861,
            "System_Title": "Tegne ER ",
            "System_WorkItemType": "Task",

            "Microsoft_VSTS_Scheduling_OriginalEstimate": 3.0,
            "Microsoft_VSTS_Scheduling_RemainingWork": null,
            "Microsoft_VSTS_Scheduling_Size": null,
        },
        "EditedByUser": {
            "Name": "Håkon Åreskjold",
            "Email": "s326169@oslomet.no"
        },
        "WorklogDate": {
            "Year": 2020,
            "Month": 2,
            "Day": 0,
            "ShortDate": "2020-02-17"
        }
    }
```


Kodeliste 2.2: Resultat etter API-kall

Postman

Postman er en programvare for spørring og testing av APIer. I vårt prosjekt har vi brukt programvaren for å utforske API-et på en enkel og effektiv måte. Programmet er gratis og virker på macOS og Windows.

2.3.3 OData

Ifølge «Open Data Protocol (OData)» (Garg, 2020) er OData sitt formål å levere en protokoll som er basert på REST-arkitekturen for CRUD-operasjonene. Fordelene med OData er at det tillater utviklere samhandle med dataen ved bruk av RESTful web service. OData har støtte for CRUD gjennom POST, PATCH, PUT og DELETE som HTTP-protokollen gir oss.

I tillegg til de gitte endepunkter, tillater OData oss å legge til parametre på spørringene. For eksempel filter, orderby, expand og select. Dette er parametre som lar oss tilpasse hvilke data vi skal få i retur. Se API-kall 2.2 for eksempel.

2.3.4 OAuth 2.0

OAuth 2.0 er en protokoll som tillater bruker å gi tilgang fra en maskin til en annen, uten å gi ut noe ekstra informasjon. OAuth 2.0 deler inn i fire roller. (OAuth, 2020)

1. Ressurseier
    - Personen som gir tilgang til litt av kontoen sin. Ressursene er dataen det gis tilgang til, tjenestene kontoen tilbyr eller andre ting tjenesten gjør via kontoen til personen.
2. Ressursserver
    - Serveren som har brukerdataen som blir aksessert fra tredje-parts applikasjonen. Serveren må kunne akseptere og validere tilgangsnøkkelen og gi tilgang til det som brukeren har gitt tillatelse til å bruke. Serveren trenger ikke nødvendigvis kjenne til applikasjonen.
3. Klient
    - Applikasjonen som handler på vegne av brukeren og spør/trenger ressursene. Før applikasjonen kan få tak i ressursene kreves det tilgang av brukeren. Klienten spør etter godkjenning enten direkte fra brukeren, eller fra authorization serveren.
4. Autorisasjonsserver
    - Serveren som autentiserer ressurseieren og utsteder tilgangsnøkkel. I dette tilfellet OAuth.

Tilgangsnøkkelen til API-et genereres i 7Pace Timetracker sine innstillinger. Det er denne som gjør at vi får lov til å koble oss til API-et vi ønsker å bruke. Innovasjon Norge er ressurseier, ressursserver er API-et, applikasjonen som utvikles er klienten, og OAuth er som sagt autorisasjonsserver.

2.4 Sikkerhet

Siden trusselbildet i dag er preget av applikasjoner og kontakt med internett, så er også de fleste systemer avhengig av beskyttelse mot trusler fra internett. Mange systemer har innebygde beskyttelsesmekanismer, men for hver oppdatering så åpnes det muligheter for svakheter og sårbarheter som kan utnyttes av trusselaktører. Dermed var det viktig for oss å hele tiden ha sikkerhet i bakhodet når vi utviklet applikasjonen. Innovasjon Norge arrangerte et foredrag om sikkerhet holdt av Miguel Figueroa-Calix, som er ansvarlig for sikkerhet, risiko og etterlevelse. I tillegg deltok også gruppemedlemmene på et faglig seminar om sikkerhet knyttet til webapplikasjoner og DevOps, arrangert av mnemonic AS. Dette for å øke gruppemedlemmenes kunnskaper knyttet til sikkerhet og beste praksis når det kommer til sårbarheter i webapplikasjoner og DevOps.

Hvert år gjennomfører Verizon en sikkerhetsrapport som måler temperaturen i sikkerhetsverden og undersøker hvilke trusler som er mest aktuelle. Ifølge rapporten (Verizon, 2020) er 43% av alle innbrudd knyttet til webapplikasjoner, noe som er en dobling fra i fjor. Dette peker på viktigheten av å sikre applikasjonen mot trusler fra internett, og derfor ønsket gruppen å gjennomføre en sikkerhetstest av applikasjonen i etterkant av utviklingen. Det som dog viste seg å være en utfordring knyttet til dette er måten applikasjonen vår er satt opp på. Gruppen rådførte seg med sikkerhetsanalytikere hos IT-sikkerhetsfirmaet mnemonic AS i forhold til hvordan en slik test trygt kan gjennomføres siden det ikke var hentet inn tillatelser til å sikkerhetsteste hverken API'et til 7Pace TimeTracker eller Innovasjon Norge, og heller ikke fra Microsoft som betjener innloggingen til applikasjonen via Innovasjon Norge. Analytikeren foreslo å kun teste GET request som applikasjonen foretok, men problemet som da kunne oppstå var at applikasjonen sender GET request fra frontend mot backend som henter data via 7Pace TimeTracker API'et. Dermed kunne vi ikke være sikker på at testen ikke ville trigge Interne deteksjons- eller beskyttelsesmekanismer i nettverket, og dette var ikke ønskelig. Vi visste heller ikke om dette kunne føre til nedetid hos kunden og dermed et potensielt erstatningskrav. Gruppen undersøkte derfor med Miguel Figueroa-Calix hvorvidt Innovasjon Norge utførte sikkerhetstester og fikk en oversikt over hvordan man håndterer dette internt. Gruppen besluttet å overlate sikkerhetstesting av applikasjon til Innovasjon Norge slik at de kunne styre når og hvor dette skulle gjennomføres, og heller utføre en sikkerhetsanalyse av applikasjonen. Sikkerhetsanalysen finnes som vedlegg C.

2.5 Utviklingsprosess

2.5.1 Sprinter

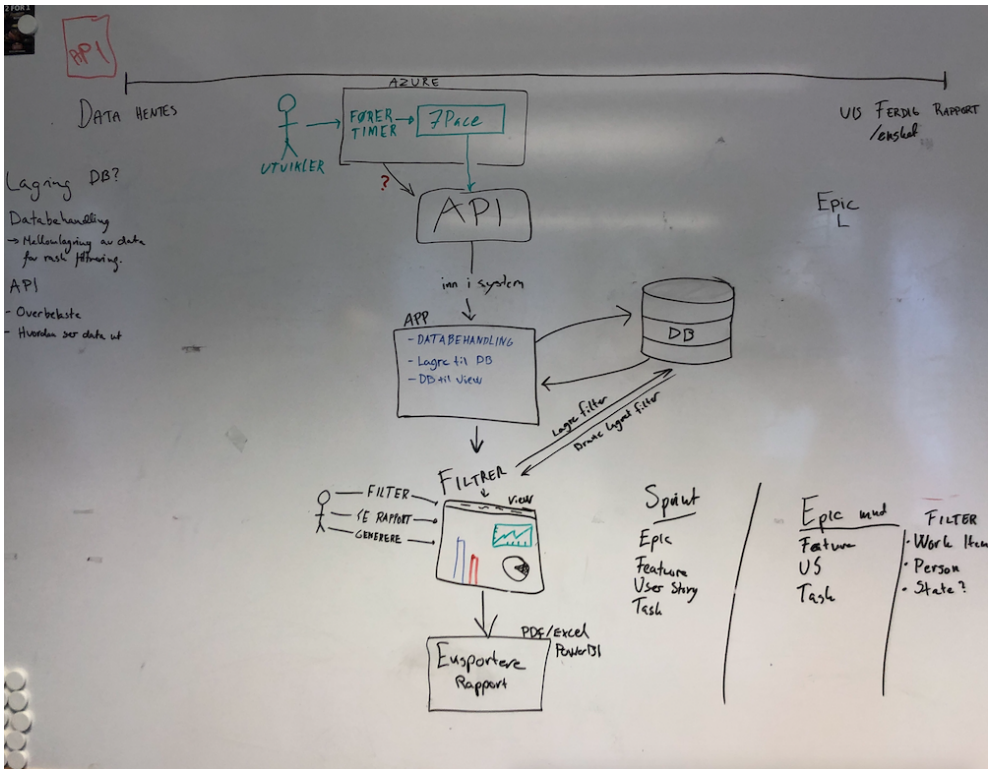
Som en del av Scrum-metodikken arbeidet gruppen i sprinter, for å jobbe i kortere iterasjoner med utviklingen. Dette gjorde at gruppen kunne få tilbakemeldinger på arbeidet underveis, og omprioritere hvis noe av arbeidet skulle vise seg overflødig eller unødvendig. Det ble gjennomført 6 sprinter, med en varighet på 1-2 uker. Det var spesielt de første ukene at sprinter på 1 uke var nødvendig siden det var stort behov for å kunne omstille seg hurtig.

Sprint 1

27. januar - 9. februar

I første sprint hadde gruppen følgende spørsmål som den ønsket svar på; "Hvem skal bruke applikasjonen?", "Hva har brukerne bruk for?", "Hvilke type data hentes i applikasjonen og fra hvor?", og "Hvilke data skal lagres?". Det ble sendt meldinger til teamledere, prosjektledere og avdelingsledere, for å få deres tilbakemelding på hva som var viktig for dem i en slik applikasjon. For å få en bedre forståelse over produktet vi skulle utvikle skisserte gruppen opp de ulike kompo-

nentene som vi mente trengtes (figur 2.6). Ved å visualisere komponentene gav dette et bedre bilde av hva det endelige resultatet skulle bli, og gjorde det enklere å jobbe med kravspesifikasjonen. Dette var også viktig for å få en felles forståelse om strukturen og innholdet i produktet.



Figur 2.6: Bilde fra brainstorming

Første sprint gikk parallelt med at vi arbeidet med ny oppgave. Noen dager i forkant av sprinten fant gruppen ut at Innovasjon Norge allerede hadde rapporteringsverktøy utviklet av 7Pace TimeTracker selv, noe som gjorde vår oppgave overflødig. Veileder hos Innovasjon Norge mente først at det var positivt siden vi da hadde en mal, men usikkerhet gjorde at gruppen ønsket bekreftelse på at oppgaven skulle bestå. Gruppen kontaktet veileder og avdelingsleder hvor tilbakemelding fra avdelingsleder var at oppgaven måtte redefineres. Dermed startet arbeidet med redefinering av oppgaven. Her ble også teamkoordinatorer i Innovasjon Norge involvert. 04. februar fikk vi innspill til ny kravspesifikasjon av Innovasjon Norge. Denne kom ikke fra teamkoordinatorer hos Innovasjon Norge, men var noe som vår veileder og en annen veileder presenterte for oss som en mulig oppgave.

Oppgaven ble ikke ferdig definert i denne sprinten, men vi kom i gang med ny kravspesifikasjon som skulle definere oppgaven.

Sprint 2

11. februar - 24. februar

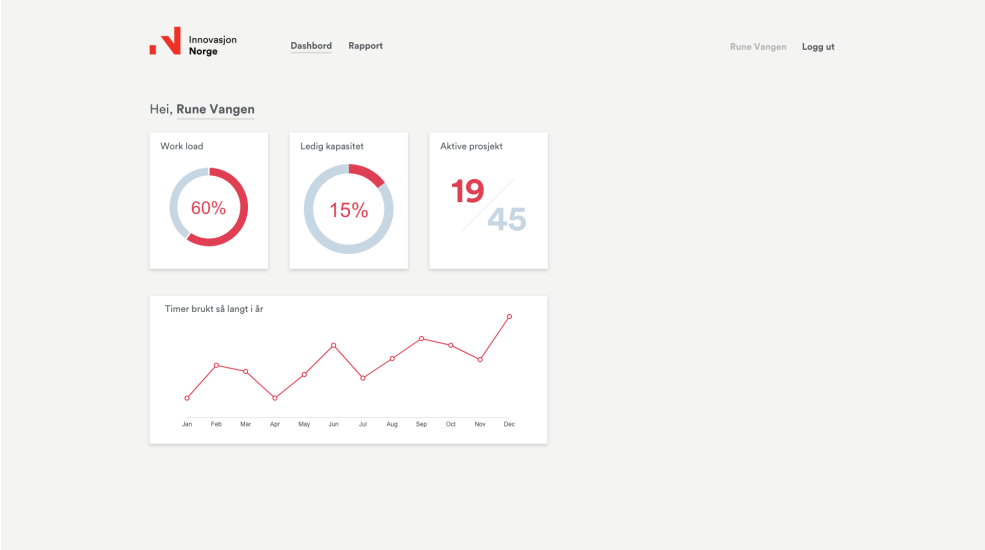
I løpet av de første dagene i sprinten fikk vi på plass ny kravspesifikasjon. Når gruppen hadde klar kravspesifikasjonen ble denne gått gjennom sammen med veileder som godkjente. Gruppen kunne da ta fatt på oppgaven.

Denne sprinten gikk til utforsking av APIet til 7Pace TimeTracker, oppsett av repositories, skriving om planleggingsverktøy og planlegging av "oppgavefordeling" mellom klient og server.

Å bli kjent med APIet til 7Pace TimeTracker var tidkrevende, da APIet er bygget med OData (se 2.3.3), noe som var nytt for gruppemedlemmene. Hvordan kunne vi hente ut data, og hvordan kunne vi vite at den dataen vi hentet var "riktig"? Vi hadde ikke noen mulighet for å kryssjekke at dataen vi hentet var all tilgjengelig data som var reell for fremstilling. Vi sendte flere e-poster til 7Pace TimeTracker da vi opplevde at et av endepunktene og mulighet for å hente forelder-element ikke virket slik det skulle. Dette ble ordnet etter kort tid. Innovasjon Norge var ikke kjent med APIet og kunne ikke hjelpe oss annet enn å henvise til dokumentasjon.

Infrastruktur for repositories og release ble ferdigstilt i denne sprinten. Innovasjon Norge ønsket at alt av repositories skulle være knyttet til Azure DevOps, og dette måtte være på plass før vi kunne ta fatt på utviklingen. Her fikk vi god hjelp fra en annen veileder, Tommy Bakken Mydland, som hadde god kunnskap rundt disse verktøyene.

Planleggingen av hvilke funksjonaliteter som skulle legges til serveren og hva som skulle til klienten ble også utført i denne sprinten. Her dukket spørsmål om i hvilken grad dataen, hentet fra APIet til 7Pace TimeTracker, måtte behandles før den ble visualisert. Vi diskuterte også om vi skulle mellomlagre dataen i database, og så på hvilke fordeler og ulemper en database ville medbringe. Vi hadde et ønske om at dataen som ble brukt i applikasjonen alltid skulle være den siste oppdaterte, og konkluderte derfor at det var like greit å hente direkte fra 7Pace TimeTracker. Et argument for å mellomlagre data er om 7Pace TimeTracker skulle være nede. Da 7Pace TimeTracker daglig er i bruk av alle utviklerne ved Innovasjon Norge som rapporterer sine timer, konkluderte vi med at APIet var stabilt nok.



Figur 2.7: Prototype av webapplikasjon

I denne sprinten undersøkte vi også om vi kunne få tilgang til React-komponent-biblioteket til Innovasjon Norge, som er et bibliotek hvor Innovasjon Norge har bygget en rekke komponenter som kan brukes i webapplikasjonen. Dette skulle la seg ordne så snart de fikk gitt oss tilgang. Samtidig ble det utarbeidet en prototype av webapplikasjonen som skulle tjene som inspirasjon i utviklingsfasen (figur 2.7). Prototypen ble vist til veileder som mente det var et godt utgangspunkt.

Det var viktig for gruppen at problemer ble løftet opp tidlig slik at konflikter ikke skulle bygge seg opp. Det var ved noen anledninger tilløp til støy i lokalet hos Innovasjon Norge som var satt av til bachelorgruppene. Gruppen opplevde det som ubehagelig å måtte ta opp situasjonen med veileder, men gruppen ønsket ikke at dette skulle være et vedvarende problem gjennom prosjekttiden. Dette problemet ble tatt opp med veileder hos Innovasjon Norge, noe som førte til endringer i arbeidsmiljøet og det ble roligere i lokalet.

Sprint 3

25. februar - 9. mars

Vi startet i denne sprinten med utviklingen. Gruppen begynte med oppsett av backend, samtidig som vi startet med å sette opp webapplikasjonen. Oppsettet sørget for at applikasjonene snakket sammen og miljøet til webapplikasjonen benyttet React med Typescript som språk. TypeScript er et open source programmeringsspråk som er utviklet og ivaretatt av Microsoft. Språket er inspirert av JavaScript, Java og C. TypeScript ble utviklet som et alternativ til JavaScript og er mer egnet til større applikasjoner, noe som gjorde språket godt egnet vårt prosjekt samt for videreutvikling. TypeScript transkompileres til JavaScript ved kompile-

ring (Bright, 2012). Språket er strengt typet, som vil si at man definerer om en variabel er f.eks. tekst, tall eller sann/usann, noe som er forskjellig til JavaScript. At TypeScript er strengt typet, sammen med at koden transkompileres, hindrer minsker risikoen for at applikasjonen kjører med feil. TypeScript var forholdsvis nytt for medlemmene da prosjektet startet. Språket er nokså likt JavaScript, og medlemmene er vant til å programmerere i sterkt-typede språk, men det krevdes likevel en del tid for å bli vant til syntaks kontra JavaScript.

Vi startet også å se på sikkerheten mellom server og klient, dette for å legge grunnlaget for implementering av funksjonaliteten. Selve implementasjon var planlagt til senere sprinter, men utforsking ble utført i denne sprinten. Da webapplikasjonen og serverapplikasjonen er to selvstendige applikasjoner var vi usikre på hvordan autentiseringen skulle settes opp. Kommunikasjonen mellom server og klient måtte sikres, samtidig som bare personer som er innlogget skulle få tilgang til webapplikasjonen. Planen var å sikre kommunikasjonen mellom server og klient med bruk av JSON Web Tokens som er en standard for sikker kommunikasjon av JSON-objekter (Auth0, 2020). Til innlogging skulle vi bruke Azure AD.

I tillegg jobbet gruppen videre med å undersøke 7PTT APIet for å forstå hvilke data som kunne hentes ut og hvordan. Dette var mer tidkrevende enn først antatt.

Sprint 4

10. mars - 23. mars

Vi jobbet med å utvikle en datahenter i backend. Datahenteren er hovedkomponenten i serverapplikasjonen da det er den som sørger for at vi får inn data fra 7Pace TimeTracker. Dette var en krevende oppgave da datahenteren måtte kunne hente inn ulik type data avhengig av hvilke spørringer gruppen ønsket å utføre. Her måtte vi ta i bruk generiske klasser for å kunne mate dataen inn i representative modeller. Kodeliste 2.3 viser hvordan klassen brukes. `WorkItem` er modellen vi forventer at spørringen skal returnere. Variabelen `resultat` blir da en liste av `WorkItem`-objekter.

```
var fetcher = new Fetcher();

await fetcher.Get<WorkItem>(EndPointEnum.WorkItems, query);

var resultat = fetcher.GetResponse<WorkItem>();
```

Kodeliste 2.3: Viser hvordan Fetcher brukes

Ut fra kravspesifikasjonen definerte vi tre forskjellige scenarier som vi ønsket å fremstille grafisk, som vi jobbet med videre:

- Vise data som viser faktisk tid brukt mot estimert tid.
- Vise Epics med antall Features og UserStories nødvendig for leveranse.

- Tid brukt fra rapportert bug til “Closed”.

Disse ble fordelt gruppemedlemmene i mellom og systematisk jobbet med. Spørringen mot 7Pace TimeTracker-APIet måtte defineres, for så å behandle dataen i serveren før den ble visualisert i webapplikasjonen.

Det var i denne sprinten Norge for alvor ble rammet av tiltakene knyttet til Covid-19, og vi måtte gjøre store endringer i måten vi jobbet på. Gruppen gikk over til å jobbe på hjemmekontor, og forsøkte å holde på rutinene som tidligere var fastsatt i samarbeidskontrakt (vedlegg B) ved oppstart. Rutinene ga gruppen god drahjelp, men det var også tydelig at gruppemedlemmene var preget av usikkerheten som dette førte med seg. Denne sprinten regnes nok som en av de mindre produktive, med tanke på synlig resultat ved sprint retrospekt.

Sprint 5

24. mars - 6. april

I denne sprinten fortsatte utviklingen rundt de ulike scenarioen som vi definerte. Samtidig måtte datahenteren skrives om, for å hente all data knyttet til en spørring. Gruppen opplevde at en spørring kunne inneholde flere “sider” med data. Datahenteren måtte derfor rekursivt utføre en ny spørring om resultatet inneholdt informasjon om at det fantes flere “side”. Som kodeliste 2.4 viser får vi informasjonen om neste side i form av `@odata.nextLink`, dette er URLen til neste forespørsel som klassen rekursivt kaller dersom den eksisterer i resultatet.

