NTNU

Kunnskap for en bedre verden

Bacheloroppgave

Utvikling av et utlånssystem for Dokflyt AS

16. mai 2017

Bachelor i webutvikling

Forfattere

Toni Sucic

Geir Marius Kirkeberg Jansson

Sammendrag

Dato: 16/05-17

Tittel	Utvikling av utlånssystem for Dokflyt AS
Deltakere	Toni Sucic Geir Marius Kirkeberg Jansson
Veileder	Anders-Petter Andersson
Oppdragsgiver	Dokflyt AS, Raymond Dalsborg
Stikkord	Web app, single page app, design, programmering
Antall sider	80 (63 + 17)
Antall vedlegg	3
Publiseringsavtale inngått	Ja

Beskrivelse av bacheloroppgaven

Vi fikk i oppgave å bygge en webapplikasjon for bedriften Dokflyt AS med formål om å gjøre det enklere og mer oversiktlig for montører i en bedrift å låne utstyr. Via applikasjonen kan man låne utstyr, se utstyr som allerede er utlånt, ta utstyr ut til service og inkludere merknader om hvordan servicen gikk, samt få kontaktopplysningene til personen som har lånt et gitt utstyr.

Abstract

Date: 16/05-17

Title	Utvikling av utlånssystem for Dokflyt AS
Participants	Toni Sucic Geir Marius Kirkeberg Jansson
Supervisor	Anders-Petter Andersson
Employer	Dokflyt AS, Raymond Dalsborg
Keywords	Web app, single page app, design, programmering
Number of pages	80 (63 + 17)
Number of appendices	3
Availability	Open

Description

We were tasked with building a web application for the firm Dokflyt AS with the aim of making it easier and neater for fitters in a firm to borrow equipment. With the application one can borrow equipment, see which equipment has already been borrowed, perform service on equipment, include notes about how the service went, and get the contact information of the person who has borrowed a given piece of equipment.

Forord

Vi ønsker å gi en stor takk til Dokflyt for et fabelaktig samarbeid og for alltid å være tilgjengelige med konstruktive tilbakemeldinger. Vi ønsker også å rette en takk til vår veileder for gode råd og til alle medstudenter for alt av hjelp vi har fått.

– Toni Sucic, Geir Marius Kirkeberg Jansson

Innholdsfortegnelse

Figurer

1 Introduksjon

I dette kapittelet skal vi gjøre rede for prosjektet vi har jobbet med. Det inneholder opplysninger om hva det dreide seg om, hvilke rammer vi jobbet innenfor, og hvilke mål vi satte oss for prosjektet.

1.1 Rapportstruktur

Kapittel 1: Introduksjon
Diskutere hva prosjektet gitt ut på, beskrive prosjektet, presentere målgruppen, vise hvilke mål vi hadde for prosjektet, samt vår faglige bakgrunn.

Kapittel 2: Metoder
Gjennomgå de metoder vi tok i bruk for å gjennomføre prosjektet og svare på problemstillingen.

Kapittel 3: Resultat
Gjennomgå resultatene til prosjektet, det vil si analyser, intervju, testing, og prototyper.

Kapittel 4: Diskusjon
Diskutere prosjektet vi har jobbet med med tanke på samarbeidet, problemstillingen, prosessen, og resultatet.

Kapittel 5: Konklusjon
Konkludere rapporten med en oppsummering av prosjektet, og en konklusjon av samarbeidet og resultatet.

1.2 Prosjektbeskrivelse

Formålet med dette prosjektet er å utvikle en fungerende, digital prototype i form av en responsiv (mobiltilpasset) webapplikasjon som skal vise hvordan man potensielt kan erstatte en utlånsløsning som taes i bruk i elektro- og byggebedrifter. Sluttproduktet kommer altså til å være en webapplikasjon som består av en front-end som tar i bruk Angular 2 med TypeScript, Sass, og HTML, samt en back-end skrevet i C# som tar i bruk ASP.NET Core MVC med Entity Framework for å persistere (lagre) data. Back-enden fungerer som et API for front-enden.

1.2.1 Problemstilling

Hvordan lage en digital løsning for web på mobil som er mer brukervennlig, effektiv, og medfører færre menneskelige feil ved reservasjon av utstyr?

1.3 Målgruppe

Hovedmålgruppen til webapplikasjonen er montører (bl. a. elektrikere og tømrere) som jobber ute i felten. I tillegg til hovedmålgruppen er det en mindre målgruppe som er saksbehandlere (disse er ansvarlige for å legge inn utstyr i applikasjonen).

1.4 Mål

Målet med oppgaven er å ha et utlånssystem som fungerer bedre, og er lettere å bruke enn den eksisterende utlånsløsningen, altså må det være enklere enn å skrive seg inn i en loggbok. Det skal også øke effektiviteten av å reservere utstyr i fremtiden, sjekke hvem som har brukt utstyret, og sjekke når det skal på service.

1.4.1 Effektmål

- I løpet av første kvartal 2018 skal applikasjonen fullstendig erstatte den eksisterende løsningen.
- Det skal ikke lenger være nødvendig å ha notatbøker for varene som skal lånes ut, som også vil medføre at folk ikke vil måtte ringe hverandre for å sjekke om opplysningene i notatbøkene stemmer.
- Varer med feil skal kunne fikses raskere, og feil vil kunne rapporteres raskere enn før.