```
{
    "@odata.context": "innovationnorway.timehub.7pace.com/$metadata",
    "value": [
        {
            "Eksempel": "data",
        }, ...
    ],
    "@odata.nextLink": "innovationnorway.timehub.7pace.com/$skip=100"
}
```

Kodeliste 2.4: Viser struktur på API-responsen

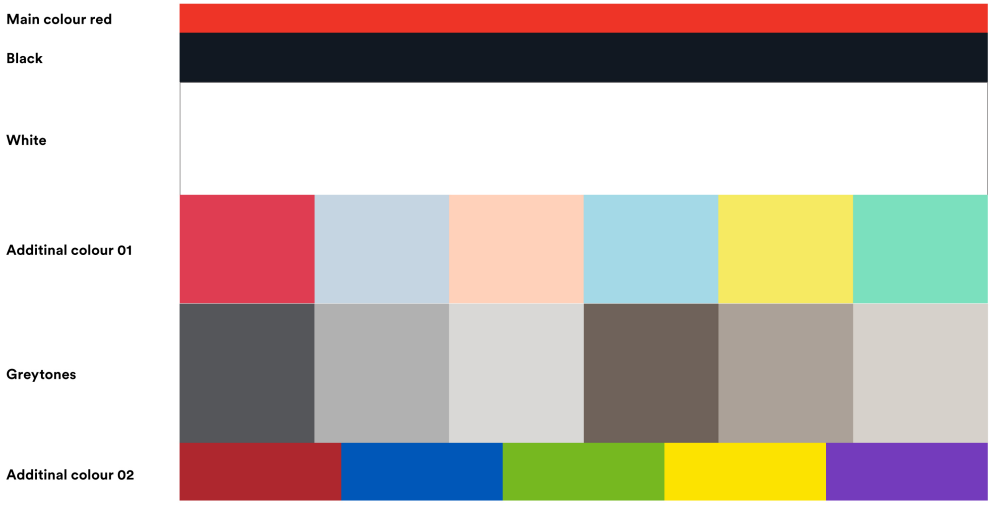
Etter to måneder med forespørsler ble det konkludert med å droppe React-komponent-biblioteket til IN, da tilgangen uteble og gruppen måtte komme videre med applikasjonen. Vi ble nødt til å utvikle komponenter selv der planen var å bruke Innovasjon Norge sitt bibliotek. Dette var oppgaver som ikke var regnet inn i tidsregnestykke, men som måtte gjøres plass for. Vi ønsket fortsatt å designe nettsiden slik at den matcher INs nettside. På dette tidspunktet økte også fokuset på sluttrapporten da tekst måtte flyttes fra støttedokumenter og inn i sluttrapporten.

Sprint 6

14. april - 28. april

Grunnet påskeferie fikk gruppen et opphold etter sprint 5. Før påsken delegerte deltagerne oppgaver som man jobbet med i påsken, men på grunn av forskjellige forpliktelser ble det vanskelig å avholde faste møter. Derfor valgte gruppen en fleksibel løsning i påsken, men passet på at man opprettholdt fremgangen. Gruppen arbeidet videre med data som skulle visualiseres, og det ble arbeidet med webapplikasjonens komponenter, som meny og innlogging samt design og farger.

I Innovasjon Norge's profilhåndbok (Innovasjon Norge, 2020) kommer det tydelig frem hvordan farger skal brukes og hvilke som er primær, sekundær og tertiær. Å velge farger fra denne paletten til bruk i grafer var utfordrende. Ifølge «Kva seier forskrifta?» (Digitaliseringsdirektoratet, 2020b) omfattes ikke denne applikasjonen av retningslinjene for universell utforming på nett da det er et internt verktøy. Likevel er dette gode retningslinjer for å få en applikasjon som er tjenlig for folk flest. Formålet med retningslinjene er å sikre alle brukergrupper en god opplevelse på nett. Vi har spesielt tatt hensyn til dette ved valg av farger til grafer og illustrasjoner.



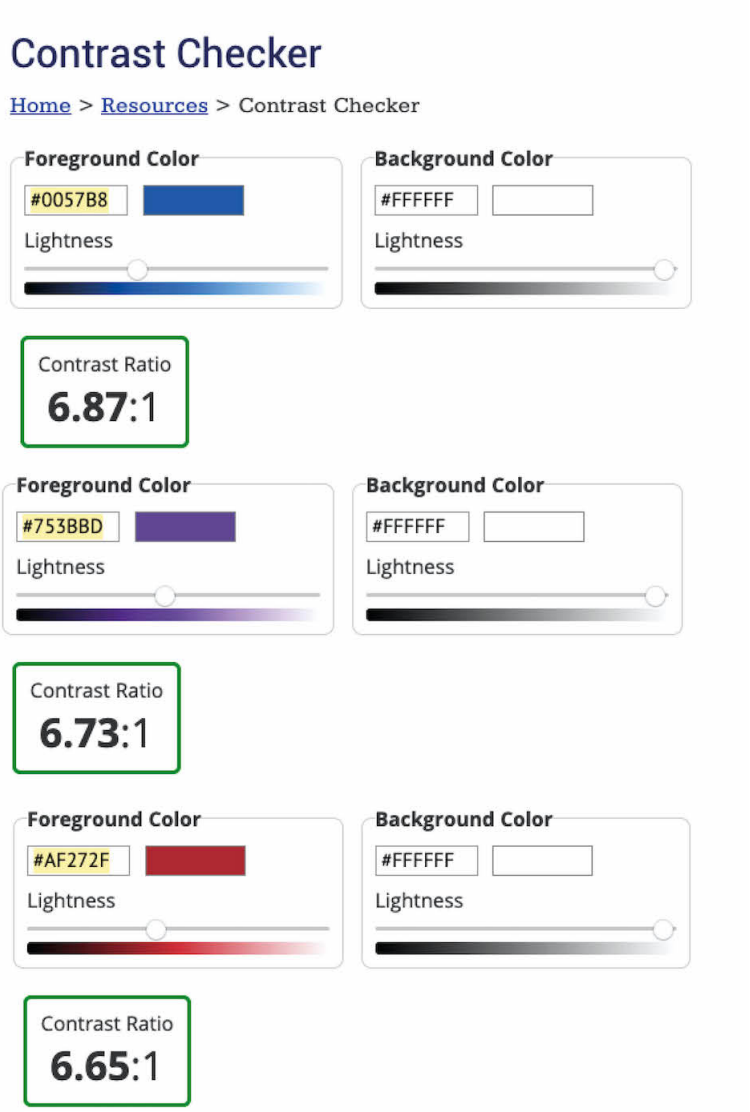
Figur 2.8: Vekting av fargene til Innovasjon Norge.

I «1.4.11 Kontrast for ikke-tekstlig innhold (Nivå AA)» (Digitaliseringsdirektoratet, 2020a) henvises det til nye krav i EUs webdirektiv som enda ikke har trådd i kraft. Planen var likevel å ta hensyn til disse kravene da de etter planen skulle tre i kraft 1. juni 2020. Disse er i skrivende stund utsatt. (Digitaliseringsdirektoratet, 2020c)

"I diagram, regnes hver linje, stolpe, sektor eller lignende i diagrammet

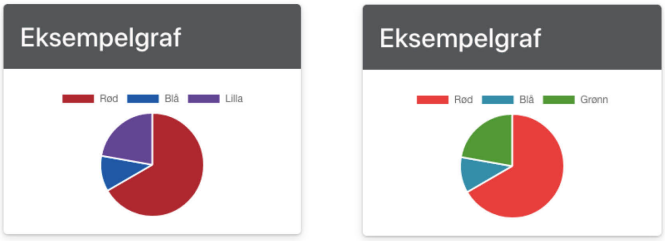
> som et grafisk objekt. Det samme gjelder eventuelle bakgrunnslinjer i diagrammet. Disse objektene skal ha kontrast på minst 3:1 mot bakgrunnen. I for eksempel linjediagram, der linjene i liten grad overlapper hverandre, er det ikke et krav om at linjene skal ha kontrast mot hverandre. Kravet gjelder kun kontrast mellom grafiske objekter som ligger rett ved siden av hverandre (i direkte tilknytning)."
>
> - (Digitaliseringsdirektoratet, 2020a)

Vi ønsket tre farger som det er godt samsvar mellom og som tilfredsstiller kravene om kontrast på 3:1 mot bakgrunnen. For å finne farger hos Innovasjon Norge som tilfredsstiller, må vi nederst på vektingen av Innovasjon Norge's farger som er farger til bruk av detaljer. Tre av de fem fargene er over 3:1-kravet, som vist i figur 2.9.



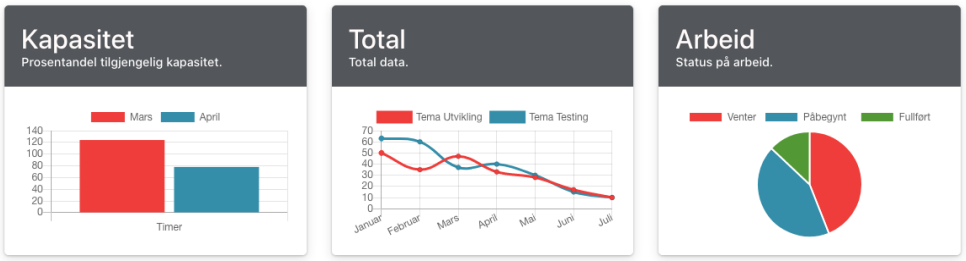
Figur 2.9: Farger fra Innovasjon Norge for detaljer som tilfredsstiller 3:1-kravet mot bakgrunn

Vi valgte derfor å tilpasse tre farger til bruk i grafer. Disse fargene er inspirert av Innovasjon Norge sin profil, men er tilpasset kravet om kontrast, som er 3:1 til bakgrunn.



Figur 2.10: Sammenligning mellom fargene til Innovasjon Norge som tilfredsstiller krav og farger inspirert av Innovasjon Norge sine farger som tilfredsstiller krav.

Figur 2.11: Farger brukt i grafer.



Figur 2.12: Eksempel på bruk av farger i grafer.

All tekst er enten hvit eller svart ut fra hvilken bakgrunnsfarge den er på. Ifølge den grafiske håndboken gir dette et stilrent og ryddig utseende.

I sprint 6 hadde vi et stort fokus på sluttrapporten for å sikre fremdrift i skriveprosessen.

Tiden etter siste sprint

Etter sprint 6 stoppet gruppen med sprinter. Fra og med 28. april var det fullt fokus på sluttrapporten samtidig som vi klargjorde webapplikasjonen slik at den var klar for demo. I denne tiden fikk gruppen tilbrakt den siste uken sammen for å sitte i samme rom når oppgaven skulle ferdigstilles. Dette gjorde at man kunne kommunisere enda mer effektivt, samt avklare utfordringer som dukket opp raskt.

Veiledning

Underveis i utviklingsprosessen, og gjennom sprintene hadde gruppen faste møter med intern og ekstern veileder. For å møte ekstern veileder var gruppen hver tirsdag hos IN. Der har vi kunnet stille spørsmål om det skulle være behov for det, og har fått assistanse ved tekniske utfordringer. Etter at Innovasjon Norge stengte kontorene har gruppen hatt kontakt ved behov, og kontakten har gått over til Teams. Gruppen fikk beskjed om å ta kontakt ved behov, ellers har veileder sjekket inn på oss et par ganger. Siden den avsatte dagen falt bort ble det mer utfordrende for gruppen å få avklaringer, noe som gjorde at gruppen måtte holde i mange forskjellige tråder mens man ventet på avklaring. Kommunikasjonen pågikk over Teams da dette var ønskelig fra ekstern veileders side, men det gikk ofte en uke før svaret kom.

Intern veiledning fortsatte i samme intervall, og her gikk gruppen over til videomøter på Teams. Gruppen fikk avklaringen på spørsmål knyttet til oppgaveskriving og innspill på arbeidsprosessen så langt. Gruppen fikk også inkludert veileder i Overleaf - en online kompilator for LaTeX - hvilket gjorde prosessen med tilbakemelding på skriving enklere.

2.6 Testing

2.6.1 Innledning

Testing handler om å teste om applikasjonen gjør som den skal. Programvare som ikke fungerer som det skal kan blant annet føre til problemer som tap av penger, tid og ryktet til virksomheten. I verste fall kan konsekvensene av å ikke teste et program være katastrofale. I «ISTQB CTFL Syllabus 2018 V3.1» (Olsen, Posthuma & Ulrich, 2018) beskrives programvaretesting som en måte å evaluere kvaliteten til programvaren og til å redusere risikoen for at programvaren feiler i produksjon. Hvor omfattende testingen skal være, kommer an på hva som er programvarens funksjon og hva det skal brukes til.

2.6.2 Planlegging av testing

Vi fikk i starten semesteret beskjed av Innovasjon Norge at applikasjonene vi utviklet skulle gjennom omfattende testing før de skulle ut i produksjon. Dette førte til at vi ikke satte opp en testplan, da Innovasjon Norge skulle stå for testingen. Siden Innovasjon Norge skulle teste applikasjonen, bygde vi den på en måte som legger opp til videre testing ved å følge dependency inversion-prinsippet (se 2.2.7) og implementere dependency injection (se 4.3.6). I tillegg testet vi kontinuerlig applikasjonen manuelt, mens vi utviklet.

2.6.3 Eksempel på enhetstesting av kontrollere

Applikasjonen vår er bygget med dependency injection. Dette forenkler testingen, da vi kan opprette falske tjenester med testdata som vi injiserer i kontrolleren vi skal teste (se 4.3.6). I kodeliste 2.5 ser vi en testmetode der vi tester at EpicController sin get-metode gir ønsket resultat.

```
[Fact]
public void Get_WhenCalled_ReturnsAllItems()
{
    // Act
    var okResult = _controller.GetEpics().Result as OkObjectResult;

    // Assert
    var items = Assert.IsType<List<SelectOptions>>(okResult.Value);
    Assert.Equal(2, items.Count);
}
```

Kodeliste 2.5: Viser testing av kontroller.

Kapittel 3

Kravspesifikasjon og dens rolle

En kravspesifikasjon er en oversikt over hvilke krav som stilles til applikasjonen eller systemet. Kravene beskriver funksjonalitet systemet skal tilby og begrensningene det har (Sommerville, 2016, s. 102). En utarbeidelse av kravspesifikasjonen er ofte første steg i å utvikle et nytt system (Sommerville, 2016, s. 104). Kravspesifikasjon er utarbeidet etter mal fra Innovasjon Norge og godkjent av ekstern veileder hos Innovasjon Norge. Kravspesifikasjonen er vedlagt som vedlegg D, og den tar for seg følgende punkter:

- Kort beskrivelse av prosjektet
- Mål
- Generelle krav
- Funksjonelle krav
- Funksjonelle krav risikovurdering
- Ikke-funksjonelle krav
- Ikke-funksjonelle krav risikovurdering

Utdrag fra kravspesifikasjon vil bli trukket frem i dette kapittelet, for å drøfte hvorvidt oppnådd resultat samsvarer med kravspesifikasjon. Det vil også bli sett på endringer som oppstod i kravspesifikasjonen, samt hva kravspesifikasjonen har betydd for utvikling av produktet.

3.1 Sammendrag av krav

Målet var å lage en webapplikasjon som fremstiller tid brukt på prosjekt for å effektivisere utviklingsprosessen til IT-avdelingen, samt som analyserer og visualiserer tidsbruken for ansatte ved Innovasjon Norge sin IT-avdeling.

Kravspesifikasjonen er endret helt fra den opprinnelige oppgaven, siden gruppen oppdaget at de kravene som ble stilt allerede hadde en løsning i tredjepartsproduktet, 7Pace TimeTracker, som Innovasjon Norge brukte. Som tidligere nevnt i 2.2.2 måtte derfor oppgaven endres etter en måneds arbeid. Gruppen fikk innspill fra veileder på kravspesifikasjon og utarbeidet en ny kravspesifikasjon ut fra innspillene. Det ville nok vært lettere om Innovasjon Norge hadde laget kravspesifikasjonen for oss, slik det skulle vært gjort ifølge dokumentasjonen, men når Innovasjon Norge hadde misforstått sitt ansvar valgte gruppen å utarbeide den for å få fremgang i prosjektet fremfor å bruke ytterligere tid på å vente. Gruppen brukte den neste uken på å utarbeide en ny oppgave, før veileder godkjente den i sprint 3. Gruppen merket at det var utfordrende å omstille seg til å skrive den nye oppgaven når vi først hadde skrevet og innarbeidet den første oppgavens kravspesifikasjonen.

Prosjektet innebar å lage en webapplikasjon som behandler og visualiserer data for å effektivisere utviklingsprosjektene til IT-avdelingen i Innovasjon Norge. Det var ønskelig å se nærmere på estimering og visualisering av prosjekter på ulike nivåer, og også kunne se hvor mange personer et prosjekt vil kreve. Det nye systemet skulle være en responsiv webapplikasjon som passer både mobile enheter og større skjermer.

3.1.1 Generelle krav

For å få en oversikt over hvor mye av kravspesifikasjonen som er oppnådd har vi sortert listen fra kravspesifikasjonen basert på hva vi oppnådde og hva som gjenstår. For å sammenligne kravspesifikasjonen med det endelige resultatet har vi valgt å gå gjennom først de generelle kravene, og deretter de funksjonelle og ikke-funksjonelle. Følgende liste er en oversikt over de generelle kravene som applikasjonen løser:

- Skal visualisere timer brukt opp mot timer estimert på work-item (task). ✓
- Vise hvor mange årsverk som er brukt på et prosjekt. ✓
- Skal kunne brukes til å estimere fremtidige prosjekt ✓
- Vise hvor mye tid som ble brukt forrige uke, måned, kvartal, år ✓
- Tid brukt på bugs, fra innrapportert til siste status ved fullført (Closed) ✓
- Vise estimeringsfeil forrige uke, måned, kvartal, år ✓
- Visualisere graf som viser faktisk tid brukt mot estimert tid ✓
- Mulighet for filtrering og fordeling mellom faktisk utviklingstid og estimert utviklingstid i analysegrafer eller segmenter ✓
- Analyse av Epics med antall Features og User Stories nødvendig for leveranse ✓

- Vise uferdig arbeid forrige uke, måned, kvartal, år ✗
- Vise hvor mange prosent man pleier å over- eller underestimere ✗
- Filtrering på person og tid brukt på tildelte oppgaver tasks og bugs ✗
- Mulighet for visning av gjennomsnittlig tid brukt for Features og User Stories ✗

Applikasjonen har mulighet til å vise hvor mye tid en utvikler estimerer at hen skal bruke på et work-item, og i dette tilfellet en taske eller oppgave. Dette vises frem i komponenten som heter “Sammenligning mellom estimerte og gjennomførte timer” og kan sees i seksjon 5.3.2. Denne komponenten tar inn data fra utviklere og finner avstanden mellom estimert tid og brukt tid. Den skiller også på om utvikler er innleid konsulent eller fast ansatt i Innovasjon Norge. Kravet for å “vise estimeringsfeil forrige uke, måned, kvartal, år” inneholder datofilter, slik at det er mulig å velge for et gitt tidsrom.

Kapittel 5.3.3 viser komponenten Timeliste for konsulenter som kan hente ut antall timer jobbet på et prosjekt ned til enkeltperson. Dette var ønsket fra avdelingsleder, og denne ble prioritert når gruppen fikk beskjed om dette 5. mai. På den måten vil det være mulig å “vise hvor mange årsverk som er brukt på et prosjekt”. Videre vil det være mulig å se på en oppdeling av junior - og seniorutvikler,

men dette var noe vi ikke kom i mål med. Dette vil kreve enten en database med hjelpetabeller som inneholder ansattopplysninger eller metadata i tasks som utfyller informasjon om rollen til utvikleren. Denne komponenten oppfyller også krav om "Vise hvor mye tid som ble brukt forrige uke, måned, kvartal, år" siden den har filtreringsmuligheter for dato. Dermed kan den stilles inn på den perioden man ønsker. Komponenten kan dermed brukes til å "estimere fremtidige prosjekt".

På dashbordet (se 5.3.1 ligger flere komponenter som kan "brukes til å estimere fremtidige prosjekt". "Analyse av Epics med antall Features og User Stories nødvendig for leveranse" er mulig gjennom komponenten "Epics" som gir en oversikt over antall user stories og features som finnes på en epic. Dette kan velges gjennom en nedtrekksliste, og det finnes også en fordelingskomponent som gir en prosentvis oversikt over features og user stories i en epic. Videre gir komponenten "Bug per prosjekt" svar på kravet "Tid brukt på bugs, fra innrapportert til siste status ved fullført (Closed)", hvor man kan se en oversikt over hvert enkelt prosjekt og hvor lang tid man over en gitt tidsperiode har brukt på å lukke bugs som har oppstått. Den gir også en oversikt over hvor mange åpne bugs som gjenstår. Det gruppen ikke kom i mål med var generelle krav som "Vise uferdig arbeid forrige uke, måned, kvartal, år", noe som kunne vært løst hvis gruppen hadde hatt mer tid. Dataen er tilgjengelig, og Innovasjon Norge vil i fremtiden kunne hente inn denne informasjonen gjennom en komponent som filtrerer ut, for eksempel sprintene og henter work items (task) som er lagt til i sprinten, men som ikke er closed. Når det gjelder kravet "Vise hvor mange prosent man pleier å over- eller underestimere" så kunne det vært løst ved å bruke fordelingskomponenten som allerede eksisterer. Skrive den litt om og hente ut tallene som allerede eksisterer fra komponenten som henter inn estimerte og gjennomførte timer. Dette ville vært et enkelt regnestykke. Kravet om "Filtrering på person og tid brukt på tildelte oppgaver tasks og bugs" er også løsbart og kan gjøres ved å sette opp en nedtrekksliste med personer i stedet for prosjekter, og hente inn alle work items for en person i en gitt tidsperiode. "Mulighet for visning av gjennomsnittlig tid brukt for Features og User Stories" løses ved å hente inn data i en komponent fra tasken får status som aktiv og frem til den lukkes. Dette kunne også vært løst hvis gruppen hadde hatt mer tid, men samtidig gir dette også Innovasjon Norge mulighet til å teste ut applikasjonen og se om det er annen funksjonalitet enn den som er der nå som er ønskelig. Applikasjonen med sine komponenter gjør det enkelt å legge til ytterligere funksjonalitet, så dette gir et godt utgangspunkt for videre utvikling.

3.1.2 Funksjonelle krav

Funksjonelle krav er beskrivelser av hvilke tjenester systemet skal tilby, hvordan systemet skal respondere på ulike inputs, og hvordan systemet skal oppføre seg i ulike situasjoner (Sommerville, 2016, s. 105). De funksjonelle kravene skal i sum beskrive hva systemet skal gjøre.

Innovasjon Norge beskrev dette som systemkrav som skulle gi detaljerte beskrivelser av systemets funksjoner, tjenester og operasjonelle begrensninger. Altså definerer vi her hva som blir utviklet. I samarbeid med Innovasjon Norge kom vi frem til følgende funksjonelle krav:

- Som leder vil jeg estimere hvor lang tid mine ansatte vil bruke på et prosjekt i timer.
- Som leder vil jeg estimere hvor mange ansatte jeg vil trenge på et prosjekt.
- Som bruker vil jeg se estimert, gjenværende og brukt tid, samt feilmargin på estimering av mine oppgaver.
- Som leder vil jeg se feilmarginer for estimert tid i mitt team eller min avdeling.
- Som leder ønsker jeg å se "work load" for de ulike fasene; planlegging, utvikling og vedlikehold.
- Som bruker vil ønsker jeg å kunne eksportere data basert på filtreringer

De fem første kravene var prioritert som høy i kravspesifikasjonen, mens det siste kravet hadde prioritet medium. Vi kom i mål med de fire første kravene i løpet av prosjektet, noe vi er godt fornøyd med. I tillegg viste det seg vanskelig å få gjennomført kravet "Som leder ønsker jeg å se "work load" for de ulike fasene; planlegging, utvikling og vedlikehold." fordi Innovasjon Norge ikke brukte tags i work items konsistent. Dermed visste gruppen ikke om man kunne stole på metadata man sorterte på, og om det ville gi verdi. Gruppen satte derfor dette kravet til side og valgte heller å fokusere på de andre kravene.

3.1.3 Ikke-funksjonelle krav

Brukervennlighet

- Skal være enkelt å forstå, slik at opplæring kan holdes til et minimum ✓
- Systemet støtter generelt tilgang / operasjoner fra mobile enheter, som nettbrett ✓

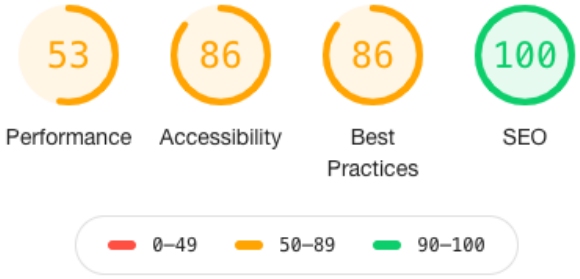
- Støtter smarttelefon ✓
- Koden skal være enkel å forstå slik at den kan bygges på i etterkant ✓

Som vist i produktinformasjon støtter applikasjonen nettbrett og smarttelefon og er responsiv. Applikasjonen er enkel å forstå, og den er laget på en måte som gjøre det enkelt å bygge videre på. Gruppen hadde høyt fokus på denne delen, og vi føler oss godt fornøyd med endelig resultat.

Universell utforming

- Skriften skal være leselig og ikke for liten ✓
- Grafene skal ha nok kontrast til at en som er fargeblind kan tydelig skille mellom dem ✓
- Det skal være god kontrast mellom tekst og bakgrunn ✓
- Knappene skal være tydelige og det skal ikke kreves presisjon for å klikke på knapper ✓

Gruppen har kjørt Google Lighthouse som er et verktøy for test av universell utforming, hvor den scorer webapplikasjonen til 86/100. Det bør selvsagt tilstrebes å score 100/100, og dette er nok noe Innovasjon Norge vil måtte se på i videre arbeid og testing før den settes i produksjon, men testen peker også mot at applikasjonen er på rett vei.



Figur 3.1: Utsnitt fra måling gjort med Google Lighthouse

Tilgjengelighet

- Applikasjonen skal være responsiv og skal kunne brukes på store og mindre skjermer ✓

Som vist i kravet "Brukervennlighet" er applikasjonen responsiv og kan benyttes på både store og mindre skjermer.

Estetiske krav

- Applikasjonen skal bruke Innovasjon Norge sin grafiske profil. ✓
- Det skal brukes komponenter fra Innovasjon Norge sitt React komponentbibliotek. ✗
- Grafene skal bestå av både visualiseringer og tekst så grafene er lette å forstå. ✓

Gruppen har benyttet Innovasjon Norge sin profil, og tilstrebet at farger skulle være hentet fra Innovasjon Norge så langt som det lot seg gjøre. Videre er også rapporten i seg selv tilpasset Innovasjon Norge gjennom valg av farger til overskrifter i rapporten. Det var ønskelig å bruke komponentbiblioteket til Innovasjon Norge, men siden vi aldri fikk tilgang til det så måtte vi gå bort fra det kravet.

Sikkerhet

- Bruker må være innlogget for å kunne se data. ✓

Dette er implementert, og bruker kan kun logge inn ved å autentisere seg gjennom Azure Active Directory med sin bruker fra Innovasjon Norge.

Ytelse

- Poengsummen til webapplikasjonen skal være mellom 50 og 100 på Lighthouse Scoring Guide.✓

Dette har gruppen lykkes med, som kan ses av figur 3.1.

Pålitelighet

- Applikasjonen skal ha en oppetid på 99% såfremt API-tilkobling virker.✓

Applikasjonen skal har ikke andre avhengigheter enn strøm og API-tilkoblingen til 7Pace TimeTracker så den skal kunne opprettholde oppetidskravet.

Robusthet

- Applikasjonen skal ikke bryte sammen som følge av feil i koden.✓

Det er så langt ikke funnet feil i koden som fører til at den bryter sammen. Koden er kjørt og testet visuelt gjennom hele prosessen. Videre testing før implementasjon vil også kunne verifisere dette kravet bedre.

Prosesskrav

- Systemet skal utvikles med smidige metoder. ✓
- Backend utvikles i ASP.Net Core 3✓
- Frontend utvikles i React med TypeScript.✓
- Visual Studio benyttes for programmering i ASP.Net Core 3.0✓
- Visual Studio Code benyttes for programmering i React✓

Gruppen bruker Scrum-metodikk for utvikling av webapplikasjonen. Gruppen har også holdt på alle andre krav knyttet til prosess, og samtlige teknologier og språk som var tenkt brukt har blitt brukt.

3.2 Design og implementering

Gruppen har siden oppstart hatt kravspesifikasjonen i bakhodet. Ut fra kravspesifikasjonen valgte gruppen teknologi som ville kunne oppfylle de fleste av kravene som var stilt. Siden applikasjonen Innovasjon Norge ønsket skulle kunne behandle forskjellige data dynamisk var vi avhengig av at applikasjonen var rask, noe en Single Page Application kunne hjelpe oss med. Dette gjorde at man ikke måtte navigere til nye sider, men bare endret views. For å hente inn så mye forskjellig data dynamisk var det fornuftig å finne en måte å gjøre dette på som gjorde at alt ikke måtte hentes samtidig når siden ble lastet. Ved å bruke komponenter fra React / TypeScript så fikk vi mulighet til å hente data når det var nødvendig ved hjelp av async-komponenter som først kjørte når de ble kalt på. I tillegg kunne vi benytte routing som gjør det mulig for Innovasjon Norge å aksessere nettsidene direkte ved å bruke URL. En annen fordel ved å bruke React var at dette var noe som Innovasjon Norge hadde i sin tech stack, og dermed ville det bidra til at videreutvikling ville være enklere.

Som nevnt i sprint 6 la gruppen ned betydelig innsats for å utarbeide farger som var godkjente for bruk, og som ville oppfylle de ikke-funksjonelle kravene. Det at applikasjonen skal være universelt utformet er noe gruppen har lagt vekt på, og dette har også vært et viktig tema fra faget Webutvikling fra studiets første år. Gruppen gjennomførte den gang et prosjekt innen webutvikling hvor det var stilt krav til 100% score for universell utforming, og en nettside som skulle kunne benyttes kun med HTML og CSS. Gruppen har ikke laget denne applikasjonen for kun HTML og CSS på grunn av kravene for dynamiske funksjoner, men har likevel båret med seg universell utforming gjennom utviklingsprosessen.

Kapittel 4

Produktdokumentasjon

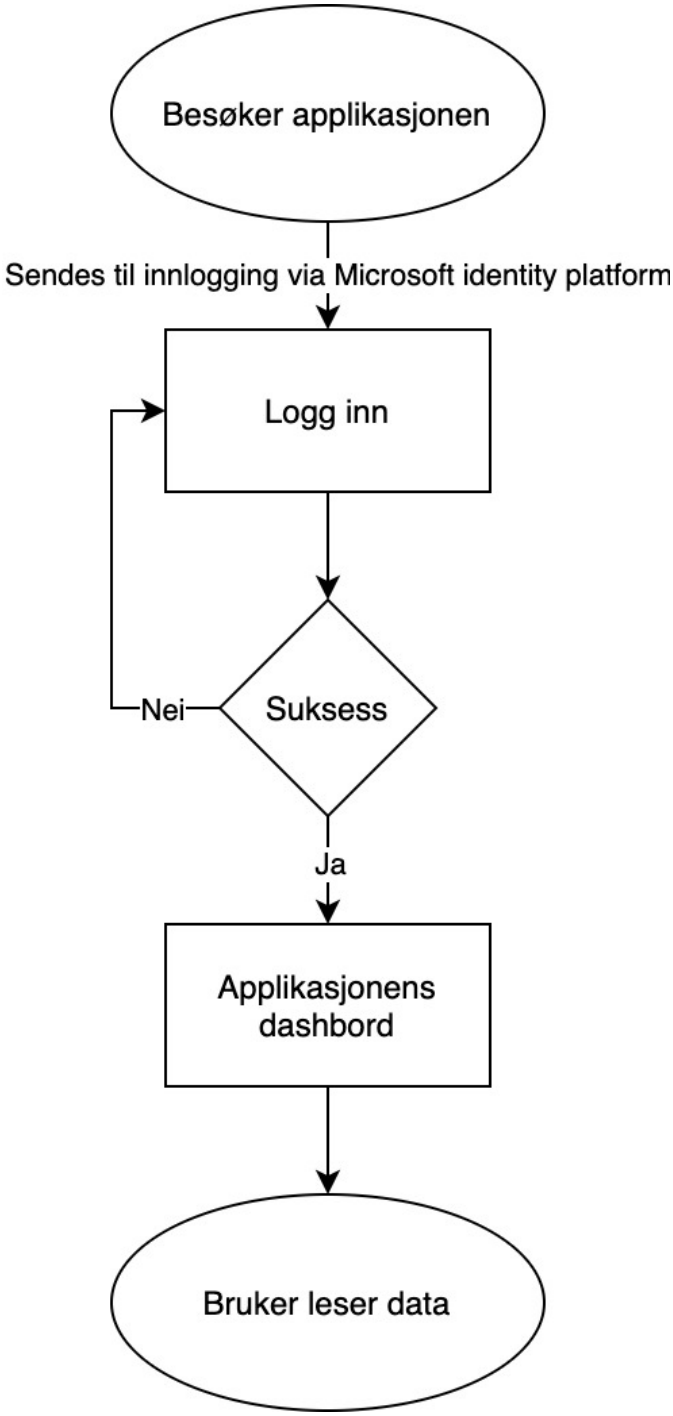
4.1 Innledning

Produktdokumentasjonen beskriver det ferdige produktet samt den tekniske dokumentasjonen knyttet til produktet. Den tekniske kompetansen som trengs for å lese dokumentasjonen er kunnskaper om objektorientert programmering, webutvikling, og tekniske begreper innen systemutvikling. Begrep og forkortelser er forklart i vedlegg A. Kapittelet er delt opp i følgende deler;

- DevOps
- Server
- Klient
- Klient

4.1.1 Systemdesign

Systemet virker ved at en prosjektleder / avdelingsleder logger seg inn og får oversikt over relevante data for et pågående eller tidligere prosjekt. Eksempler på data kan være hvor mange personer som er knyttet til prosjektet, eller hvor mange estimerte timer det er sammenlignet med faktisk brukte timer. Dataen blir hentet direkte fra systemet utviklerne bruker for å registrere timer på de ulike arbeidsoppgavene i prosjektene. Figur 4.1 viser enkel flyt fra bruker logger inn til data vises.



Figur 4.1: Enkelt flytdiagram over applikasjonens flyt

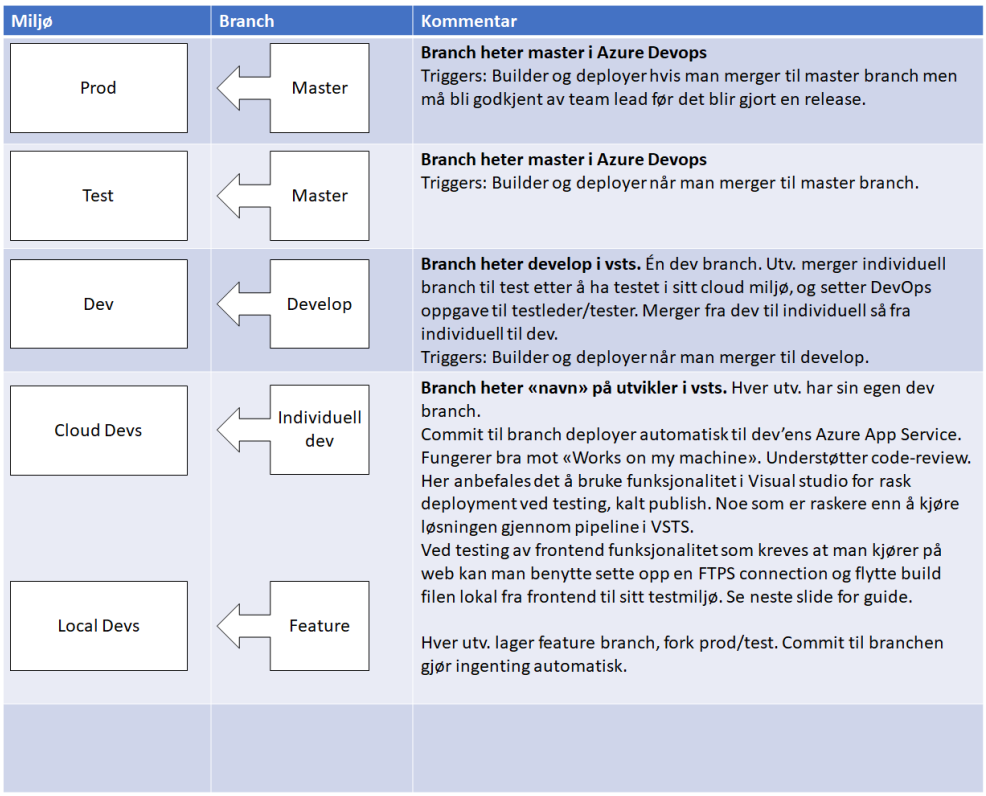
4.1.2 Om applikasjonen

Applikasjonen vi har utviklet skal være et verktøy for estimering av fremtidige prosjekter samt visualisering av status for pågående prosjekt. I første omgang er tanken at dette skal være et verktøy for prosjektledere. Ved videreutvikling er det ønskelig at teammedlemmer også kan benytte seg av verktøyet.

4.2 Azure DevOps

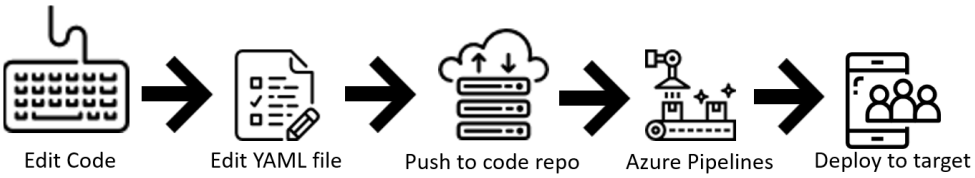
4.2.1 Pipeline

For å kontinuerlig bygge og teste kode tar vi i bruk Azure Pipelines. Dette er en delivery pipeline som består av flere steg for å levere koden til Azure Repos.



Figur 4.2: Branches konvensjon i IN

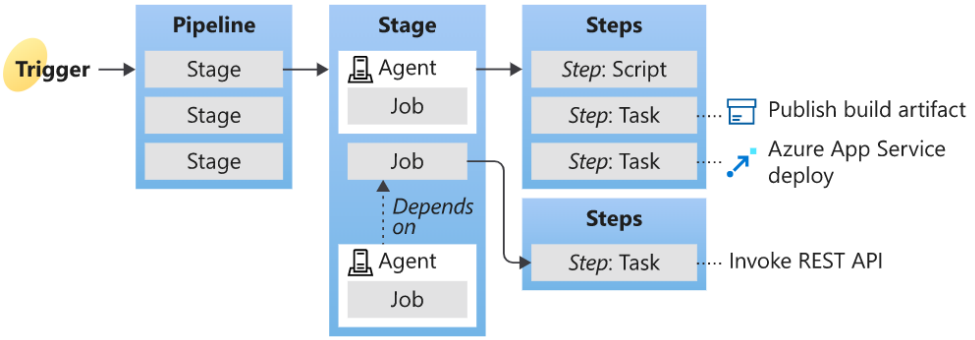
Vi har definert mesteparten av vår pipeline med "infrastructure as code", som vil si at vi automatiserer oppsettet med kode. Vi har fått det meste av denne koden av Innovasjon Norge. Figur 4.3 viser de ulike stegene for å deploye en applikasjon med "infrastructure as code".



Figur 4.3: Flyt i "infrastructure as code" (KathrynEE mfl., 2019).

Vi har infrastrukurkoden vår i en fil kalt azure-pipelines.yml. Her definerer vi hvordan byggingen av applikasjonen skal foregå. Språket vi bruker her er YAML som er et språk som ofte brukes i konfigurasjonsfiler.

I figur 4.4 har Microsoft tatt for seg nøkkelkonsepter i Azure Pipelines.



Figur 4.4: Nøkkelkonsepter i pipeline (Kulla-Mader mfl., 2020)

Vi ser at en pipeline aktiveres av en trigger. Pipelinen består av flere stages, heretter omtalt som etapper, der en eller flere jobber kjører. Hver av disse jobbene består av flere steps, heretter omtalt som steg.

4.2.2 Triggere

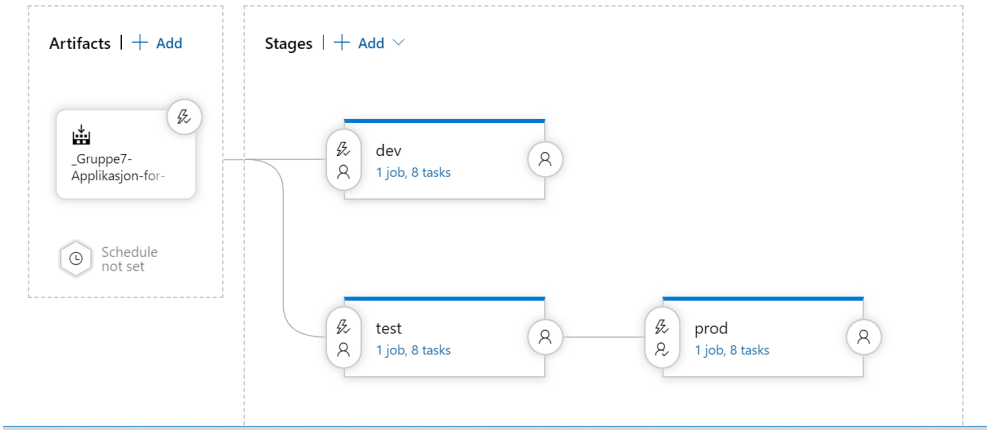
Triggerene sier når pipelinen skal kjøre. I 4.5 definer vi at pipelinen skal kjøre når vi merger med develop-branch eller master-branch. For at mergen skal godkjennes må pipelinen kjøre uten feil. Dette fører til at man minsker sjansene for å merge feil kode inn i develop -og master-branch.

```
5   trigger:
6     branches:
7       include:
8       - develop
9       - master
```

Figur 4.5: Definerer triggere i azure-pipelines.yaml

4.2.3 Etapper

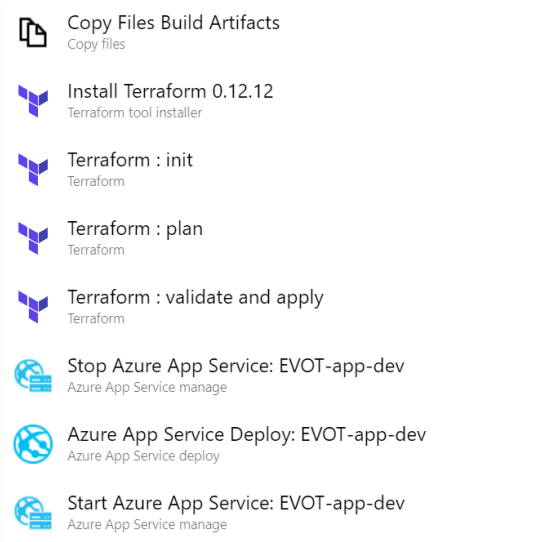
Etappene er definert manuelt og ikke med kode. Igjen så gjør vi dette på måten Innovasjon Norge gjør det. En etappe består av en eller flere jobber. Som vist i figur 4.6 har vi i vår pipeline 3 etapper: dev, test og prod. I hver etappe har vi én jobb med 8 oppgaver.



Figur 4.6: Etapper i pipeline

4.2.4 Oppgaver

Figur 4.7 er hentet fra Azure DevOps og viser hvilke oppgaver vi har lagt inn på jobben i etappen. Denne er lik på alle etappene.



Figur 4.7: Oppgaver(tasks) i pipeline

Disse gjør følgende:

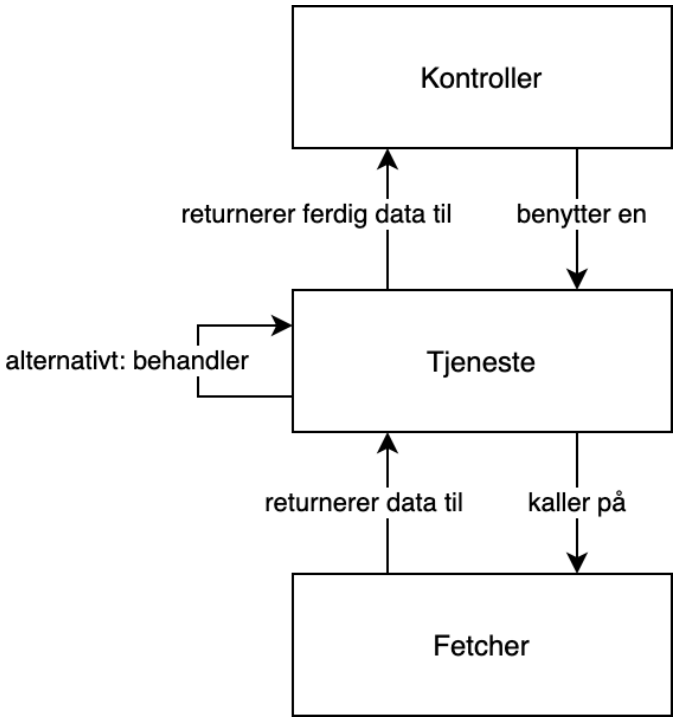
1. Kopierer filene i prosjektet til etappen og bygger artifaktene. Artifakter er filene vi ønsker at builden får skal ha.
2. Installerer Terraform
3. Initialiserer Terraform. Utfører flere steg for å ta i bruk mappen med filene vi skal bruke («Command: init», 2020).
4. Lager en utførelsesplan for Terraform («Command: plan», 2020)
5. Validerer konfigurasjonsfilene i mappen («Command: plan», 2020).
6. Anvender endringene som trengs for å oppnå ønsket tilstand («Command: apply», 2020).
7. Stopper, deployer og starter applikasjonen i Azure App Service.

4.3 Server

Serverens oppgave er å hente, behandle og levere data til klienten, på en sikker måte. For å gjøre dette kreves ulike mekanismer som vi skal ta for oss her. Serveren danner APIet som webapplikasjonen vår “snakker” med.

4.3.1 Livsløp

I dette avsnittet skal vi ta for oss livsløpet fra et endepunkt blir kallet til endepunktet returnerer data.



Figur 4.8: Livsløp fra endepunkt i kontroller blir kalla til den returnerer data

Vi ønsker at endepunktene i kontrollerne skal være ryddige, så derfor skjer databehandling i tjenester. Som kodeliste 4.1 viser, er alt dette endepunktet gjør, å ta inn Id-en til epicen vi skal telle antall features til.

```
[Route("api/[controller]")]
[ApiController]
public class FeaturesController : ControllerBase
{
    readonly IFeatureService _service;

    public FeaturesController(IFeatureService service)
    {
        _service = service;
    }

    // GET api/<controller>/5
    [Authorize]
    [HttpGet("{EpicId}")]
    public async Task<IActionResult> GetCount(int epicId)
    {
        var response = await _service.GetCount(epicId);

        return Ok(response);
    }
}
```

Kodeliste 4.1: FeatureController.

Neste steg er tjenesten `FeatureService`, som vi ser i kodeliste 4.2. Her opprettes det et nytt objekt av typen `Fetcher` som brukes for å hente responsen fra 7Pace TimeTracker-APIet. Vi definerer en spørring som gir oss den dataen vi ønsker. I dette tilfellet ønsker vi barn av en spesifikk epic og sender derfor med en id. Spørring, id, endepunktet til 7Pace TimeTracker, og hvilke type objekt vi får returnert, mates inn i metoden `Get` som `Fetcher` gir. Metoden `GetResponse<>()` returnerer en liste med objekt av typen som vi forventet. Denne listen kan behandles videre eller returneres til kontrolleren, slik som i 4.2.

```
public class FeatureService : IFeatureService
{
    public FeatureService()
    {
    }

    // henter children til epic som er feature
    public async Task<IList<WorkItem>> Get(int epicId)
    {
        Fetcher fetcher = new Fetcher();

        string query = "Children?$filter=System_WorkItemType eq 'Feature'";

        await fetcher.Get<WorkItem>(EndPointEnum.WorkItemsHierarchy, query,
        epicId);

        var response = fetcher.GetResponse<WorkItem>();

        return response;
    }
}
```

Kodeliste 4.2: FeatureController.

4.3.2 Nøkkelkomponenter

Datahenter

Dataen vi skal behandle kommer fra et eksternt API. Oppgaven til datahenteren er å hente dataen vi ønsker fra 7Pace TimeTracker APIet. I kildekoden kaller vi klassen `Fetcher`. Fetcher-en har to nøkkelmetoder. Første metode er `GetAsync<T>`. Metoden er generisk, som vil si at den tar inn en typeparameter `T`, som kan settes til en valgfri datatype når en bruker metoden. I tillegg tar metoden inn to obligatoriske parametre og en valgfri. De obligatoriske er endepunkt til 7Pace TimeTracker API-et og spørringen. 7Pace-API-et har også mulighet for å hente barn ut ifra id, derfor er det også en valgfri parameter for id i metoden.

Den andre metoden er `GetResponse`-metoden. Denne er også generisk, og må ha samme typeparameter som `Get`-metoden. Denne metoden returnerer resultatet som `IList` av typen som hentes. På denne måten får vi en liste som bare inneholder de objektene vi er ute etter, ikke annen data fra API-et som ikke er relevant å hente ut.

For å få tilgang til API-et må vi ha en token generert av 7Pace TimeTracker. Denne nøkkelen har en levetid på ett år før den må fornyes. Nøkkelen sendes i forespørselen fra serverapplikasjonen til 7Pace TimeTracker, som vist i kodeliste 4.3. På denne måten autentiseres applikasjonen, og den får tilgang til data fra API-et.

```
private async Task<ApiResponse<T>> GetAsync<T> (EndPointEnum endPointEnum,
string query, int? id)
{
    using var client = new HttpClient();
    client.DefaultRequestHeaders.Authorization =
    new AuthenticationHeaderValue("Bearer", accesstoken);

    try
    {
        var baseUriStringWithType = (id != null) ? baseUriString +
        endPointEnum.Get + string.Format("({0})/", id) : baseUriString +
        endPointEnum.Get;

        var baseUri = new Uri(baseUriStringWithType);

        client.BaseAddress = baseUri;

        var response = await client.GetAsync(query);
        response.EnsureSuccessStatusCode();

        var data = await response.Content.ReadAsStringAsync();

        return JsonConvert.DeserializeObject<ApiResponse<T>>(data);

    }
    catch (HttpRequestException httpRequestException)
    {
        var errormessage = new ApiResponse<T>
        {
            ODataContext = httpRequestException.Message
        };
        return errormessage;
    }
}
```

Kodeliste 4.3: Utdrag av koden til Fetcher

Tjenester

I tjenestene blir dataen som ble hentet fra API-et behandlet, for å gjøres tilgjengelig for kontrollere. Dette kan være sortering, kalkulering eller annen omforming av dataen. I de fleste tilfeller vil det være en service som bruker Fetcher-klassen som ble nevnt i forrige avsnitt. I kodeliste 4.4, ser vi et utdrag der dataen grupperes ut fra dato og legges i objekt som inneholder bare de datafeltene som er ønsket.

```
private static List<TimeByDate> GroupByDate(IList<TimePlus> data)
{
    var groupedList = new List<TimeByDate>();
    var groupBy = data.GroupBy(x => x.System_CreatedDate);

    foreach (var item in groupBy)
    {
        var timeGrouped = new TimeByDate
        {
            Date = item.Key
        };

        foreach (var time in item)
        {
            if (time.Microsoft_VSTS_Scheduling_CompletedWork != null)
                timeGrouped.CompletedWork +=
                (double) time.Microsoft_VSTS_Scheduling_CompletedWork;
            if (time.Microsoft_VSTS_Scheduling_OriginalEstimate != null)
                timeGrouped.OriginalEstimate +=
                (double) time.Microsoft_VSTS_Scheduling_OriginalEstimate;
        }
        groupedList.Add(timeGrouped);
    }
    return groupedList;
}
```

Kodeliste 4.4: Eksempel på behandling av data fra API.

Kontrollere

Kontrollerne er den delen som klienten kommuniserer med for å hente data. En kontroller inneholder en rekke endepunkt som klienten kaller på ut fra hva som skal hentes. Det er de ulike endepunktene i kontrollerne som trigger tjenestene som henter og behandler dataen som skal returneres til klienten.

Sikkerhet

Innovasjon Norge er en bedrift som i stor grad baserer seg på Microsoft Office 365. Derfor var det naturlig å bruke innlogging via Azure Active Directory (Azure AD). Ved å benytte seg av sentralisert identitetshåndtering slipper applikasjonen å forholde seg til administrering av brukeridentiteter og tilgangsstyring. Om en ansatt ved Innovasjon Norge er innlogget et annet sted med jobbkontoen, vil personen med et tastetrykk være innlogget i applikasjonen om den åpnes. Protokollen som brukes for dette er OpenID Connect. Denne protokollen er sponset av selskaper som Facebook, Microsoft og Google.

OpenID Connect definerer tre aktører

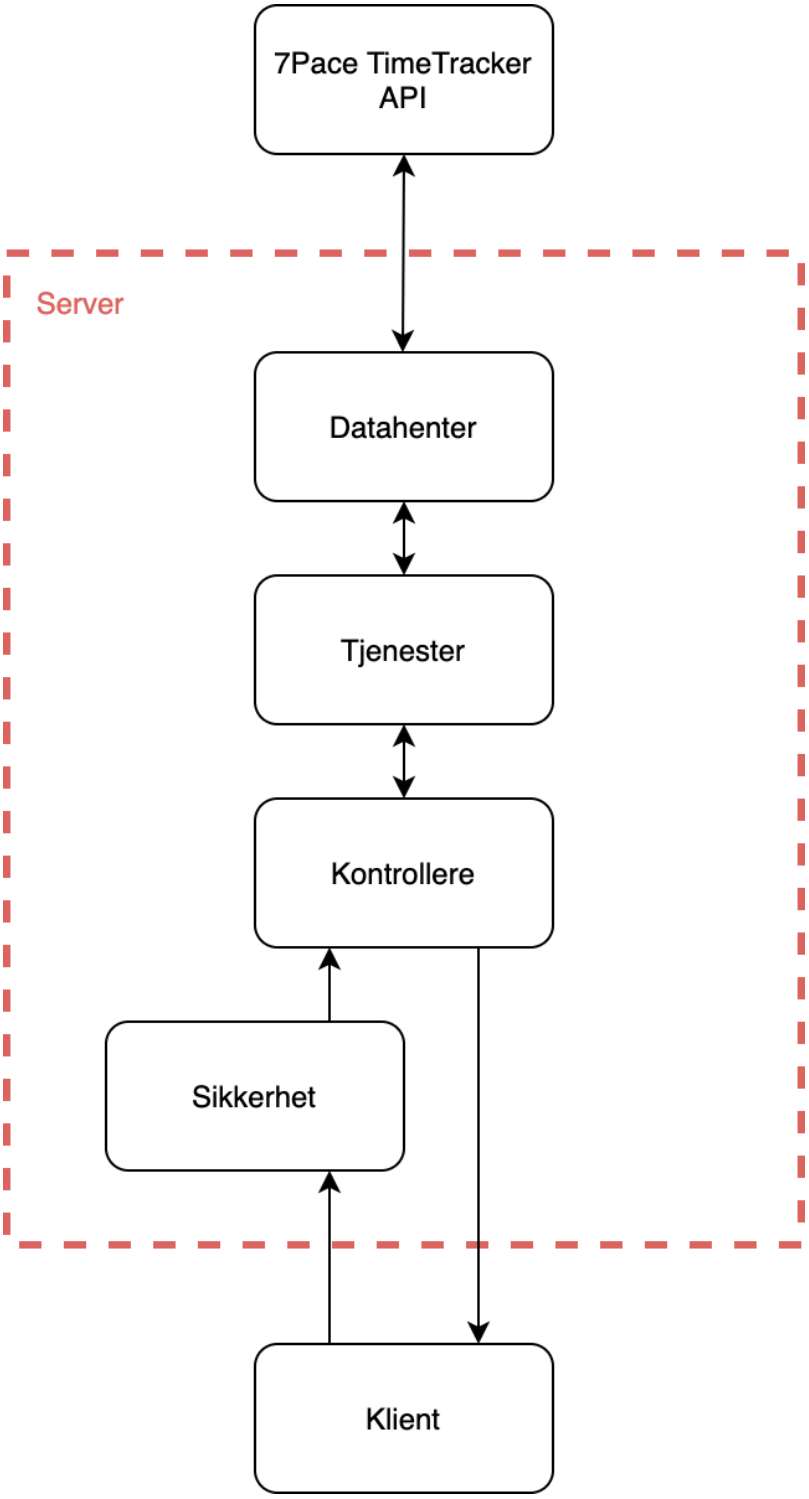
- Sluttbruker
- Ressurs
- OpenID utsteder

Sluttbruker er brukeren som ønsker å aksessere en ressurs. Ressursen er i dette tilfellet applikasjonen vår som det bare skal gis tilgang til, om sluttbrukeren godkjennes. Azure AD er utsteder av OpenID og autentiserer brukeren før den eventuelt får tilgang til ressursen. Kunnskap rundt dette fikk vi i faget ITPE3100 Datasikkerhet. På denne måten får vi en sikker applikasjon som bare kan aksesseres om brukeren har tilknytning til Innovasjon Norge.

Når brukeren først besøker applikasjonen, sjekker serveren om brukeren er innlogget i nettleseren. For å sjekke om brukeren er innlogget sender klienten en forespørsel til endepunktet `auth/isAuthenticated`. Om brukeren ikke er innlogget, gir forespørselen feilkoden "401 - Uautorisert feil". Da utfører klienten en ny forespørsel mot `auth/login` som sender brukeren til innlogging. Dette endepunktet tillater, ved å bruke `[AllowAnonymous]` på metoden, forespørsler uten at man er innlogget. Her kommer det tydelig frem at man må logge inn med Innovasjon Norge-domenet. Når brukeren er innlogget lagres et token i nettleseren som forteller serveren at brukeren er innlogget og applikasjonen blir tilgjengelig.

```
[HttpGet]
[AllowAnonymous]
[Route("login")]
public IActionResult Login(CancellationToken cancellationToken = default)
{
    if (User.Identity.IsAuthenticated) //=Er bruker autentisert av Azure AD
    {
        try
        {
            // Sjekk om bruker har tilgang til applikasjonen, kaster
            // UnauthorizedAccessException om nei
        }
        catch (UnauthorizedAccessException)
        {
            return Unauthorized(new { Message = "Du er ikke autorisert"+
            "for denne applikasjonen." });
        }
    }
    else
    {
        // Trigger Azure AD autentisering
        return Challenge(AzureADDefaults.OpenIdScheme);
    }
    // Sender til hjem om autentisert
    return Redirect("/");
}
```

Kodeliste 4.5: Endepunkt for innlogging.

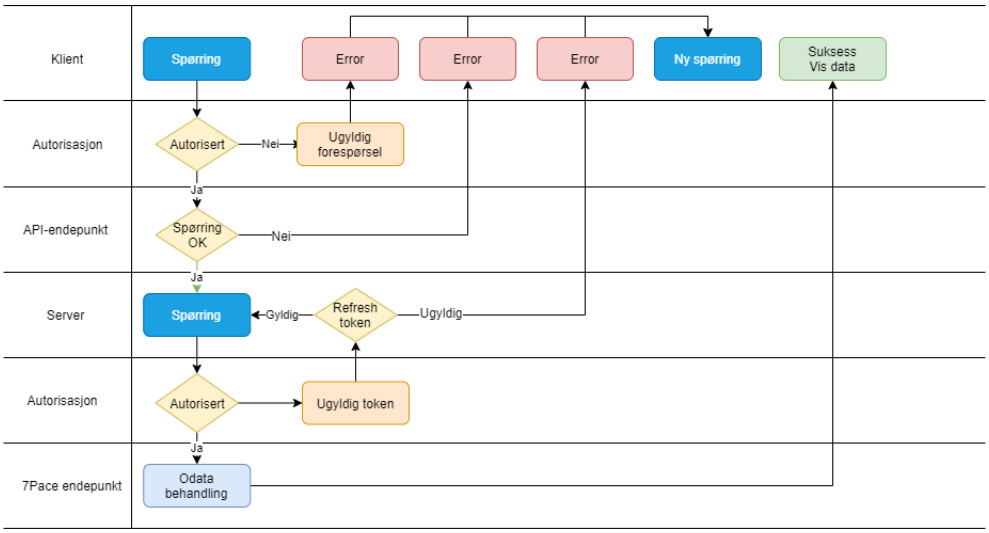


Figur 4.9: Flytdiagram over hvordan de ulike delene i serveren snakker sammen.

4.3.3 API

Flyt i API

Dataen som vises i applikasjonen, henter vi fra 7Pace sitt reporting API (se 2.3.2). Det er klienten som igangsetter spørringen ved å kalle på et endepunkt i APIet vårt. Endepunktene er koblet til en kontroller på ASP.NET Core applikasjonen vår på serveren vår. For at spørringen skal være gyldig, må bruker være innlogget. På server utfører vi spørringene mot 7Pace-APIet, der vi også må ha gyldig token som vi får fra 7Pace. På server behandles dataen og returnerer ønsket data til klient i JSON format.



Figur 4.10: Flyt i API

Figur 4.10 viser flyten til APIet:

- Klienten sender en spørring til vårt API.
- Hvis bruker er autorisert og spørringen er ok, sender serveren vår videre en spørring til 7Pace sitt API.
- API-endepunktet i 7Pace gir data som videre behandles på server.
- Server returnerer data.

Endepunkt

På de neste sidene følger en oversikt over API'ets endepunkter utviklet til Applikasjonen. Alle endepunkter har `api/` i starten av url-en, dette er for å kunne skille mellom API og annen navigering i webapplikasjonen.

auth

GET `/auth/login` Sender bruker til innlogging

GET `/auth/user` Henter brukerinformasjon

Resultat

Eksempel på resultat

```
{
  ”name”: ”Ola Hansen”,
  ”mail”: ”ola.hansen@mail.no”
}
```

GET `/auth/isAuthenticated` Returnerer OK om innlogget

Resultat

200: OK

401: Unauthorized

businessepic

GET `/businessepic/all` Gir alle business epics

Resultat

Eksempel på resultat

```
[
  {
    ”label”: ”Business Epic 1”,
    ”value”: ”152”
  }, ..
]
```

GET `/businessepic/all/startdate` Gir alle business epics med startdato

Resultat

Eksempel på resultat

```
[
  {
    ”name”: ”Business Epic 1”,
    ”startdate”: ”152”,
  }, ..
]
```

epics

GET `/epics` Gir alle epics

Resultat

Eksempel på resultat

```
[
  {
    "label": "Epic 1",
    "value": "152",
  }, ..
]
```

features

GET `/features/{epic_id}` Gir antall features tilknyttet epic

Resultat

Eksempel på resultat

```
{
  "count": 1
}
```

userstories

GET `/userstories/{epic_id}` Gir antall user stories tilknyttet epic

Resultat

Eksempel på resultat

```
{
  "count": 1
}
```

bug

GET `/bug/allbugs` Gir data om alle lukka bugs

Parametere

Navn	Beskrivelse
project *påkrevd (streng)	Navn på prosjekt.
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/bug/allbugs?project=TeamProsjekt&start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    ”project”: ”TeamProsjekt”,
    ”openbugs”: 4,
    ”closedbugs”: 2,
    ”timetoclose”: 2
  }, ..
]
```

time

GET `/time/person` Liste over personer med antall timer og estimering

Parametere

Navn	Beskrivelse
project *påkrevd (streng)	Navn på prosjekt.
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/time/person?project=TeamProsjekt&start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    ”person”: ”Ola Hansen”,
    ”OriginalEstimate”: 43,
    ”CompletedWork”: 52
  }, ..
]
```

GET `/time/date` Liste over datoer med antall timer og estimering

Parametere

Navn	Beskrivelse
project *påkrevd (streng)	Navn på prosjekt.
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/time/date?project=TeamProsjekt&start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    ”date”: ”2020-04-21”,
    ”OriginalEstimate”: 84,
    ”CompletedWork”: 94
  }, ..
]
```

GET `/time/consultant` Timeliste for konsument-utviklere på prosjekt

Parametere

Navn	Beskrivelse
project *påkrevd (streng)	Navn på prosjekt.
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/time/consultant?project=TeamProsjekt&start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    "user": "total",
    "totalhours": 214
  },
  {
    "user": "Ola Hansen",
    "totalhours": 42
  },..
]
```

GET `/time/inhouse` Timeliste for inhouse-utviklere på prosjekt

Parametere

Navn	Beskrivelse
project *påkrevd (streng)	Navn på prosjekt.
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/time/inhouse?project=TeamProsjekt&start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    "user": "total",
    "totalhours": 214
  },
  {
    "user": "Ola Hansen",
    "totalhours": 42
  },..
]
```

GET `/time/consultant/allteams` Timeliste for alle konsument-utviklere

Parametere

Navn	Beskrivelse
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/time/consultant/allteams?start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    "user": "total",
    "totalhours": 214
  },
  {
    "user": "Ola Hansen",
    "totalhours": 42
  },..
]
```

GET `/time/inhouse/allteams` Timeliste alle inhouse-utviklere

Parametere

Navn	Beskrivelse
start *påkrevd (ISO 8601)	Starttidspunkt på data som skal hentes.
end *påkrevd (ISO 8601)	Sluttidspunkt på data som skal hentes.

Eksempel

```
/time/inhouse/allteams?start=2020-01-01T22:00:00.000Z&end=2020-01-31T22:00:00.000Z
```

Resultat

Eksempel på resultat

```
[
  {
    "user": "total",
    "totalhours": 214
  },
  {
    "user": "Ola Hansen",
    "totalhours": 42
  },..
]
```

4.3.4 Lagdeling

Implementering av MVC

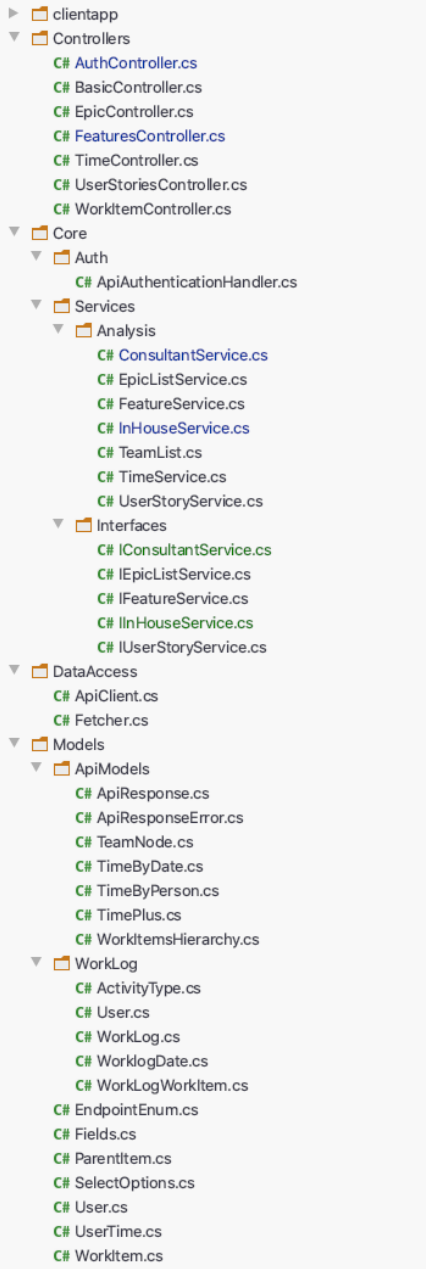
Applikasjonen baserer seg på designmønsteret, MVC (se 2.2.7). MVC står for model view controller og beskriver et designmønster der applikasjonen deles opp i de tre delene.

Siden vi lager et API, viker vi litt fra dette prinsippet, da vi ikke har views i form av HTML-sider, men returnerer heller serialisert data i JSON i kontrollerne.

I oppgaven hentes en rekke data fra 7Pace sitt API, (se 2.3.2). Disse dataene tilpasses modeller og behandles i tjenester og kontrollere før den serialiseres og deles med klient.

4.3.5 Mappestruktur

Klientapplikasjonen vår i React ligger under mappen, “clientapp”. I tillegg har vi egne mapper til “Models” og “Controllers” der vi legger modellene og kontrollerene, noe vi har hentet fra MVC. I tillegg har vi “DataAccess” og “Core”. “DataAccess” inneholder klasser som henter data, som i vårt tilfelle er klassen “Fetcher”. I “Core” finner man klasser som trengs for å gjøre ulike oppgaver. Disse kan ses på som ulike verktøy i en verktøykasse, hvor du finner alle tjenester som behandler dataen, og interfacene de implementerer. Denne mappestrukturen gjør det oversiktlig å finne de ulike klassene og fungerer godt til applikasjonenes formål.



Figur 4.11: Mappestruktur

4.3.6 Håndtering av avhengigheter i applikasjonen

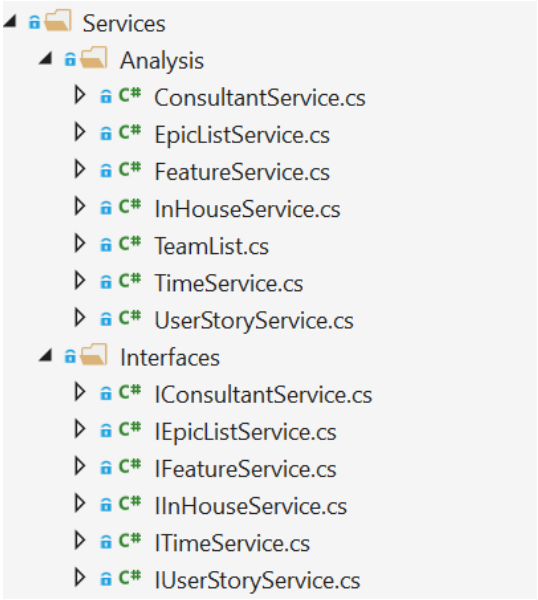
For å gjøre applikasjonen testbar, modulær og mulig å vedlikeholde benytter vi oss delvis av designprinsippet dependency inversion (se 2.2.7). Måten vi gjør dette på, er at vi lager interfaces til tjenestene vi bruker i kontrollere. Kontrollerene er avhengige av tjenestene for å fungere, og når vi da lager interfaces til tjenestene, blir

kontrollerene avhengige av abstraksjonen av tjenesten og ikke implementasjonen av tjenesten (S. Smith & Addie, 2020).

Fordelen med dette er at vi kan implementere tjenestene på forskjellige måter i kontrollerene. Dette gjør at vi kan benytte oss av dependency injection, som er veldig nyttig å bruke når vi skal teste applikasjonen (se 4.3.6). Det gir også applikasjonen fleksibilitet siden vi ikke trenger å endre implementasjonen av tjenesten, om vi ønsker å implementere den annerledes (S. Smith & Addie, 2020).

Implementering av dependency injection

Måten vi implementerer dependency injection i ASP.NET Core (se 2.2.4), er at vi bruker et interface til å abstrahere implementasjonen av avhengigheten. Figur 4.12 viser de ulike tjenestene med tilhørende interfaces.



Figur 4.12: Tjenester og tilhørende interfaces

I kodeliste 4.6 ser vi interfacet til EpicListService. Interfacet kan ses på som en kontrakt. Når en klasse implementerer dette interfacet kreves det at klassen implementerer metoden eller metodene definert i interfacet.

```
public interface IEpicListService
{
    Task<IList<WorkItem>> Get();
};
```

Kodeliste 4.6: IEpicListService.

I kodeliste 4.7 ser vi at vi implementerer metoden, `Task<IList<WorkItem» Get();`,

i tjenesten. Vi velger selv hvordan vi ønsker å implementere den, så lenge den returnerer riktig type `Task<IList<WorkItem»`.

```
public class EpicListService : IEpicListService
{
    public EpicListService()
    {
    }

    public async Task<IList<WorkItem>> Get()
    {
        var fetcher = new Fetcher();

        // Gir alle closed epics som har "barn"
        var query =
            "?$select=" +
                "System_Id" +
                ",Microsoft_VSTS_Common_ClosedDate" +
                ",System_Title" +
                ",System_WorkItemType" +
                ",HasChildren" +
            "&$filter=" +
                "System_WorkItemType eq 'Epic' " +
                "and Microsoft_VSTS_Common_ClosedDate ne null";


        await fetcher.Get<WorkItem>(EndPointEnum.WorkItemsHierarchy, query);

        var data = fetcher.GetResponse<WorkItem>();

        return data;
    }
}
```

Kodeliste 4.7: EpicListService.

Vi ønsker å kunne injisere tjenesten i andre moduler av applikasjonen. Tjenesten må da gjøres tilgjengelig for de delene av programmet som skal bruke den. For å få til dette registrerer vi tjenestene i startup.cs i metoden, `public void ConfigureServices(IServiceCollection services)`. Tjenestene registreres da i ASP.NET Core sin innebygde tjenestebeholder. Når vi registrerer tjenesten med `AddSingleton`, opprettes den første gang tjenesten blir forespurt. Alle påfølgende forespørsler vil da bruke denne instansen av tjenesten (S. Smith & Addie, 2020).

```
services.AddSingleton<IEpicListService, EpicListService>();
services.AddSingleton<IFeatureService, FeatureService>();
services.AddSingleton<IUserStoryService, UserStoryService>();
services.AddSingleton<IConsultantService, ConsultantService>();
services.AddSingleton<IInHouseService, InHouseService>();
services.AddSingleton<ITimeService, TimeService>();
```

Kodeliste 4.8: Fra metoden `ConfigureServices` i startup.cs.

Når tjenesten er tilgjengelig tjenestebeholderen, kan vi injisere tjenester inn i konstruktører til klassene der servicene blir brukt. Dette blir i vårt tilfelle i kontrollerne. Konstruktøren til `EpicController` i kodeliste 4.9, forespør en tjeneste av type IEpicListService, som den avhenger av. ASP.NET Core vil da opprette en instans av tjenesten når det utføres en forespørsel mot den kontrolleren, og kvitter seg med den når tjenesten ikke lenger er aktiv (S. Smith & Addie, 2020).

```
[Route("api/[controller]")]
[ApiController]
public class EpicController : ControllerBase
{
    readonly IEpicListService _service;

    public EpicController(IEpicListService service)
    {
        _service = service;
    }


    [HttpGet]
    // GET: api/Epics
    public async Task<IActionResult> GetEpics()
    {
        var epics = await _service.Get();
        var response = new List<SelectOptions>();
        foreach (var epic in epics)
        {
            var epicOption = new SelectOptions();
            epicOption.label = epic.System_Title;
            epicOption.value = epic.System_Id.ToString();
            response.Add(epicOption);
        }

        return Ok(response);
    }
}
```

Kodeliste 4.9: EpicController.cs.

Dependency injection i testing

Denne måten å utvikle applikasjonen på, kommer veldig godt med når applikasjonen skal testes. Siden kontrollerne våre nå ikke er avhengige av implementasjonen av tjenestene, men abstraksjonen av de, kan vi implementere "falske" tjenester. Dette er tjenester som implementerer interfacene som abstraherer tjenestene, og returnerer data som vi har lagt inn for å teste kontrollere. Vi kan da teste mot den dataen vi ønsker og framprovosere de situasjonene vi ønsker å teste.

I kodeliste 4.10, implementerer vi `IEpicListService` med falsk data. Dette er data som replikerer dataen som hadde blitt returnert fra 7pace sitt API.

```
class EpicListServiceFake : IEpicListService
{
    readonly List<WorkItem> _7paceApiData;

    public EpicListServiceFake()
    {
        _7paceApiData = new List<WorkItem>()
        {
            new WorkItem()
            {
                System_Id = 1,
                HasChildren = true,
                System_Title = "FakeEpic1",
                System_WorkItemType = "Epic",
                Microsoft_VSTS_Common_ClosedDate = "2016-5-08T10:20:13.894Z"
            },
            new WorkItem()
            {
                System_Id = 2,
                HasChildren = true,
                System_Title = "FakeEpic2",
                System_WorkItemType = "Epic",
                Microsoft_VSTS_Common_ClosedDate = "2018-10-09T11:30:27.267Z"
            }
        };
    }

    public async Task<IList<WorkItem>> Get()
    {
        return _7paceApiData;
    }
}
```

Kodeliste 4.10: EpicListServiceFake.cs.

I selve testen vil vi da injisere den falske servicen, som vist i kodeliste 4.11.

```
public class EpicControllerUnitTest
    {
        IEpicListService _service;
        EpicController _controller;

        public EpicControllerUnitTest()
        {
            _service = new EpicListServiceFake();
            _controller = new EpicController(_service);
        }

        .
        .
        .
```

Kodeliste 4.11: EpicControllerUnitTest.cs.

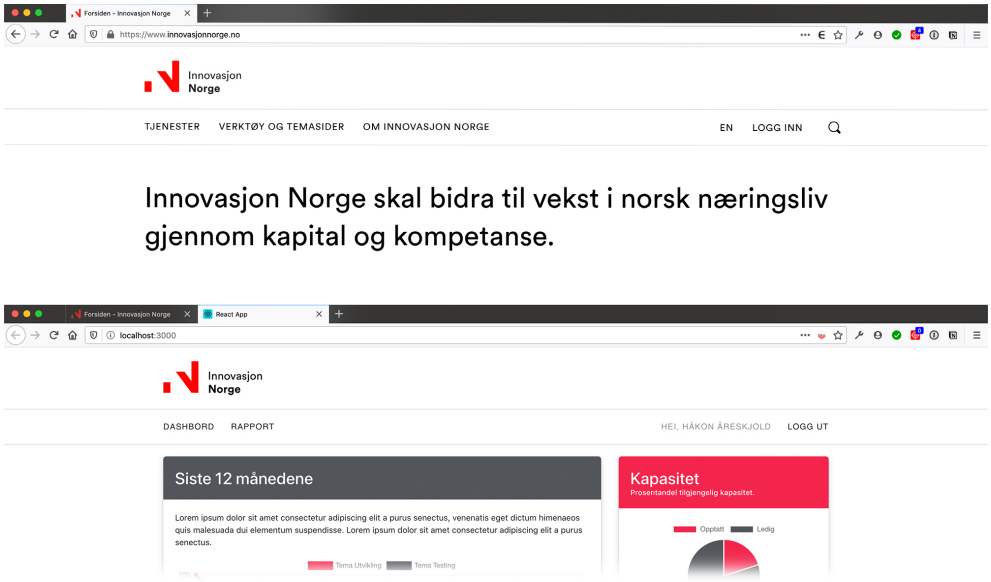
4.4 Klient

I dette kapittelet går vi gjennom hvordan klient-applikasjonen fungerer og dens oppbygging. Som nevnt tidligere i oppgaven er dette en React-applikasjon. Applikasjonen er avhengig av data fra server-applikasjonen vår.

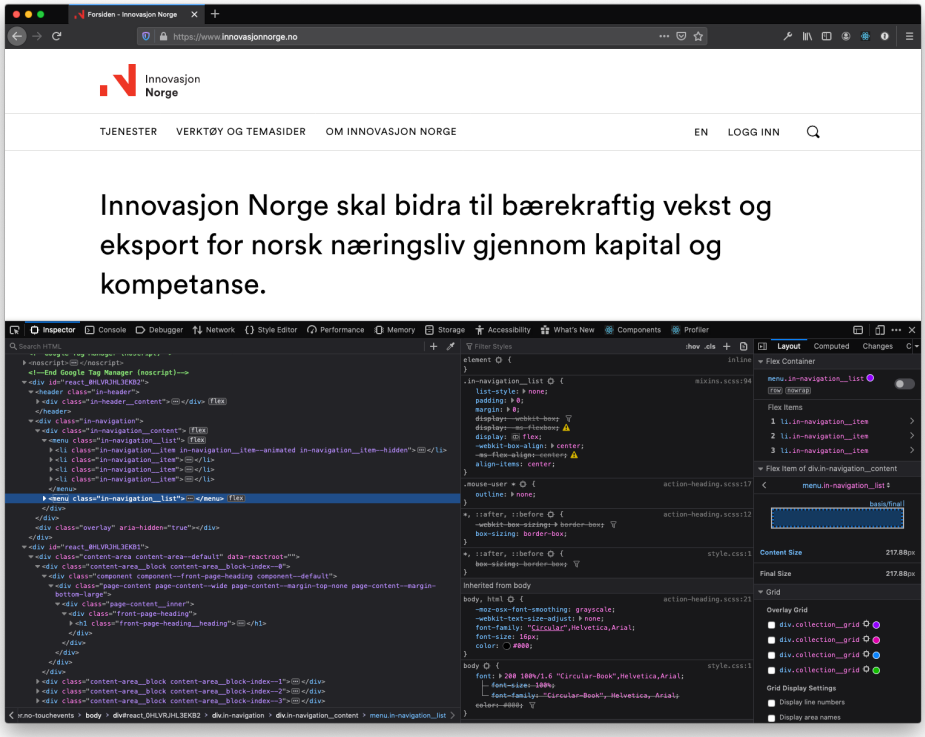
4.4.1 Komponenter og layout

Vi ønsket at ansatte ved Innovasjon Norge som bruker applikasjonen skulle "føle seg hjemme" og ønsket derfor at applikasjonen skulle være designet i Innovasjon Norge-stil.

Ved å benytte seg av en moderne nettleser er det mulig å se utdrag av både HTML og CSS til en nettside. På denne måten fikk vi et innblikk i hvordan ulike komponenter på nettsiden til Innovasjon Norge er bygget opp og designet (figur 4.14). Vi benyttet oss særlig av dette da vi lagde menyen til nettsiden. I figur 4.13 ser vi Innovasjon Norge sin og vår ovenfor hverandre. Dette var litt tidkrevende, men vi er fornøyd med resultatet. Menyen er også tilpasset mobil.



Figur 4.13: Sammenligning av Innovasjon Norge sin meny og menyen utviklet i forbindelse med prosjektet



Figur 4.14: Skjermbilde av inspektør-verktøy i nettleser

For å ha en responsiv nettside der elementer tilpasses etter skjermstørrelse valgte vi å bruke grid-løsningen til Bootstrap (se 2.2.4). Til React finnes biblioteket "reactstrap" (se 4.4.3). Dette biblioteket tilbyr Bootstrap-elementer som React-komponenter. For å få en strukturert side er kortene definert som kolonner i en rad. Se utdrag fra kode i kodeliste 4.12.

```
<Row>
    <Card
        color={Colors.Abbey}
        size="3"
        title="Fordeling"
        subtitle="Prosentfordeling"
        clickable={false}
        isChildrenChart={false}>
            <ConsHouse inHouse={inHouse} consultant={consultant}
            isDataLoaded={isLoadedConsultant && isLoadedInHouse}/>
    </Card>
    <Card
        color={Colors.Abbey}
        size="9"
        title="Timeliste for konsulenter"
        subtitle="Timeliste for konsulenter i den gitte perioden."
        clickable={false}
        isChildrenChart={false}>
            <TableOfConsultants consultants={consultant}
            isDataLoaded={isLoadedConsultant} />
    </Card>
</Row>
```

Kodeliste 4.12: Utdrag av kode som viser bruk av Grid-systemet til reactstrap

Kort

Gjennom hele applikasjonen møter man komponenter som fremstiller data. Disse komponentene er designet i stilen kjent som kort. Det vil si at hver komponent har sin tittel, undertittel, og innhold. Komponentene har samme oppbygging og har en skygge og ramme som enkelt skiller de fra hverandre. Dette er en ryddig og effektiv måte for å vise mye forskjellig informasjon på en side. Å bruke kort for å skille innhold er et kjent designprinsipp. Se skjermdump fra nrk.no i figur 4.16.

I kodeliste 4.12 ser man hvordan kort-komponenten blir brukt. Elementer lagt mellom `<Card>` og `</Card>` blir innhold i kortet. Selve komponenten har navnet `Card` og har følgende egenskaper:

color Egenskapen er påkrevd og definerer bakgrunnsfargen til tittel og undertittel.

size Egenskapen er påkrevd og definerer bredden på kortet. Verdien må være mellom en og 12 der 12 er full bredde.

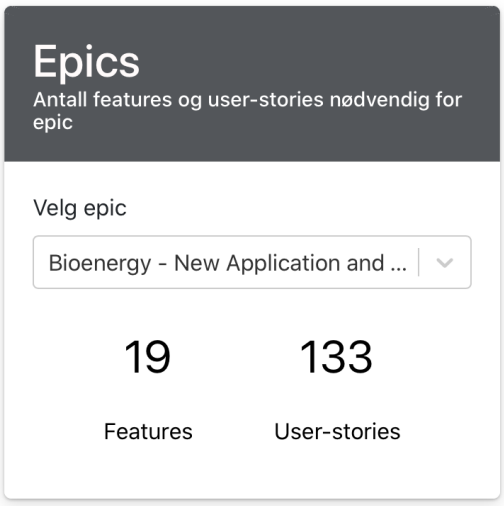
title Egenskapen er påkrevd og definerer tittel på kortet.

subtitle Egenskapen er valgfri og definerer undertittel på kortet.

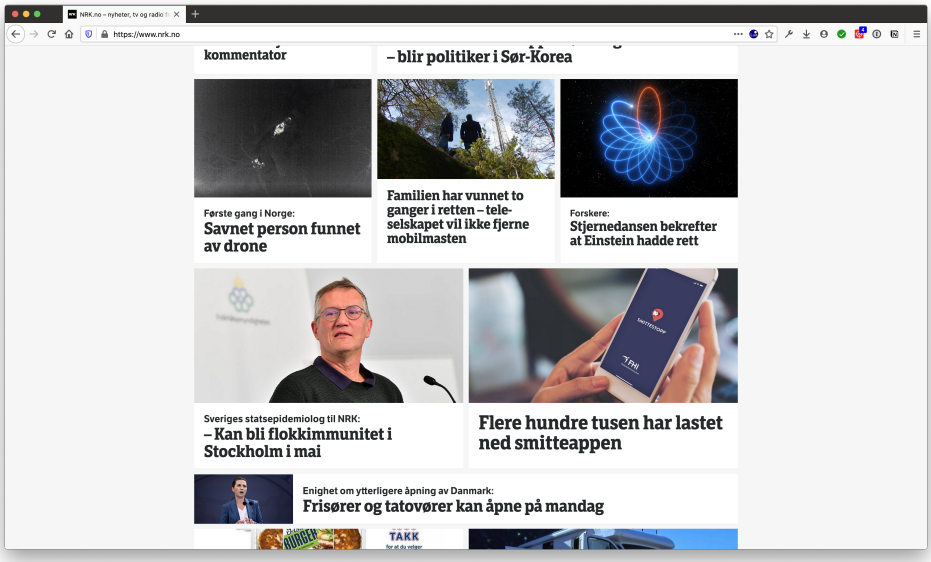
clickable Egenskapen er påkrevd og definerer om man kan klikke på kortet for å få det i fullskjerm.

isChildrenChart Egenskapen er påkrevd og skal være sann om kortet inneholder en graf. Verdien definerer plassering ved fullskjerm.

text Egenskapen er valgfri og definerer innhold i kortet.



Figur 4.15: Eksempel på komponent som er et kort.



Figur 4.16: Skjermbilde av NRK.no der kort blir brukt for å skille informasjon.

4.4.2 Datahenter

For å hente data fra APIet vårt utviklet i backend brukes Fetch API. Dette er et JavaScript-interface som tillater oss å sende forespørsler og motta respons fra nettet. Fetch API brukes ved å kalle på den globale metoden `fetch()` som lar oss hente ressurser asynkront over internett (MDN contributors, 2020).

Datahentingen gjøres i en funksjon som er `async`. Dette gjør at applikasjonen ikke stopper opp, men utfører forespørslene samtidig som den laster inn resten av applikasjonen. På denne måten lastes litt og litt inn, i stedet for at siden er hvit frem til all data er klar for å vises.

```
async function fetchUseStories() {
        setuserStoriesData(null);
        const res = await fetch("api/UserStories/" + epicData);
        res.json()
          .then(r => setuserStoriesData(r.response))
      }
```

Kodeliste 4.13: Utdrag av kode hvor vi henter inn data fra vårt API

I React versjon 16.8.0 ble hooks for første gang introdusert. Dette er en måte å lagre data på uten å måtte opprette klasser. `useState()` er et eksempel på en hook. Denne lar gir oss to variabler. En som gir verdien til variabelen og en som brukes til å sette verdien – altså en setter og en getter (Reactjs, 2020). Man kan også lage egne hooks.

```
export const useData = () => {
    const [data, setData] = useState<any>();
    const [isLoaded, setIsLoaded] = useState<boolean>(false);
    const reload = () => {
        setIsLoaded(false);
    }
    useEffect(()=>{
        if(data) {
            setIsLoaded(true);
        } else {
            setIsLoaded(false);
        }
    }, [setIsLoaded, data]);


    return [data, setData, isLoaded, reload]
}
```

Kodeliste 4.14: Utdrag av kode som viser vår useData-hook

Vi bruker useData (kodeliste 4.14), når vi henter data fra APIet vårt. Denne hooken gir oss to ekstra variabler, `isLoaded` og `reload`. `isLoaded` returnerer sant når data er satt og usant så lenge data ikke er satt. Dette er viktig da man ikke kan vise data før den er satt. `reload` brukes for å sette `isLoaded` til usann. Dette for å ta bort dataen, om ny data skal lastes inn. Dette trigger også symbolet som vises på skjermen når data lastes inn.

```
{prop.consultants.sort((a, b) => parseFloat(b.totalHours)
    - parseFloat(a.totalHours)).slice(1).map((row, index) =>
    <tr>
        <th scope="row">{index}</th>
        <td>{row.user}</td>
        <td>{row.totalHours}</td>
    </tr>
)}
```

Kodeliste 4.15: Utdrag av kode som viser data som er hentet fra APIet

4.4.3 Eksterne biblioteker

Her følger en oversikt over nyttige eksterne biblioteker vi bruker, og hva de brukes til.

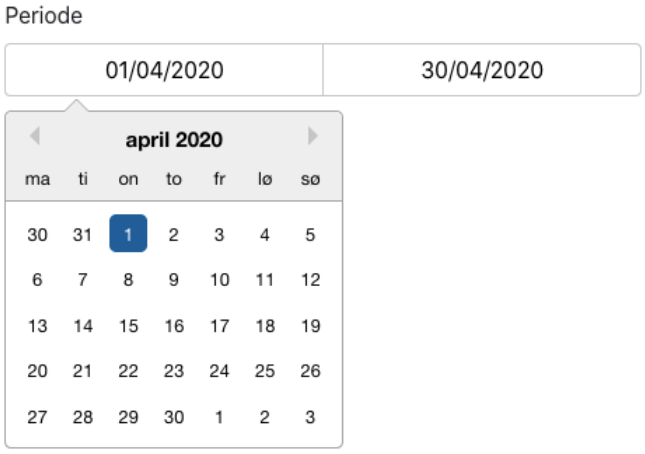
reactstrap

Dette er Bootstrap som React-komponenter. Brukes i hovedsak til grid og oppsett av layout. Grid er oppsett av rader og kolonner som definerer plassering av elementer. Dette er ofte responsivt, noe som sørger for at elementene holder sin

logiske plassering uavhengig av skjermstørrelse.

react-datepicker

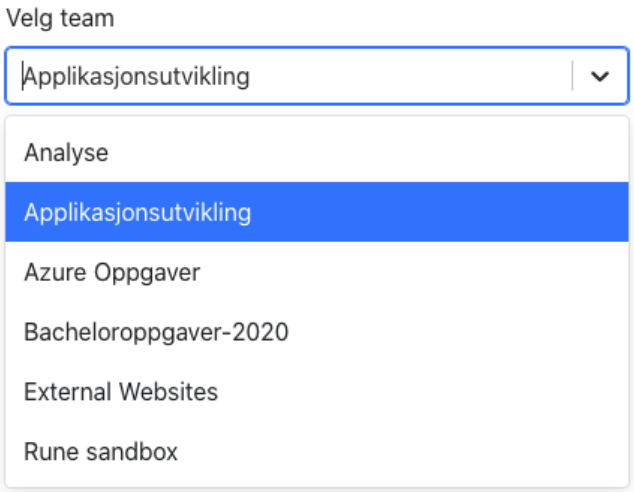
Datovelger som React-komponent. Brukes for å velge dato der det trengs.



Figur 4.17: Datovelger.

react-select

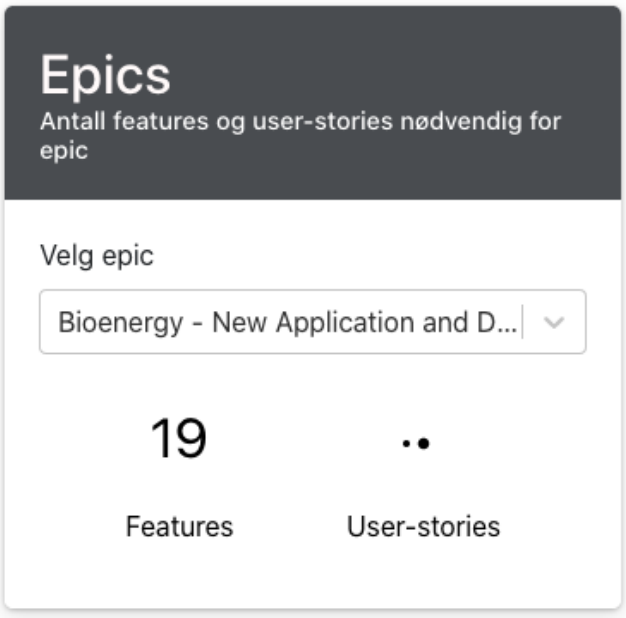
Nedtrekksliste som React-komponent. Brukes blant annet der man ønsker å velge prosjekt.



Figur 4.18: Nedtrekksliste.

react-loading

Animert innlastingsikon som React-komponent. Brukes når man venter på respons for å vise at applikasjonen jobber.



Figur 4.19: Animert innlastingsikon.

react-chartjs-2

Bibliotek for fremstilling av grafer som React-komponenter. Dette biblioteket ble anbefalt av Innovasjon Norge.

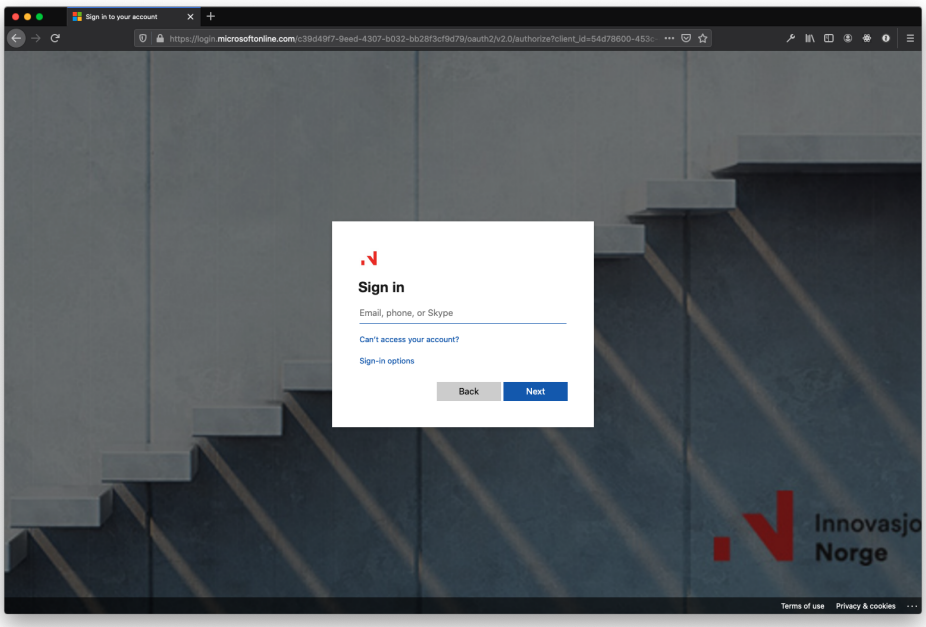
Kapittel 5

Brukermanual

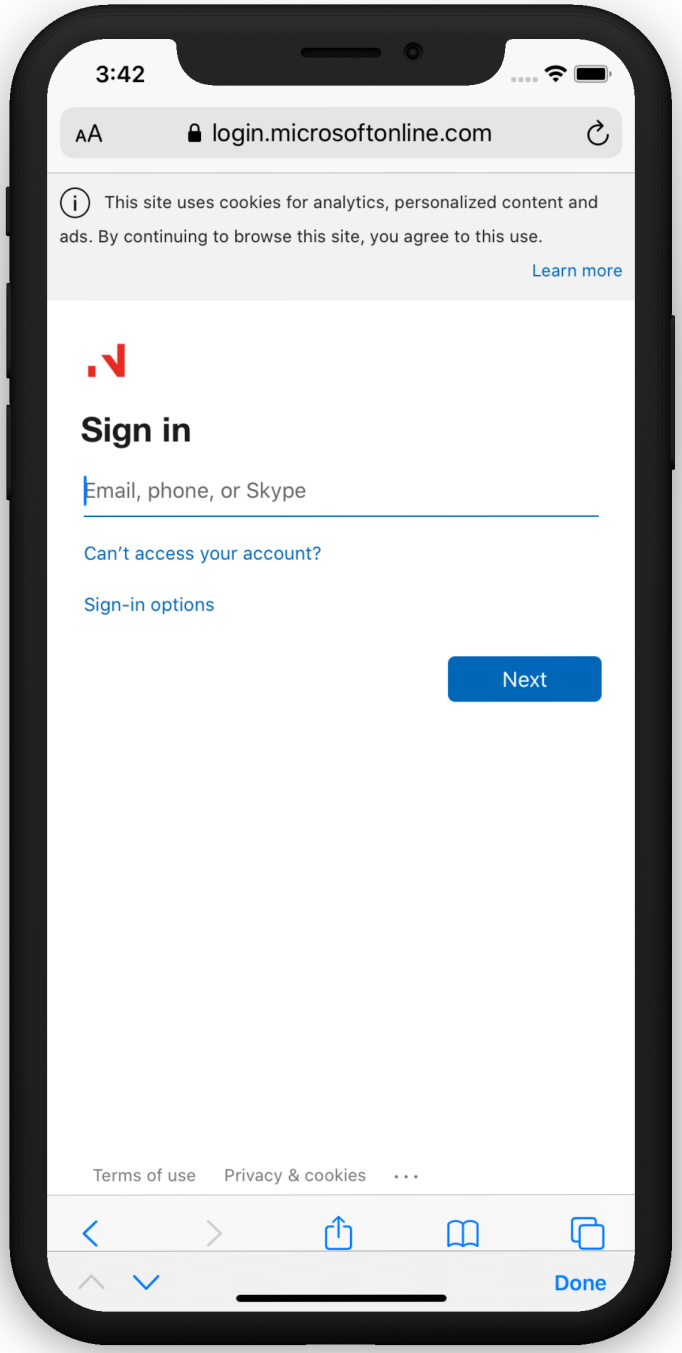
I dette kapittelet vil vi gå gjennom hvordan applikasjonen virker. Denne manualen går ikke i dybden på teknologi som er benyttet. For informasjon om dette henvises det til Utviklingsprosessen og Produktdokumentasjon.

5.1 Innlogging

For å besøke applikasjonen kreves innlogging med konto utstedt av Innovasjon Norge. Er man ikke innlogget sendes man hit når man besøker applikasjonen. Når innloggingen er gjennomført sendes bruker videre til applikasjonen.



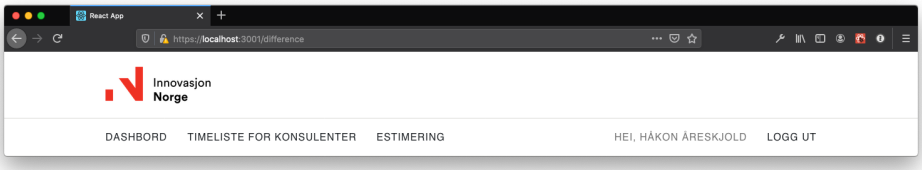
Figur 5.1: Innlogging via Microsoft på desktop.



Figur 5.2: Innlogging via Microsoft på mobil.

5.2 Meny

I menyen finner man oversikten over de andre sidene. Her vises også en hilsen med navnet til brukeren som er innlogget, samt mulighet for å logge ut av applikasjonen.



Figur 5.3: Meny på desktop.

Figur 5.4: Meny på mobil.

5.3 Sider

5.3.1 Dashboard

Dashbordet er kjernen til webapplikasjonen og det første brukeren møter når en er innlogget. Dashbordet gir et raskt overblikk over hvilken informasjon som er tilgjengelig. All informasjon finnes ikke her, men ved hjelp av menyen kan man navigere seg gjennom de ulike sidene.

Epics

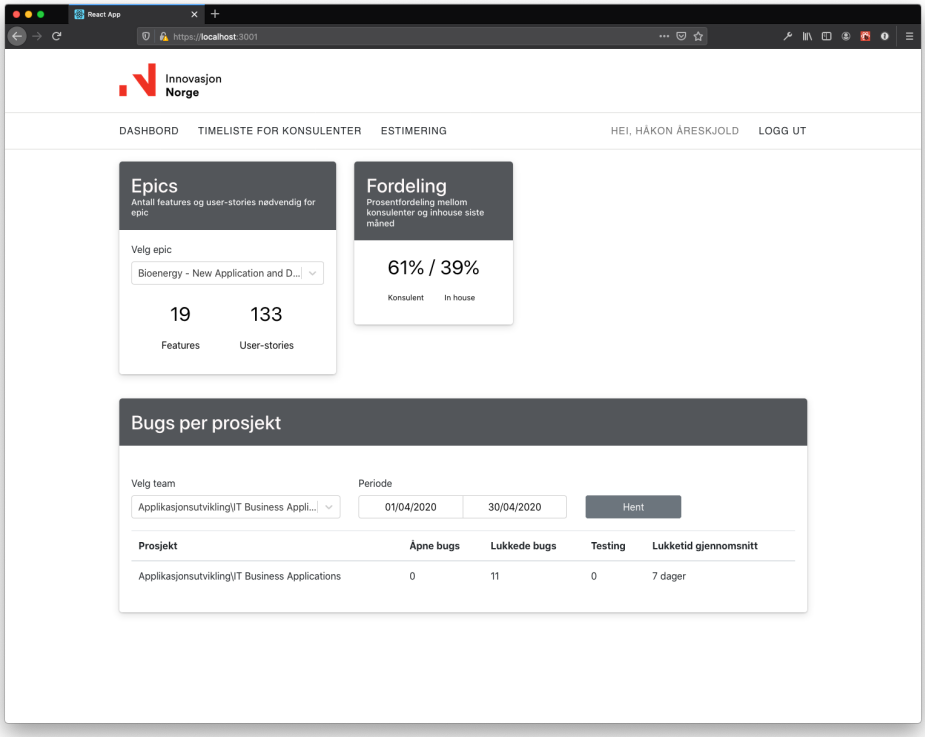
Kortet med navn “Epics” viser antall features og user stories tidligere epics har krevd. Dette er for å gi en indikasjon på hvor stort et tidligere prosjekt har vært når en skal planlegge fremtidige prosjekter.

Fordeling

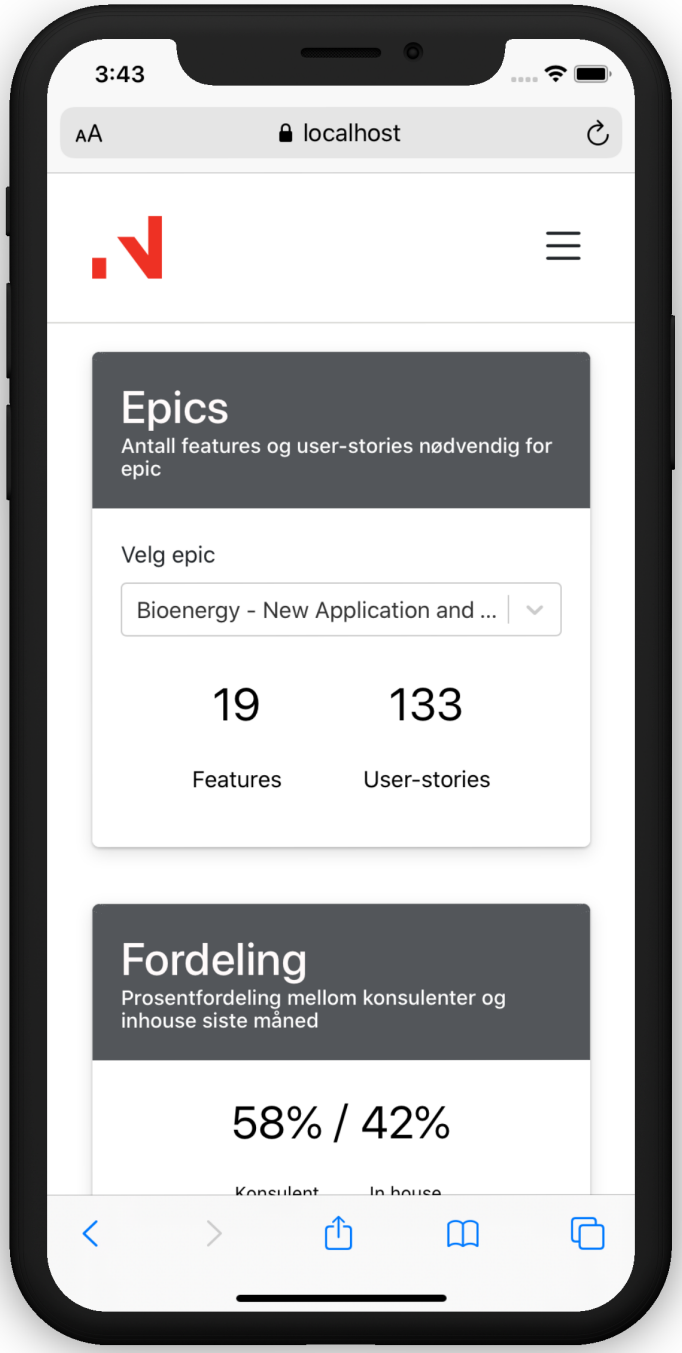
Kortet med navn “Fordeling” gir en indikasjon på fordelingen av antall timer de siste 30 dagene mellom Innovasjon Norge sine utviklere og konsulenter på deres prosjekt. Dette gir en indikasjon på hvor mye ressurser som hentes utenifra.

Bug per prosjekt

Kortet med navn “Bug per prosjekt” viser status på bugs i prosjektet i det gitte tidsrommet. En bug kan endre tilstand og siste gjeldende tilstand vil vises. “Åpne bugs” er bugs som er oppdaget men ikke behandlet. “Lukkede bugs” er bugs som er løst. “Testing” er bugs som er behandlet og sendt til testing før den eventuelt blir lukket. “Lukketid gjennomsnitt” er gjennomsnittstiden det tar fra en bug blir rapportert til den blir satt som lukket.



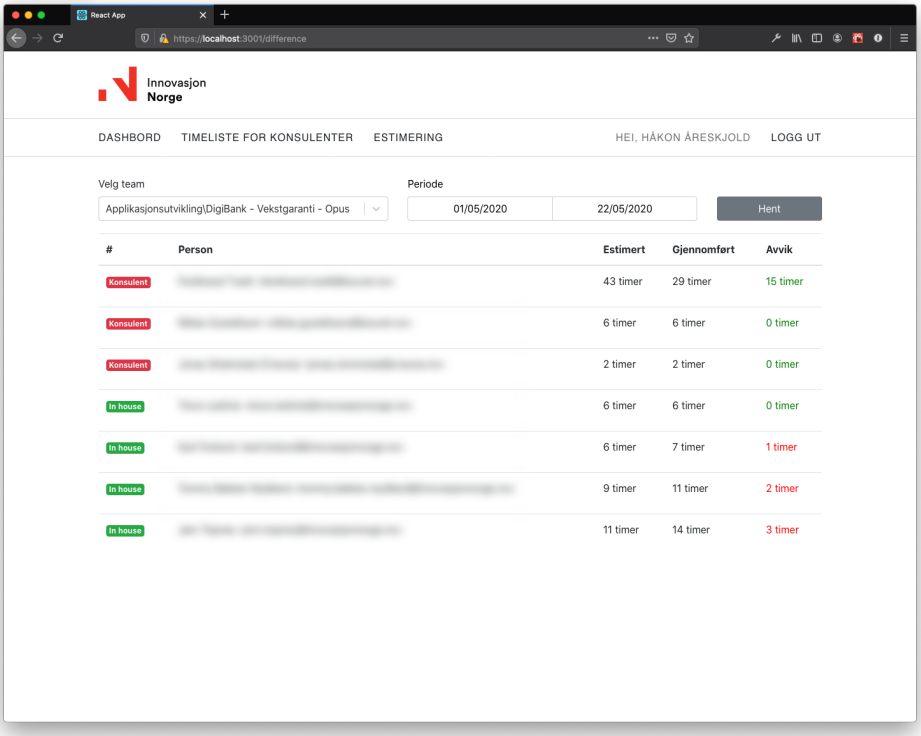
Figur 5.5: Skjermbilde av dashbord på desktop.



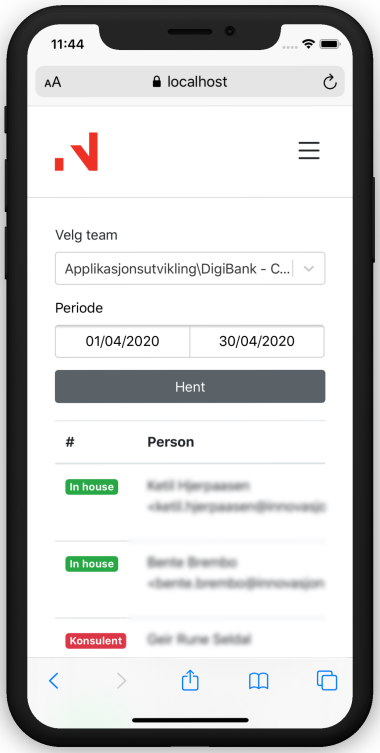
Figur 5.6: Dashbord på mobil.

5.3.2 Sammenligning mellom estimerte og gjennomførte timer

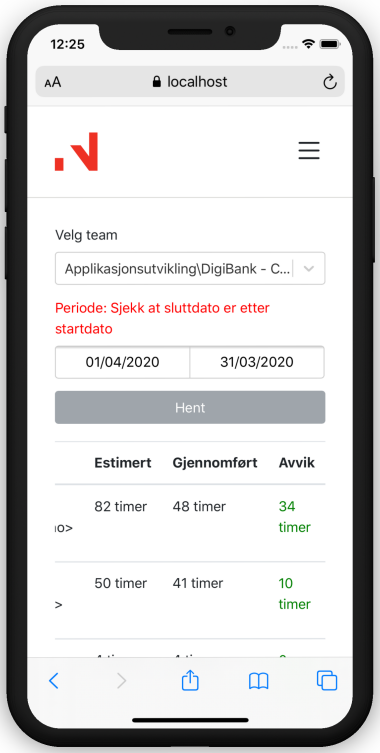
Denne siden viser antall timer utviklerne estimerer og faktisk bruker på oppgaver i den gitte perioden. For å hente data kreves det at man velger et team samt start- og slutt-tidspunkt. Velges en sluttdato som er før startdato vil en melding dukke opp og knappen for å hente inn data blir utilgjengelig. Dette illustreres i figur 5.7. Når man da trykker på hent-knappen vil tabellen oppdatere seg. Her får man en liste over utviklerne i teamet, hvor mange timer de har estimert, og hvor mange timer de har skrevet på oppgaver i tidsrommet. Kolonnen "Avvik" viser estimerte timer minus gjennomførte timer. Er tallet grønt betyr det at de har jobbet x antall timer mindre en estimert, er taller rødt har de jobbet x antall flere timer en estimert.



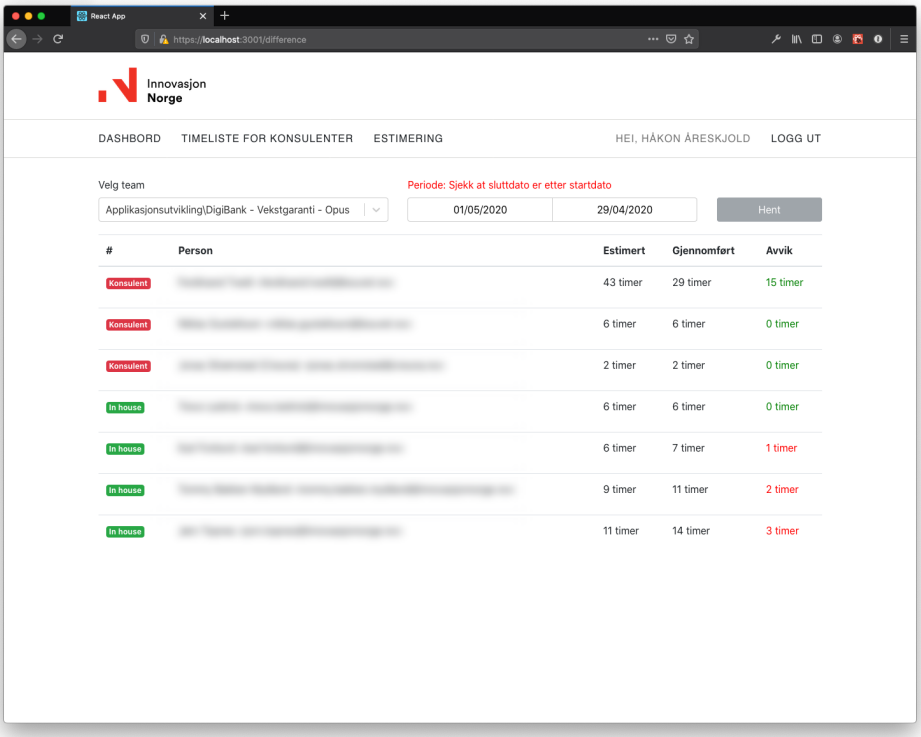
Figur 5.7: Side for sammenligning mellom estimert og gjennomførte timer på desktop.



Figur 5.8: Mobilversjon av siden for estimerte og gjennomførte timer.



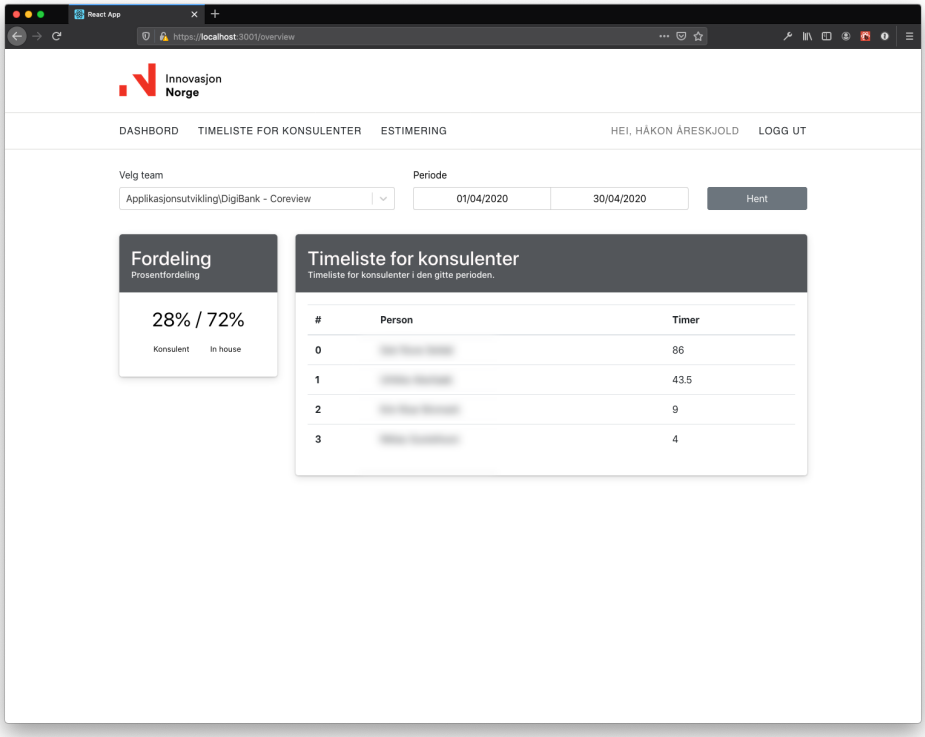
Figur 5.9: Tabellen scrolles horisontalt for å se all data. Det vises også feilmelding på mobil.



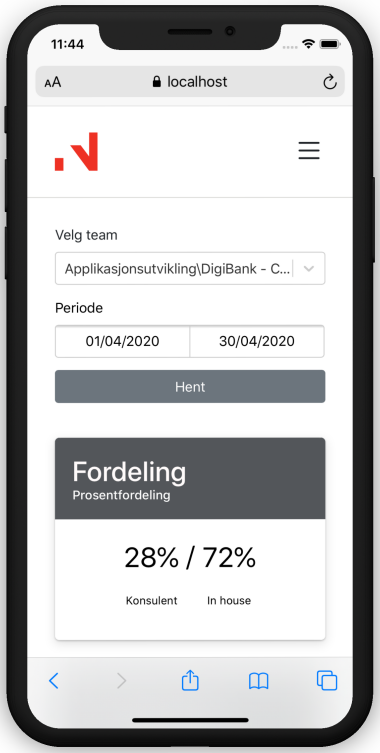
Figur 5.10: Advarsel om at sluttdato er før startdato.

5.3.3 Timeliste for konsulenter

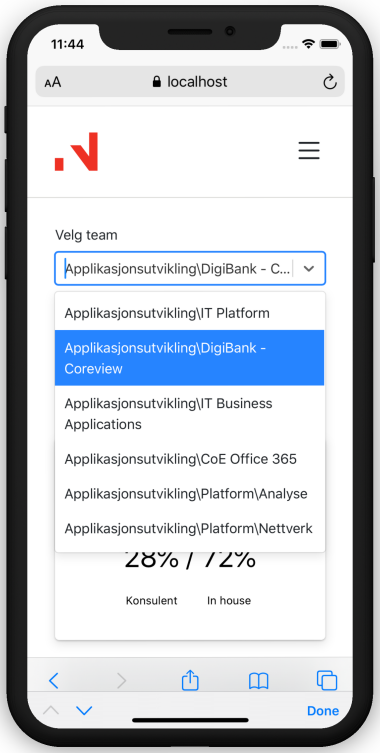
Kortet med tittel “Fordeling” forteller balansen mellom konsulenter og INs egne utviklere i det valgte teamet. Som utgangspunkt laster tabellen inn data fra forrige måned. Man kan enkelt velge hvilket tidsrom man ønsker data ved å bruke dato-velgerne på siden. Tabellen viser antall timer konsulentene har utført på oppgaver. Denne listen brukes også til å sammenligne de fakturerte timene fra konsulentene.



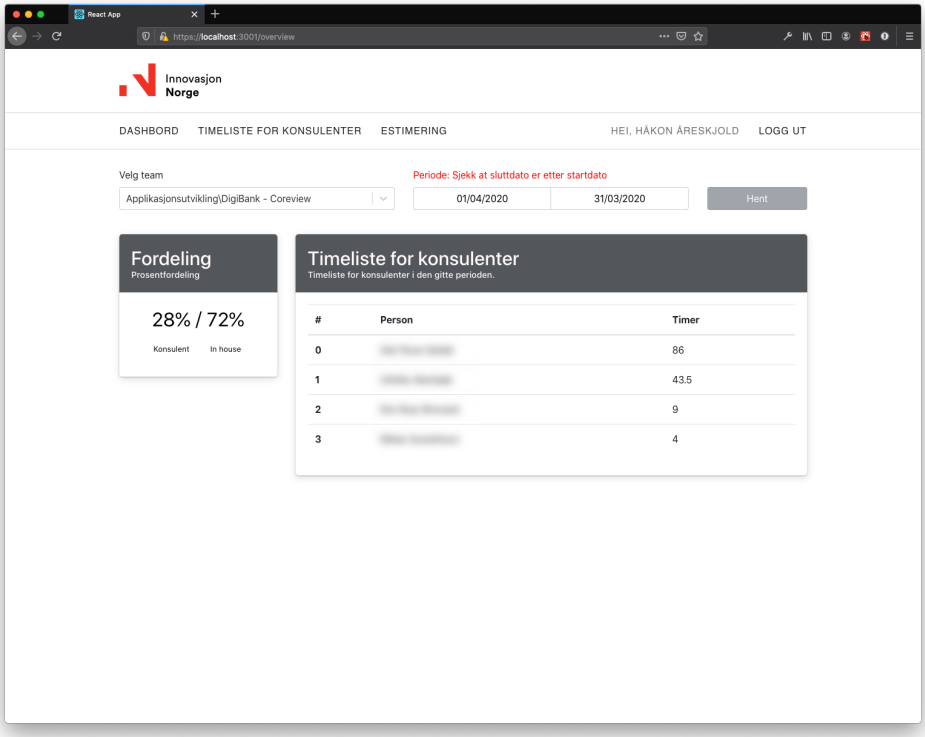
Figur 5.11: Side for timeliste for konsulenter på klient.



Figur 5.12: Side for timeliste for konsulenter på telefon.



Figur 5.13: Nedtrekksmeny for valg av team.



Figur 5.14: Advarsel om at sluttdato er før startdato.

5.4 Utlogging

For å logge ut av applikasjonen benyttes knappen “Logg ut” som finnes i menyen. Denne knappen sørger for at brukeren blir logget av fra sin konto på maskinen.

Kapittel 6

Evaluering og konklusjon

Avslutningsvis vil det oppsummeres i ulike deler gruppens tanker rundt prosjektet, dets resultat og veien videre. Konklusjonen skal belyse ulike valg gruppen har foretatt underveis, og utdype der gruppen finner det nødvendig. Kapittelet evaluering og konklusjon er delt opp i følgende deler:

- Prosess
- Produkt og Kravspesifikasjon
- Oppdragsgiver - Innovasjon Norge
- Veien videre

6.1 Prosess

Gruppen er godt fornøyd med måten vi har klart å gjennomføre prosjektet på, både med tanke på utfordringer knyttet til ny oppgave og endring i arbeidssituasjon som følge av Covid-19. Vi er glade for at vi har vært tre stykker som sammen har jobbet med prosjektet, da vi har kunnet støttet oss på hverandre og utvekslet kunnskap. Etter å ha tilegnet seg kunnskap fra ulike emner i 3 år har det vært givende å sette alt vi har lært i perspektiv i en større oppgave. Dette har vært med på å gi oss en større forståelse for arkitektur og planlegging.

Det har vært spennende og til tider utfordrende å jobbe med alle de ulike aspektene ved utvikling og planlegging av et så stort prosjekt som dette. Vi syns vi har landet godt på den andre siden og ser tilbake på et lærerikt semester.

6.2 Produkt og Kravspesifikasjon

Gruppen har laget en webapplikasjon som gir verdi til Innovasjon Norge i form av følgende generelle krav fra kravspesifikasjonen:

- Skal visualisere timer brukt opp mot timer estimert på work-item (task). ✓
- Vise hvor mange årsverk som er brukt på et prosjekt. ✓
- Skal kunne brukes til å estimere fremtidige prosjekt ✓
- Vise hvor mye tid som ble brukt forrige uke, måned, kvartal, år ✓
- Tid brukt på bugs, fra innrapportert til siste status ved fullført (Closed) ✓
- Vise estimeringsfeil forrige uke, måned, kvartal, år ✓
- Visualisere faktisk tid brukt mot estimert tid ✓
- Mulighet for filtrering og fordeling mellom faktisk utviklingstid og estimert utviklingstid i analysegrafer eller segmenter ✓
- Analyse av Epics med antall Features og UserStories nødvendig for leveranse ✓

Det finnes lignende applikasjoner, og f.eks. Power BI fra Microsoft er et produkt som er veldig likt i form av å presentere informasjon i dashbord. Ulempen med denne typen program er at det er lavkode og gir ikke større frihet til å hente data og visualisere enn det Microsoft tillater gjennom grensesnittet. Fordelen med vårt produkt er at det gir en stor frihet å ha noe som er produsert på egenhånd fordi man kan endre koden, når det måtte passe. Velger man en hyllevare så er det ikke gitt at man kan få ønsket tilleggsfunksjon, men nå kan i stedet Innovasjon Norge tilpasse produktet som de ønsker.

Sett opp mot kravspesifikasjonen gruppen laget i samarbeid med Innovasjon Norge har vi kommet frem til et produkt som vi er godt fornøyd med, og som vi føler samsvarer med mange av kravene som er stilt. Vi kom ikke i mål med en trendanalyse av tidligere prosjekter. Gruppen innså at det ikke ville bli nok tid til å bygge komponentene for en slik analyse, og prioriterte derfor å få på plass de andre komponentene. Hadde gruppen hatt en måned ekstra så ville dette vært gruppens neste prioritet, men det var viktig for gruppen å komme i mål med prosjektet på en helhetlig måte. Gruppen vurderte å søke om utsettelse, men gruppen anså ikke endring av oppgaven som en ekstraordinær grunn til å utsette oppgaven. Slike tilfeller finner man nok ofte også i arbeidslivet, og gruppen ønsket å gjennomføre utviklingsprosessen på normert tid, på tross av at man mistet en måned.

6.2.1 Utfordring ved APIet

Når gruppen i koden spesifiserer data som skal hentes fra 7Pace TimeTracker-API'et gjøres dette ved at det skrives en spørring som definerer dataen. Da 7Pace TimeT-

racker sitt API er en "svart boks" der spørringen definerer hva som skal hentes ut, kan det være vanskelig å vite om dataen som hentes inn faktisk er all dataen som er tilgjengelig. Gruppen opplevde at noen endepunkter leverte mer data enn andre uten at vi helt forstod grunnen til dette. Heller ikke Innovasjon Norge kunne svare oss på dette punktet og henviste oss videre til 7Pace TimeTracker. Det endepunktet som gav oss størst mengde data ble etter en stund midlertidig tatt ned grunnet problemer med ytelsen. Ved leveranse har endepunktet vært nede i over én måned, og når gruppen henvendte seg til 7Pace TimeTracker for å undersøke når endepunktet forventes tilbake på nett så var det uvisst. Innovasjon Norge vil nok i større grad kunne undersøke hvorvidt all data som kan hentes ut blir hentet ut, da de har alle tilganger som kreves.

Med dagens løsning, og per kundes ønske, blir dataen i dag aksessert gjennom API'et til 7Pace TimeTracker og dette API'et har vist seg å ha en oppetid som ikke er forenelig med å ha en løsning som ene og alene er avhengig av dataen derfra. På sikt vil det nok være fornuftig å hente data direkte fra Azure DevOps siden Innovasjon Norge allerede sitter på denne dataen. Gruppen diskuterte tidlig i løpet hvorvidt det ville være fornuftig å hente all dataen inn i en database, men gikk bort fra dette da vi fikk forståelse for at det ikke ville være noe problem å hente data direkte fra API'et til 7Pace TimeTracker.

Etter flere måneder med bruk av 7Pace TimeTracker har vi kommet frem til at det ville vært tjenlig å mellomlagre data. Ved flere anledninger har APIet vært delvis eller helt nede. Man ser også at flere av forespørslene tok lengre tid enn hva som er ønskelig. Hadde vi vært klar over dette tidligere ville vi lagret nødvendig data i en database. På denne måten hadde vi sikret at vi alltid hadde data raskt tilgjengelig for fremvisning og vi kunne med gitte intervaller hentet inn ny data uten at brukeren måtte vente på dette.

6.2.2 Testing av applikasjon

Testing av applikasjon for å verifisere og validere er en viktig del av programutvikling, og denne webapplikasjonen er intet unntak. Derfor ble gruppen informert av veileder tidlig i utviklingsfasen at Innovasjon Norge kom til å gjennomføre en rekke tester på applikasjonen før den kunne settes i produksjon. Med tanke på at vi måtte utvikle applikasjonen på kort tid, grunnet endringer i kravspesifikasjonen, valgte vi og prioritere å få på plass et minimal viable product. Videre gjorde vi produktet klart til testing ved å sette opp et testprosjekt. Vi har også brukt designmønster som gjør det enkelt for Innovasjon Norge å komme i gang med testing.

6.2.3 Designmønster

Etter hvert som applikasjonen ble større, fant vi ut hvor viktig det er å følge gode designprinsipper. Dette er med på å gjøre programvarekoden mer fleksibel og forståelig, og gjør det lettere å implementere ny funksjonalitet. Hadde vi startet på nytt hadde vi vert mer tydelige på hvilke designmønster vi skulle brukt når vi

programmerer, og vi hadde fulgt SOLID-prinsippene fra vi startet å skrive applikasjonen.

I fremtidig utvikling ville vi f.eks. implementert dependency inversion i `Fetcher`-klassen. Slik applikasjonen er nå, er tjenestene som bruker `Fetcher` avhengig av implementasjonen av klassen. For å gjøre applikasjonen mer fleksibel bør vi heller abstrahere implementasjonen av `Fetcher` til et interface og gjøre kontrollerne avhengige av interfacet, heller enn klassen.

6.2.4 Sikkerhet

Som nevnt i kapittel 2.4 så ønsket gruppen å gjennomføre en sikkerhetstest av applikasjonen. Dette gikk gruppen bort fra da det ikke var mulig å unngå å berøre Innovasjon Norge eller 7Pace TimeTracker sine APIer. Derfor besluttet gruppen å heller sette opp en analyse av sikkerheten i applikasjonen basert på OWASP Top 10 (OWASP & hblankenship, 2020) som er vedlagt. Gruppen undersøkte med sikkerhetsansvarlig hos Innovasjon Norge hvorvidt det skulle gjennomføres testing og fant ut at de setter bort testing av applikasjoner til eksterne. Dermed gikk gruppen ikke videre med testingen, siden det ikke var noen i Innovasjon Norge som satt på nok kompetanse til å fortelle oss at det er trygt å teste mot APIet.

6.3 Oppdragsgiver - Innovasjon Norge

Innovasjon Norge arrangerte bachelorskriving for første gang, og det var tydelig at de ønsket å gjennomføre på en god måte. De hadde satt av både ressurser og lokaler en dag i uken for å gi studentene gode rammer til å skrive oppgaven. Siden det var første gang de gjennomførte dette var det naturligvis noen barnesykdommer, men god innsats fra flere av de ansatte ble utfordringer raskt ble fikset. Vår største utfordring hos Innovasjon Norge var den stadige endringen av oppgaven. Oppgaven ble endret fire ganger fra vi ble enige om å skrive oppgave for Innovasjon Norge og dette stilte krav til omstilling fra gruppens side, noe vi føler vi taklet på en god måte. Leif Olav Alnes sa til NRK på 17. mai, "...Mennesker bruker for mye energi på å bekymre seg for scenarier som aldri kommer", og dette kjennetegner godt hva gruppen forsøkte å unngå i januar. Vi kanaliserte energien inn i oppgaver vi kunne gjøre, og utsatte de oppgavene vi måtte vente på. Hver gang oppgaven måtte endres valgte gruppen å fokusere på det positive ved å få en ny oppgave fremfor å tenke på den tapte tiden.

6.3.1 Oppgave

Til tross for positiv innstilling og hurtig omstilling så er det ikke til å stikke under en stol at vi gjerne skulle unngått å miste en måned for å jobbe videre med prosjektet. Det hadde også gjort jobben enklere å fått en tydelig kravspesifikasjon fra Innovasjon Norge fremfor å måtte lage denne selv og få den godkjent av ekstern veileder, spesielt når oppgaven endret seg flere ganger. Dette kan nok tilskrives

barnesykdommer, men det tar likevel bort tid fra oppgaven at gruppen selv måtte utarbeide kravspesifikasjon. Siden vi mistet så mye tid, er vi godt fornøyd med å ha oppnådd et resultat som samsvarer på mange av punktene i kravspesifikasjonen, og som vi mener kan tas i bruk av Innovasjon Norge. Samtidig er den også bygget på en måte som gjør at de kan utvide applikasjonen med ytterligere tilleggsfunksjonalitet om de skulle ønske det.

Siden løsningen vi utviklet var spesielt laget for ledelsen og prosjektledere, krevde det innspill fra nettopp denne gruppen. Det skulle vise seg utfordrende å få tilbakemelding på spørsmål knyttet til løsningen, og gruppen erfarte at det var vanskelig å innhente tilbakemelding fra avdelingsleder, mellomledere og prosjektledere. Det var vanskelig å få ansatte til å bry seg om noe de ikke har et personlig forhold til, og spesielt når det ikke er noe de er forpliktet til. Gruppen måtte til slutt be ekstern veileder om å gripe inn for å få svar, selv om han også strevde med å få svar. Svaret vi til slutt fikk var nok til å fylle ut kravspesifikasjonen, men ideelt så hadde både kravspesifikasjon og oppgave vært klar før vi startet arbeidet. Manglende tilbakemelding fra mellomledere gjør at det er vanskelig å vite om løsningen man har tenkt er en løsning som vil virke nettopp for denne gruppen, men vi har forholdt oss til kravspesifikasjonen og tilbakemeldinger fra veileder.

Gruppen etterspurte ofte kontakt med avdelingsleder, men det skulle vise seg vanskelig å oppnå. Veileder forklarte som oftest at han var opptatt, men at vi skulle forsøke uken etter. Gruppen hadde et ønske om å forstå Excel-arket som avdelingsleder benyttet for å holde oversikt over timer konsulenter jobbet. Ikke før 5. mai kom det i stand et møte hvor gruppen fikk innblikk i dette dokumentet, og fikk tatt en prat med avdelingsleder. Dette resulterte i en rekke andre "krav" som nok burde vært en del av kravspesifikasjonen, men som ekstern veileder ikke hadde oppfattet når han godkjente den i februar. Nå kom det frem at avdelingsleder ønsket å kunne sjekke totalt antall timer som konsulentene hadde jobbet over en gitt tidsperiode. Det var også ønskelig å se hvor mange features som gjenstod på en leveranse, hvor mange ansatte som er ledig, hva de forskjellige teamene trenger, og ha mulighet til å sammenligne nye prosjektforslag med tidligere gjennomførte prosjekter. Det ble vanskelig for gruppen å endre på alt det avdelingsleder ønsket, men vi valgte likevel å implementere noe av funksjonaliteten, selv om dette gikk på bekostning av funksjonalitet som var forespurt i kravspesifikasjonen.

6.3.2 Hjemmekontor

Som følge av utbrudd av covid-19 ble vi informert om at mulighet for å arbeide hos Innovasjon Norge opphørte som følge av påbud om hjemmekontor. Det gjorde også at samarbeidet med Innovasjon Norge ble mer utfordrende, siden man ikke hadde de samme mulighetene som tidligere for oppfølging. De fleste tilgangene var heldigvis på plass før covid-19 inntraff i Europa, men deler av oppgaven, som for eksempel komponentbiblioteket, måtte tilpasses på grunn av tilganger som fremdeles ikke var i orden. I perioden med hjemmekontor merket vi at kontakten

med Innovasjon Norge forsvant, og veileder hørte vi kun fra et par ganger over Teams. Vi forsøkte å stille spørsmål ved utfordringer, men grunnet lang responstid merket også gruppen at det var lite motiverende å søke hjelp hos ekstern veileder. Svarene fra ekstern veileder var ofte basert på synsing, og derfor var det ofte opp til gruppen selv å finne løsninger på utfordringer som oppstod. Dette opplevdes som frustrerende, men gruppen støttet hverandre på en god måte og sammen fant vi løsninger. I retrospekt burde gruppen kanskje bedt om en ny veileder med mer erfaring, og at tre måneder i jobben som nyutdannet kanskje ikke er nok tid for å lede bachelorgrupper, men endring av oppgave og covid-19 veldig tett på hverandre gjorde en slik forespørsel uhensiktsmessig.

6.3.3 Bruken av Azure DevOps

Vi fikk tidlig beskjed om at Innovasjon Norge ønsket vi skulle benytte oss av Azure DevOps til versjonskontroll og deployering av applikasjonen. I tillegg hadde de satt opp infrastrukturkoden vi skulle bruke til dette. Dette var på mange måter greit og det gjorde at vi slapp å skrive mye av denne koden selv. Problemet var at vi da ble avhengige av Innovasjon Norge for å fikse tilganger og sette opp ressursgrupper i Azure. Når vi i tillegg hadde tilgang på få folk i Innovasjon Norge som hadde peiling og tilgang til dette, måtte vi ofte vente lenge før problemer i Azure DevOps ble fikset. Selv om vi brukte tiden på andre ting som å utvikle applikasjonen lokalt, og skrive på oppgaven, var dette til tider frustrerende. I ettertid ser vi at det hadde vært flere fordeler med å gjøre mer av dette selv. Ved å ta mer ansvar for deployering, og sette opp utviklingsmiljøet selv, hadde vi hatt mer kontroll, og vi hadde vært mindre avhengige av Innovasjon Norge til å gi oss tilganger og fikse ressursgrupper.

6.4 Veien videre

Resultatet av denne oppgaven, nemlig webapplikasjonen, er noe vi mener kan implementeres og brukes som er verktøy for Innovasjon Norge. Det vil være fornuftig å utvikle nye komponenter til applikasjonen, og den er bygget slik at det enkelt kan legges til.

Det ene funksjonelle kravet som var ønskelig fra utviklernes side var “Som leder ønsker jeg å se “work load” for de ulike fasene; planlegging, utvikling og vedlikehold.” Dette er noe som hadde vært spennende å implementert for å bedre kunne hatt oversikt over de ulike fasene. Det vil kreve at Innovasjon Norge bruker “tags” på work items slik at man kan spore hvilken fase man befinner seg i. Foreløpig er ikke dette på plass, men om man innfører dette som rutine kan dataen enkelt hentes ut og vises.

Vi er alt tatt i betraktning fornøyd med gjennomføringen og resultatet. Det har vært utfordrende å skrive en oppgave i en tid hvor hverdagslige ting fort blir uviktig. Vi har følt at det har vært givende å jobbe sammen i denne tiden, og tenker at erfaringen vi har tatt med oss fra dette prosjektet er unik. Vi har heldigvis hold oss friske, og ser frem til normale tilstander.

Bibliografi

Antonio Lima, M. M., Luca Rossi. (2014). Coding Together at Scale: GitHub as a Collaborative Social Network. Hentet 3. februar 2020, fra https://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8112/8130

Aroraa, G. & Chilberto, J. (2019). *Hands-On Software Architecture with C# and .NET Core 3*. Livery Place 35, Livery Street, Birmingham, B3 2PB, UK.: PACKT Publishing Ltd.

Asch, S. E. (1956). Studies of Indepencdence and Conformity: I. A Minority of One Against a Unanimous Majority. *Psychological Monographs: General and Applied*, *70*(9), Whole No. 416.

Auth0. (2020). Introduction to JSON Web Tokens. Hentet 24. mai 2020, fra https://jwt.io/introduction/

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70–77.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). The Agile Manifesto. Hentet 11. februar 2020, fra https://agilemanifesto.org/

Bright, P. (2012). Microsoft TypeScript: the JavaScript we need, or a solution looking for a problem? Hentet 3. februar 2020, fra https://arstechnica.com/information-technology/2012/10/microsoft-typescript-the-javascript-we-need-or-a-solution-looking-for-a-problem/

Brown, Z. (2018). A Git Origin Story. Hentet 11. februar 2020, fra https://www.linuxjournal.com/content/git-origin-story

Command: apply. (2020). Hentet 17. april 2020, fra https://www.terraform.io/docs/commands/apply.html

Command: init. (2020). Hentet 17. april 2020, fra https://www.terraform.io/docs/commands/init.html

Command: plan. (2020). Hentet 17. april 2020, fra https://www.terraform.io/docs/commands/plan.html

Command: plan. (2020). Hentet 17. april 2020, fra https://www.terraform.io/docs/commands/plan.html

Copeland, L. (2019). 7pace Timetracker Overview. Hentet 11. februar 2020, fra https://support.7pace.com/hc/en-us/articles/115000524003-7pace-Timetracker-Overview

Currie, G. & Ravenscroft, I. (1997). Mental Simulation and Motor Imagery. *Philosophy of Science*, *64*(1), 161–180.

Danielson, S., KathrynEE, Jacobs, M., Batkoski, E., Staheli, D., thomps23, ... chcomley. (2019). What is Azure Pipelines? Hentet 6. mars 2020, fra https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops

Digitaliseringsdirektoratet. (2020a). 1.4.11 Kontrast for ikke-tekstlig innhold (Nivå AA). Hentet 8. mai 2020, fra https://uu.difi.no/krav-og-regelverk/webdirektivet-og-wcag-21/wcag-21-standarden/1411-kontrast-ikke-tekstlig-innhold-niva-aa

Digitaliseringsdirektoratet. (2020b). Kva seier forskrifta? Hentet 6. mai 2020, fra https://uu.difi.no/krav-og-regelverk/kva-seier-forskrifta#kvenskalfolge

Digitaliseringsdirektoratet. (2020c). Kva seier forskrifta? Hentet 6. mai 2020, fra https://uu.difi.no/krav-og-regelverk/eus-webdirektiv-wad

Ebert, C., Gallardo, G., Hernantes, J. & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3), 94–100.

Garg, S. (2020). Open Data Protocol (OData). Hentet 17. februar 2020, fra https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/data-entities/odata

Inc, F. (2020). Introducing JSX. Hentet 22. mai 2020, fra https://reactjs.org/docs/introducing-jsx.html#jsx-prevents-injection-attacks

Innovasjon Norge. (2019). Hva gjør vi? Hentet 17. februar 2020, fra https://www.innovasjonnorge.no/no/om/hva-gjor-vi/kort-om-oss/

Innovasjon Norge. (2020). Brand. Hentet 1. mai 2020, fra https://company-139306.frontify.com/d/YokG7g2Ch3dy/brand

Janis, I. L. (1971). Groupthink. *Psychology Today Magazine*, 84–90.

KathrynEE, Danielson, S., Jacobs, M., Staheli, D., Batkoski, E., Sherer, T., ... chcomley. (2019). Use Azure Pipelines. Hentet 23. mai 2020, fra https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops

Kulla-Mader, J., Batkoski, E., Danielson, S., Staheli, D., Jacobs, M., KathrynEE, ... thomps23. (2020). Use Azure Pipelines. Hentet 23. mai 2020, fra https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops

Kwok, J. (2015). Netflix Likes React. Hentet 3. februar 2020, fra https://netflixtechblog.com/netflix-likes-react-509675426db

Lamport, L. (1994). *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. 201 W. 103rd Street, Indianapolis, IN 46290: Pearson Education Corporate Sales Division.

Laverdet, M. (2010). XHP: A New Way to Write PHP. Hentet 3. februar 2020, fra https://www.facebook.com/notes/facebook-engineering/xhp-a-new-way-to-write-php/294003943919/

Marstrander, M. & Hovind, E. (2020). Integrating Security Controls Within a DevOps Pipeline. *mnemonic Security Report 2020*, 44–49.

MDN contributors. (2020). Using Fetch. Hentet 19. mai 2020, fra https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

Microsoft. (2006). Design Guidelines for Class Library Developers. Hentet 24. februar 2020, fra https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-1.1/xzf533w0(v=vs.71)?redirectedfrom=MSDN

Microsoft. (2020a). Hva er DevOps? Hentet 3. mars 2020, fra https://azure.microsoft.com/nb-no/overview/what-is-devops/

Microsoft. (2020b). What is ASP.NET? Hentet 3. februar 2020, fra https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet

Microsoft. (2020c). What is Azure Repos? Hentet 24. februar 2020, fra https://docs.microsoft.com/nb-no/azure/devops/repos/get-started/what-is-repos?view=azure-devops

OAuth. (2020). What is the OAuth 2.0 Authorization Framework and how it works. Hentet 17. februar 2020, fra https://auth0.com/docs/protocols/oauth2

Olsen, K., Posthuma, M. & Ulrich, S. (2018). ISTQB CTFL Syllabus 2018 V3.1. International Software Testing Qualifications Board.

OWASP & hblankenship. (2020). OWASP Top Ten. Hentet 28. april 2020, fra https://owasp.org/www-project-top-ten/

Pensgård, A. M. & Hollingen, E. (2006). *Idrettens mentale treningslære (2. utgave)*. St. Olavs plass, 0130 OSLO: Gyldendal Undervisning.

Reactjs. (2020). Hooks at a Glance. Hentet 19. mai 2020, fra https://reactjs.org/docs/hooks-overview.html

Resca, S. (2019). *Hands-On RESTful Web Services with ASP.NET Core 3*. Livery Place 35, Livery Street, Birmingham, B3 2PB, UK.: PACKT Publishing Ltd.

Roth, D., Anderson, R. & Luttin, S. (2019). Introduction to ASP.NET Core. Hentet 3. februar 2020, fra https://docs.microsoft.com/nb-no/aspnet/core/?view=aspnetcore-3.1

Rua, M. (2012). *Hva gjør fengselsleger?* Kristian Augusts gate 17, 0164 OSLO: Institutt for kriminologi og rettssosiologi i samarbeid med Akademika forlag.

Schonning, N., Wagner, B., Wenzel, M., Kulikov, P., Latham, L., Victor, Y., ... Addie, S. (2020). A tour of the C# language. Hentet 1. mai 2020, fra https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Schwaber, K. & Sutherland, J. (2017). The Scrum Guide. Hentet 11. februar 2020, fra https://www.scrumguides.org/scrum-guide.html

Sherer, T., Jacobs, M., Danielson, S., Wilson, C., KathrynEE & chcomley. (2019). What is Azure DevOps? Hentet 24. februar 2020, fra https://docs.microsoft.com/nb-no/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

Smith, P. S. (2006). The effect of solitary confinement on prison inmates: A Brief history and review of the litterature. *Crime and justice*, *34*(1), 441–528.

Smith, S. & Addie, S. (2020). Dependency injection in ASP.NET Core. Hentet 21. mai 2020, fra https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.1

Sommerville, I. (2016). *Software Engineering*. Edinburgh Gate, Harlow, Essex CM20 2JE: Pearson Education.

Stapleton, J. (2003). *DSDM: Business Focused Development*. Edinburgh Gate, Harlow, Essex CM20 2JE: Pearson Education.

Store norske leksikon. (2020). CSS. Hentet 12. mai 2020, fra https://snl.no/CSS

Verizon. (2020). 2020 - Data Breach Investigations Report, Whole report. Hentet fra https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf

Wenzel, M., Smith, S., Warren, G., Victor, Y., Schonning, N., Anil, N. & Veloso, M. (2019). Architectural principles. Hentet 20. mai 2020, fra https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles

Vedlegg A

Ordliste

En oversikt over teknologiske begrep som gjerne er forkortelser eller på engelsk og er vanskelig å oversette til norsk da disse er ord som brukes i bransjen.

A

Active Directory Active Directory, forkortet AD, er en Microsoft-tjeneste for håndtering av brukere, brukerrettigheter og ressurskontroll. Dette brukes til å autentisere brukere og maskiner.

Application Interface Program (API) Grensesnitt som gir tilgang til en del av applikasjonen for andre applikasjoner. Ofte en webapplikasjon som bruker APIet til en server som lagrer og leser til database.

Autentisering Bekrefte brukerens identitet.

Autorisering Når systemet vet hvem brukeren er gjennom autentisering, er autorisering hvordan systemet bestemmer hva brukeren kan gjøre.

Azure Azure er Microsofts plattform og infrastruktur for skytjenester.

B

Backend Begrep for serverapplikasjonene. Her skjer datahentingen mot 7Pace TimeTracker-APIet og behandling av data.

Branch I tilknytning til repository deler man gjerne opp i flere grener. Grenene utgjør en kopi av koden man har i versjonskontrollen, og man kan jobbe på en gren uten å endre koden i hovedgrenen.

C

Continous integration og continous delivery Refererer til kontinuerlig integrasjon, kontinuerlig levering, som er metoder å få hyppig leveranse av programvare til kunde.

CRUD Create, read, update og delete, forkortet CRUD, brukes om applikasjoner der formålet med applikasjonen er å lage, lese, oppdatere og slette data.

D

Desktop Betegnelse på datamaskin av typen bærbar eller stasjonær.

Deployere Sette en applikasjon til bruk ved å lansere den til en webserver.

F

Frontend Begrep for webapplikasjonen som kjøres i klienten.

I

Interface Kontrakt som definerer medlemmer, som klasser som implementerer kontrakten, må implementere.

J

JSON Er en måte å formatere dataobjekter på. JSON brukes for å sende informasjon mellom backend og frontend, der data er i nøkkel-verdi-par

K

Klient Med klient menes en nettleser som en bruker benytter for å besøke en nettside.

M

Merge Betegnelse man bruker når man fletter sammen flere branches for å samle koden.

Minimum Viable Product Minimum Viable Product (MVP) er en utviklingsteknikk med de viktigste komponentene for å bygge et fungerende produkt. Dette vil være et minimum av hva som produktet trenger for å være funksjonelt.

O

Open source Åpen kildekode. Kjennetegner at en programvares kildekode er åpent tilgjengelig for alle, og at alle kan komme med forslag til utbedringer eller bidra til å rette feil i koden.

P

Pipeline Et sett av operasjoner, satt i rekkefølge, for å levere programkode til en webserver.

R

Repository Et område hvor man lagrer kildekode i versjonskontroll.

Representational State Transfer Representational State Transfer (REST) er en type programvarearkitektur. Ved bruk av REST API-arkitekturen sender en applikasjon en forespørsel til en webserver som står klar og sender data i retur når den kalles på i form av tekst, ofte Json. Dette høres over http-protokollen.

Responsiv Vil si at innholdet tilpasser seg størrelsen på skjermen det vises på.

Reverse engineering Å gjenskape noe ved å se på det ferdige resultatet og nøste bakover. For eksempel ved å gjenskape et dataprogram ved å se på hvordan det ferdige resultatet er, og de deler av kildekoden man har tilgang til.

T

Token Et token er en nøkkel i form av en tekststreng som brukes for å få tilgang til APIer og andre ressurser. Token har en gitt levetid før den må fornyes. Dette kan være alt i fra 24 timer til et år.

W

Workitem En samlebetegnelse for forskjellige typer enheter som inneholder beskrivelser av arbeidet som skal gjøres.

Worklog Worklog er en "kvitteringen" på timer som er ført på et workitem.

Vedlegg B

Samarbeidsavtale

Samarbeidsavtale - bachelorprosjekt

Tidsrom: 07.01.2020 - 12.06.2020

Arbeidsdag

Tid og sted gjelder om ikke annet er avtalt.

Mandag
Sted: OsloMet
Klokkeslett: 09.00 til 15.00

Tirsdag
Sted: Innovasjon Norge
Klokkeslett: 09.00 til 15.00

Fredag
Sted: OsloMet
Klokkeslett: 09.00 til 15.00

Kommunikasjon

Kommunikasjon, dagbok og fillagring gjøres i Microsoft Teams.
Dagbok føres med en inndeling pr dag etter endt arbeidsdag.

Arbeidsmengde

I tillegg til avtalte arbeidstider forventes det at gruppemedlemmene fullfører arbeidsoppgavene innen avtalt tid. Dette kan medføre arbeid utenfor oppsatte arbeidsdager.

Prikksystem

Utgivelse av prikker

Årsak	Antall prikker
< 15 min for sent.	1
Pr påbegynte 15 minutter for sent.	1 (maks 3 pr dag)
Kritiske systemfeil.	1
Push av kode med feil til master.	1

Straff

Beskrivelse	Prikker
Kjøpe kaffe til gruppemedlemmene.	3
Bake kake.	5
Lage middag til gruppen.	10

Medlemmets prikker blir nullstilt når medlemmet har kommet til 10 prikker og innfridd straffen.
Prikksystemet kan endres om et flertall av gruppens medlemmer behovet for det.

Konflikthåndtering

Om en konflikt oppstår må den løses så raskt som mulig. Hvis konflikten gjelder arbeidsprioritering eller retningsendring av prosjekt må dette løses umiddelbart.

Kan det ikke løses innad i gruppen må det tas til veileder.

Om en konflikt mellom to gruppemedlemmer oppstår skal den tas opp i gruppen. Løses ikke konflikten der, skal den tas videre til veileder.

Avvikshåndtering ved gitte situasjoner

Utsagn	**Avvikshåndtering**
Gruppen må avgjøre tvilsspørsmål ved flertallsavgjørelser	Om et gruppemedlem er fullstendig uenig tas spørsmålet med til veileder.
Det forventes at medlemmene gjør sin del av oppgaven	Ved brudd på arbeidsoppgaver forventes det at medlemmet fullfører oppgaven. Ved gjentatte brudd kan tvisten tas med til veileder.
Ingen må dominere diskusjonen	Gis advarsel. Om advarselen ikke overholdes må personen forlate diskusjonen.
Det forventes at alle medlemmer nedlegger mellom 20-25 timer arbeid i uken.	Tapt arbeidstid skal tas igjen fortløpende. Hvis det utvikler seg til et problem, må dette løftes til veileder.
Alle skal møte til avtalt tid	Prikksystem
Alle har ansvar for arbeidet og fremdriften	Luftes i gruppen først, deretter til veileder.
Den som oppdager problemer, har ansvar for å ta dem opp til diskusjon	

Vedlegg C

Sikkerhetsanalyse av applikasjon

Analysen vil gjennomgå de mest kjente sårbarhetene ifølge The Open Web Application Security Project (OWASP) (OWASP & hblankenship, 2020) sett opp mot applikasjonen vår. OWASP er en organisasjon som jobber for å øke sikkerhet i programvare og tilbyr en rekke verktøy og informasjon på sine nettsider for å hjelpe bedrifter og enkeltpersoner i å forbedre sikkerheten i sine programmer. Analysen vil også gjennomgå ulike sårbarheter som applikasjonene potensielt sett kan være sårbar mot.

C.1 OWASP Top 10

OWASP Top 10 er en oversikt over de 10 mest sette cyber-angrepene eller sårbarhetene i verden det siste året. For hvert angrep gir OWASP en innsikt i angrepsvektorer, sårbarheter og hvordan angrepet kan påvirke hvis det skulle være vellykket.

C.1.1 Injeksjon

Utfordring

Utfordringer knyttet til injeksjoner mot vår applikasjon mitigeres heldigvis en del ved å benytte innloggingskontroll før applikasjonen er tilgjengelig for brukeren. Det betyr at man ikke skal få tilgang til applikasjonen før man har logget inn via bedriftens Azure Active Directory-løsning. På den måten kan vi beskytte applikasjonen mot de mest vanlige formene for angrep, nemlig injeksjoner. Dette er typisk SQL, NoSQL, LDAP og OS (OWASP & hblankenship, 2020), og kommer ofte som ferdige script som kjøres som skanning mot store mengder IP'er i samme slengen. På den måten kartlegges IP'er for svakheter, og ved å luke bort tilgang til applikasjonen fra å ligge åpent mot internett, vil det være vanskeligere å angripe den. I tillegg kjøres innlogging i applikasjonen via Microsoft Office, og inkluderer som nevnt at man må ha en brukerkonto hos kunden for å få logget inn. Dermed vil

Microsoft tilby første linje i forsvaret mot inntrengere, noe som gjøre at vi ikke får sanert input til innloggingen, men samtidig testes innloggingsløsningen fra Microsoft ofte.

Tiltak

Innlogging håndteres av Microsoft, og er utenfor vår kontroll. Denne innloggingen testes nok oftere og bedre av Microsoft enn hva vi kunne klart. Applikasjonen vår har ingen database, så det vil ikke være mulig å injisere noe i databasen for å hente ut informasjon. Derfor anses injeksjoner som mindre sannsynlig å være et problem i denne sammenheng. Ved å bruke React saneres også dataen automatisk ved at i Javascript så omgås alle verdier som er embedded i koden, og dermed skal det ikke være mulig å injisere. Med sanering av data menes at man ikke tillater f.eks. at man skriver enkelte tegn i tekstfelter som kan brukes i et injeksjonsangrep. Dette er f.eks. ', " og \, som alle er tegn som kan benyttes for å utføre SQL-angrep.

C.1.2 Brutt Autentisering

Utfordring

Siden applikasjonen er avhengig av innlogging, så er brutt autentisering en trussel som påvirker oss. Hvis innloggingsdetaljer er på avveie, eller at Innovasjon Norge har dårlig kontroll på brukere i Active Directory så kan det oppstå problemer med brutt autentisering som betyr at man ikke har kontroll på hvem som har tilgang til applikasjonen. Dette kan oppstå ved at brukere mister innloggingsdetaljer gjennom phishing eller at passord gjenbrukes. Ved gjenbruk skjer det ofte at utdaterte tjenester, hvor bruker har oppgitt epost og passord, blir hacket og frigitt på internett. Dermed kan en angriper ha tilgang til applikasjonen. Har en inntrenger fått tak i en administratorkonto er hele systemet kompromittert.

Tiltak

Det som vil være viktig for Innovasjon Norge er å ha kontroll på hvilke brukere som ligger i Active Directory og som skal ha mulighet til å logge inn. Videre er det viktig at de ansatte har et sunt forhold til passord og unngår gjenbruk av passord over flere plattformer. En passord manager vil kunne hjelpe de ansatte i jobben med å holde styr på alle ulike passord, uten at det går på bekostning av sikkerhet.

C.1.3 Sensitive data eksponert

Utfordring

Svakheter i applikasjoner og APIer kan gjøre at sensitiv data blir lekket som følge av feilkonfigurasjon eller manglende sikkerhetstiltak. For applikasjonen så henter den timelister hos Innovasjon Norge med informasjon om konsulenter og ansatte,

samt epostadresser og timer arbeidet. Dette er informasjon som potensielt kan benyttes til økonomisk vinning.

Tiltak

Trafikken til applikasjonen sendes kryptert og det brukes kun https av API'et til 7Pace TimeTracker. Svakheten i applikasjonen per dags dato er at security token som brukes for autentisering av applikasjonen ligger i applikasjonen og ikke i et hvelv.

C.1.4 XML Eksterne Entiteter (XEE)

Applikasjonene tar ikke inn XML-filer så dette er ikke relevant for applikasjonen

C.1.5 Brutt adgangskontroll

Utfordring

Skulle uvedkommende komme seg inn i bygningen og få tilgang på en datamaskin som er innlogget, så vil de også potensielt kunne komme seg inn i applikasjonen. Dette er svært vanskelig for applikasjonen å forhindre.

Tiltak

At brukerer ikke bruker automatisk utfylling av passord i nettleser kan være til hjelp hvis uvedkommende får tak i en datamaskin. Videre vil også det faktum at bygget krever kort og kode for å bevege seg rundt. Det er dog ikke så vanskelig å få tak i kort til bygget fra resepsjonen.

C.1.6 Feilkonfigurering av sikkerhetelementer

Utfordring

En av de vanligste problemene knyttet til sikkerhet, hvor angripere typisk utnytter maskiner eller klienter som ikke er patchet for å få tilgang. Applikasjonen vil være sårbar hvis uvedkommende får tilgang til maskinene.

Tiltak

Kontinuerlig patching av ansattes maskiner med påtvunget omstart, samt tydelig informasjon til ansatte om hvorfor dette er viktig kan motvirke trusselen. Applikasjonen skal gjennomgå testing internt hos Innovasjon Norge før den settes i produksjon, og der vil det kunne avdekkes sårbarheter og eventuelle tilfeller som ikke følger beste praksis.

C.1.7 Cross-Site Scripting (XSS)

Utfordring

Problemet oppstår når en applikasjon inkluderer data som ikke er til å stole på i en ny webside uten at innholdet valideres eller omgås. På den måten kan brukerdata stjeles ved at bruker sender informasjon til en annen nettside enn den man opprinnelig trodde man kommuniserte med.

Tiltak

Applikasjonene benytter ikke `.innerHTML`, og den er bygget med React noe som gjør at React DOM omgår alle verdier som er embedded før de rendres. Dette skal sørge for at man ikke kan injisere noe som ikke er eksplisitt skrevet i applikasjonen (Inc, 2020). Dette motvirker XSS-angrep, men det vil alltids finnes måter å omgå selv de beste sikkerhetstiltak.

C.1.8 Usikker deserialisering

Utfordring

Denne type angrep resulterer ofte i remote code execution (RCE), hvilket er meget alvorlig, men denne typen angrep er også sjeldne og krever ofte menneskelig interaksjon for å skreddersy angrepsverktøyene til målet. For vår del vil det være problemer hvis applikasjonen deserialiserer objekter som er ondsinnede eller har blitt tuklet med.

Tiltak

Som nevnt er denne typen angrep sjeldne, men det vil være viktig å utelukke deserialisering som en måte å komme seg inn i applikasjonen. Mellom APIet til 7Pace TimeTracker og vår applikasjon vil det oppstå en deserialisering når vår applikasjon mottar data. Skulle noen komme seg inn i kommunikasjonslinjen vil applikasjonen være sårbar. Dette bør være et viktig punkt i testingen av applikasjonen da dette er et av to punkt hvor det er mulig å komme inn i applikasjonen. Det andre punktet er innlogging.

C.1.9 Bruk av komponenter med kjente sårbarheter

Utfordring

Applikasjonen er laget med siste versjon av C#, ASP.NET Core og TypeScript, og er derfor ikke bygget på utgått teknologi. Dette gjør at eier av teknologien vil komme med oppdateringer jevnlig for å tette sikkerhetshull.

Tiltak

Ved å bruke denne typen nyere teknologi er komponentene ofte oppdatert. Når det i tillegg er et stort selskap som Microsoft som står bak teknologien, øker dette hyppigheten av oppdateringer. Videre vil det være opp til Innovasjon Norge å følge med på om komponentene som er brukt i applikasjonen forblir oppdatert.

C.1.10 Utilstrekkelig logging og monitorering

Utfordring

Vi kjenner ikke til hvordan Innovasjon Norge logger nettverkstrafikk eller monitorerer denne trafikken.

Tiltak

Innovasjon Norge vil selv måtte gå god for logging av nettverkstrafikk og monitorering, samt avgjøre hvorvidt de har tilstrekkelig med IDS og IPS i nettverket for å merke innbrudd.

Vedlegg D

Kravspesifikasjon

D.1 Sammendrag av krav

Prosjektet innebærer å lage en webapplikasjon som behandler og visualiserer data for å effektivisere utviklingsprosjektene til IT-avdelingen i Innovasjon Norge. Det er ønskelig å se nærmere på estimering og visualisering av prosjekter på ulike nivåer, og også kunne se hvor mange personer et prosjekt vil kreve. Dette vil hjelpe Avdelingsleder og prosjektledere når de skal planlegge fremtidige prosjekter.

Webapplikasjonen skal hente data fra API'ene til 7Pace TimeTracker for visualisering og Azure DevOps for estimering.

Det vil bli laget en trendanalyse for å undersøke hvor mye teknisk gjeld man sitter med etter sprinter, og hvordan estimering i fortiden har vært gjort. Basert på tidligere prosjekter er det ønskelig å kunne bruke informasjonen til å ta informerte avgjørelser for prosjektplanlegging i fremtiden.

Det nye systemet skal være en responsiv webapplikasjon som passer både mobile enheter og større skjermer.

En oversikt over tid brukt i prosjektet skal vise antall hoder som kreves, og ikke antall timer, da dette kan være misvisende og vanskelig å kalkulere med. Dette kan deles på både prosjekter og sprinter.

D.2 Mål

Lage en webapplikasjon som visualiserer utviklernes timer for å analysere og estimere prosjekt.

Lage en webapplikasjon som fremstiller tid brukt på prosjekt for å effektivisere utviklingsprosessen til IT-avdelingen.

Lage en webapplikasjon som analyserer og visualiserer tidsbruken for ansatte ved Innovasjon Norge IT-avdelingen.

D.3 Generelle krav

Dette er brukerkrav som beskriver tjenestene vi skal tilby.

- Skal visualisere timer brukt opp mot timer estimert på work-item (task).
- Vise hvor mange årsverk som er brukt på et prosjekt.
- Skal kunne brukes til å estimere fremtidige prosjekt
- Vise hvor mye tid som ble brukt forrige uke, måned, kvartal, år
- Vise estimeringsfeil forrige uke, måned, kvartal, år
- Vise uferdig arbeid forrige uke, måned, kvartal, år
- Vise hvor mange prosent man pleier å over- eller underestimere
- Visualisere graf som viser faktisk tid brukt mot estimert tid
- Filtrering på person og tid brukt på tildelte oppgaver tasks og bugs
- Mulighet for filtrering og fordeling mellom faktisk utviklingstid og estimert utviklingstid i analysegrafer eller segmenter
- Analyse av Epics med antall Features og UserStories nødvendig for leveranse
- Mulighet for visning av gjennomsnittlig tid brukt for Features og UserStories
- Tid brukt på bugs, fra innrapportert til siste status ved fullført (Closed)

D.4 Funksjonelle krav

Funksjonelle krav er beskrivelser av hvilke tjenester systemet skal tilby, hvordan systemet skal respondere på ulike inputs, og hvordan systemet skal oppføre seg i ulike situasjoner (Sommerville, 2016, s. 105). De funksjonelle kravene skal i sum beskrive hva systemet skal gjøre.

IN beskrev dette som systemkrav som skulle gi detaljerte beskrivelser av systemets funksjoner, tjenester og operasjonelle begrensninger. Altså definerer vi her hva som blir utviklet. I samarbeid med IN kom vi frem til følgende funksjonelle krav:

Estimere timer i prosjekt:

Prioritering: Høy

Brukermønster: Som leder vil jeg estimere hvor lang tid mine ansatte vil bruke på et prosjekt i timer.

Aktør: Leder

Trigger: Leder henter inn data fra tidligere prosjekt

Pre-betingelse: Tidligere prosjekt har lagret data riktig

Post-betingelse: : Leder estimerer fremtidige prosjekt basert på informasjon ervervet fra applikasjonen.

Normalt hendelsesforløp:

1. Leder åpner applikasjonen
2. Leder henter inn data fra ønsket prosjekt eller periode
3. Leder analyserer data
4. Leder estimerer nye prosjekt basert på tidligere gjennomførte prosjekter

Variasjon: Leder analyserer ikke gammel rapport og estimerer nytt prosjekt galt.

Estimere personer i prosjekt:

Prioritering: Høy

Brukermønster: Som leder vil jeg estimere hvor mange ansatte jeg vil trenge på et prosjekt.

Aktør: Leder

Trigger: Leder henter inn data fra tidligere prosjekt

Pre-betingelse: Tidligere prosjekt har lagret data riktig

Post-betingelse: : Leder estimerer fremtidige prosjekt basert på informasjon ervervet fra applikasjonen.

Normalt hendelsesforløp:

1. Leder åpner applikasjonen
2. Leder henter inn data fra ønsket prosjekt eller periode
3. Leder analyserer data
4. Leder estimerer nye prosjekt med riktig antall personer basert på tidligere gjennomførte prosjekter

Variasjon Leder analyserer ikke gammel rapport og estimerer nytt prosjekt galt.

Visualisere personlig data

Prioritering: Høy

Brukermønster: Som bruker vil jeg se estimert, gjenværende og brukt tid, samt feilmargin på estimering av mine oppgaver.

> **Aktør:** Bruker
>
> **Trigger:** Bruker åpner "min profil" i applikasjon
>
> **Pre-betingelse:** Bruker har registrert og estimert timer på oppgaver og applikasjonen har hentet data fra API.

Post-betingelse: : Bruker får innsikt i egen estimering av tid for sine oppgaver.

Normalt hendelsesforløp:

1. Bruker åpner applikasjonen
2. Klikker "Min profil".
3. Scroller ned til seksjonen for timer
4. Betrakter data
5. Bruker blir mer observant på antall timer

Variasjon Bruker har ingen timer registrert på noen oppgaver og får dermed ikke noen å studere.

Visualisere teamets / avdelingens data

Prioritering: Høy

Brukermønster: Som leder vil jeg se feilmarginer for estimert tid i mitt team eller min avdeling.

> **Aktør:** Leder
>
> **Trigger:** Leder åpner "teamsiden" i applikasjon.
>
> **Pre-betingelse:** Bruker har registrert og estimert timer på oppgaver og applikasjonen har hentet data fra API.

Post-betingelse: : Leder får innsikt i teamets estimering og brukt tid.

Normalt hendelsesforløp:

1. Leder åpner applikasjonen
2. Velger "Vis alle team"
3. Klikker på ønsket team som tar leder til "team siden"
4. Scroller til seksjonen for tid
5. Betrakter data

Variasjon Bruker har ingen timer registrert på noen oppgaver og får dermed ikke noen å studere.

Visualisere ferdig prosjekt

Prioritering: Høy

Brukermønster: Som leder ønsker jeg å se "work load" for de ulike fasene; planlegging, utvikling og vedlikehold.

Aktør: Leder

Trigger: : Leder åpner "teamsiden" i applikasjon.

Pre-betingelse: Bruker har registrert og estimert timer på oppgaver og applikasjonen har hentet data fra API.

Post-betingelse: : Leder får innsikt i teamets estimering og brukt tid.

Normalt hendelsesforløp:

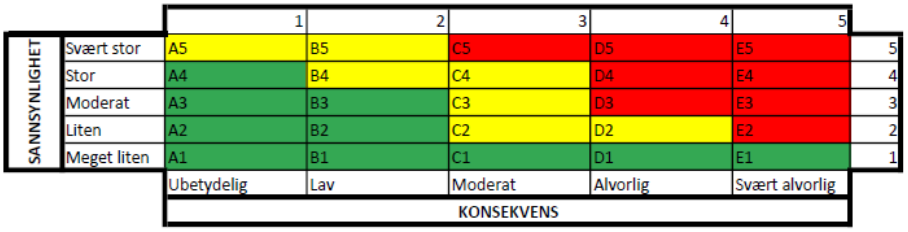
1. Leder åpner applikasjonen
2. Velger "Vis alle team"
3. Klikker på ønsket team som tar leder til "team siden"
4. Scroller til seksjonen for tid
5. Betrakter data

Variasjon Bruker har ingen timer registrert på noen oppgaver og får dermed ikke noen å studere.

D.5 Risiko

For å vurdere risiko knyttet til de forskjellige kravene har vi tatt i bruk en risikomatrise og vurdert samtlige utsagn opp mot sannsynlighet og konsekvens om hendelsen skulle inntreffe.

Figur D.1: Flytdiagram over hvordan de ulike delene i serveren snakker sammen.



D.5.1 Funksjonelle krav: risikovurdering

Risiko	Håndtering av risiko	Vurdering
Timer blir ikke ført, noe som gjør at dataene blir mangelfulle.	Timelister gjennomgås månedlig og sikrer at timer jobbet stemmer med timer registrert	B3
Dataene blir ikke hentet riktig, noe som fører til at dataene i tabellene blir feil, og man får en misvisende oversikt	Koden må gjennomgås av flere instanser og testes før den settes i produksjon slik at den henter riktig data	D2
API-tilkobling går ned og man kan ikke hente ut data.	På grunn av kobling mot 7Pace TimeTracker er det vanskelig å forebygget dette, men nedetid er som oftest kort	C3
Prosjektleder estimerer feil i et prosjekt fordi dataene er feil som følge av feil i applikasjonen. Kan påføre selskapet en stor kostnad som følge av teknisk gjeld.	Bedriften må ha rutiner for å sikre at tallgrunnlaget sjekkes før endelig avgjørelse fattes	E2

Tabell D.1: Risikovurdering av funksjonelle krav

D.6 Ikke-funksjonelle krav

Ikke-funksjonelle krav er begrensninger på systemet eller systemets funksjoner. Dette inkluderer tidsbegrensninger, hvilke utviklingsmetoder som brukes, og hvilke standarder som brukes (Sommerville, 2016, p. 105).

De ikke-funksjonelle kravene ble utarbeidet i samarbeid med IN og beskrives i følgende tabell:

Krav	Kravbeskrivelse	Prioritering
Brukervennlighet	• Skal være enkelt å forstå, slik at opplæring kan holdes til et minimum • Nettsiden skal respondere raskt så bruker slipper å vente • Systemet støtter generelt tilgang / operasjoner fra mobile enheter, som nettbrett • Støtter smarttelefon • Koden skal være enkel å forstå slik at den kan bygges på i etterkant	Høy
Universell utforming	• Skriften skal være leselig og ikke for liten • Grafene skal ha nok kontrast til at en som er fargeblind kan tydelig skille mellom dem • Det skal være god kontrast mellom tekst og bakgrunn • Knappene skal være tydelige og det skal ikke kreves presisjon for å klikke på knapper	Middels
Tilgjengelighet	• Applikasjonen skal være responsiv og skal kunne brukes på store og mindre skjermer.	Middels

Krav	Kravbeskrivelse	Prioritering
Estetiske krav	• Applikasjonen skal bruke IN sin grafiske profil. • Det skal brukes komponenter fra IN sitt react-komponent-bibliotek. • Grafene skal bestå av både visualiseringer og tekst så grafene er lette å forstå.	Høy
Sikkerhet	• Bruker må være innlogget for å kunne se data.	Høy
Ytelse	• Poengsummen til webapplikasjonen skal være mellom 50 på Lighthouse Scoring Guide.	Lav
Pålitelighet	• Applikasjonen skal ha en oppetid på 99% såfremt API-tilkobling virker.	Lav
Robusthet	• Applikasjonen skal ikke bryte sammen som følge av feil i koden.	Høy
Prosesskrav	• Systemet skal utvikles med smidige metoder. • Backend utvikles i ASP.Net Core 3 • Frontend utvikles i React med TypeScript. • Visual Studio benyttes for programmering i ASP.Net Core 3 • Visual Studio Code benyttes for programmering i React.	Høy

Tabell D.2: Ikke-funksjonelle krav

D.7 Ikke-funksjonelle krav risikovurdering

No	Risikofaktor/problem
1	Om siden ikke utformes universelt kan det være til hindring for ansatte med synsproblemer.
2	Prosessen følger ikke smidig utvikling, og ender opp med et resultat som kunden ikke har bedt om.
3	Applikasjonen har lav brukervennlighet og bruker trenger en betydelig mengde opplæring for å bruke applikasjonen.
4	Nettsiden responderer tregt, så bruker benytter unødvendig lang tid i applikasjonen.

Tabell D.3: Risikovurdering av ikke-funksjonelle krav

Risiko	Håndtering av risiko	Vurdering
Om siden ikke utformes universelt kan det være til hindring for ansatte med synsproblemer	Siden må følge standard internt for universell utforming før den settes i produksjon	B2
Prosessen følger ikke smidig utvikling, og ender opp med et resultat som kunden ikke har bedt om.	Gruppen må holde fokus på metodikk underveis og gjennomføre smidig utvikling bevisst	B1
Applikasjonen har lav brukervennlighet og brukertrenger en betydelig mengde opplæring for å bruke applikasjonen.	Gruppen må ha brukervennlighet i bakhodet når applikasjonen utvikles	C3
Nettsiden responderer tregt, så bruker benytterunødvendig lang tid i applikasjonen	Gruppen må skrive kode som gjør applikasjonen responsiv, og enkel å ta i bruk	D2

Tabell D.4: Risikovurdering av ikke-funksjonelle krav