1.4.2 Resultatmål

- Utvikle en webapplikasjon som skal gi arbeiderne bedre oversikt over reservasjoner og effektivisere prosessen med å reservere utstyr til prosjekter. Tidsramme: 3-4 måneder fra oppstartsdato.
- Gjøre applikasjonen brukervennlig gjennom gjentatt brukertesting.
- Inkludere et system for rapportering av feil eller mangler hos varene.

1.5 Faglig bakgrunn

Vi er to webutviklere som går studiet Bachelor i Webutvikling ved NTNU i Gjøvik. Hver av oss har våre styrker og svakheter. Dette var en fordel i forbindelse med utviklingen av webapplikasjonen, siden Geir Marius har mye front-end erfaring mens Toni har mye back-end erfaring. Ved å kombinere ferdighetene våre kunne vi fordele arbeidsoppgavene på en god måte og jobbe parallelt.

1.6 Terminologi

SPA: Akronym for **S**ingle **P**age **A**pplication. En webside som kun lastes inn én gang og består av én side som manipuleres med kode som kjører i nettleseren. Alle påfølgende etterspørsler til serveren gjøres i bakgrunnen istedenfor å laste inn hele siden på nytt hver gang du besøker en ny side. Dette gir brukeren inntrykket av en mye raskere nettside som føles mer som en app enn en tradisjonell nettside.

API: Akronym for **A**pplication **P**rogramming **I**nterface. API kan ha flerfoldige betydninger. I forbindelse med web er det ofte en henvisning til HTTP grensesnittet som knytter front-enden til back-enden, men det kan også henvise til ethvert grensesnitt som knytter én kodebase til en annen.

MVC: Akronym for **M**odel **V**iew **C**ontroller. Det er et programvareutviklingsmønster som går ut på at man skal fordele koden sin på en sånn måte at modeller, visninger, og kontrollere er separerte fra hverandre. En modell representerer f.eks. en bruker eller et utlån i et bibliotek. Modellene persisteres til databasen. En visning representerer et utsnitt av det grafiske grensesnittet. En kontroller inneholder businesslogikk, og fungerer som et knutepunkt mellom modellen og visningen.

JWT: Akronym for **J**SON **W**eb **T**oken. Det er en standard for å representere "påstander" (*eng.* claims) mellom to parter. I konteksten til vår webapplikasjon brukes JWT til autentisering av brukere. Det er en populær autentiseringsmetode for SPA-er.

CRUD: Akronym for **C**reate, **R**ead, **U**pdate, **D**elete. Flittig brukt begrep i databasesammenheng, da man gjerne gjør disse operasjonene på en database.

JSON: Akronym for **J**ava**S**cript **O**bject **N**otation. Et format for datautveksling som er basert på objektdefinisjonssyntaksen i JavaScript. JSON brukes mye i webapplikasjoner i forbindelse med datautveksling mellom front-enden og back-enden.

Rendering: Generering av innhold som skal presenteres. I websammenheng bruker man denne terminologien for å henvise til prosessen der HTML genereres for å bli presentert i nettleseren.

Isomorfisk webapp: En isomorfisk webapplikasjon er en webapp der front-endkoden kjører på både serveren og i nettleseren. Ulempen med en tradisjonell SPA er at det tar lang tid for siden å laste inn, i og med at all renderingen foregår i nettleseren. Med et isomorfisk oppsett kan serveren forhåndsgenerere mesteparten av nettsiden før den sendes til nettleseren. Dette betyr at klientsiden ikke lenger må generere hele siden alene, som medfører at siden vil lastes inn mye raskere.

Kodebase: Kodebasen (*eng.* code base) henviser til all koden i prosjektet i sin helhet.

Lo-fi: Lo-fi står for "low fidelity", og henviser til en prototype som er enkel og grovt utformet. Den består gjerne av enkle tegninger som i grove trekk skildrer oppbygningen til applikasjonen. Siden den er så enkel tar det ikke lang tid å utforme en lo-fi prototype.

Hi-fi: Hi-fi står for "high fidelity", og henviser til en prototype som gjerne er digital og grundig gjennomført. Av den grunn tar det også mye lenger tid å utforme en hi-fi prototype enn en lo-fi prototype. Webapplikasjonen vi bygde i forbindelse med dette bachelorprosjektet er et eksempel på en hi-fi prototype.

Webapplikasjon: En webapplikasjon (eller web app) er en nettside bygget for å fungere som en applikasjon (f.eks. en mobil app eller et program på PC-en.) I denne rapporten brukes begrepet "webapplikasjon" ofte for å henvise til hi-fi-prototypen vi bygde (implementasjonen av lo-fi-prototypen.)

WCAG: *Web Content Accessibility Guidelines* er et sett med regler som jobber mot et mål om å trumfe gjennom en delt standard for utforming av webinnhold som møter kravene til individuelle personer, organisasjoner og regjeringer på et internasjonalt nivå. [1]

ARIA: *Accessible Rich Internet Application* definerer en måte å gjøre webinnhold mer tilgjengelig for folk med funksjonshemming. Det definerer et sett med attributter som forklarer for skjermlesere eller lignende tjenester hva forskjellige HTML komponenter representerer.

Rammeverk: Et rammeverk i en programvareutviklingssammenheng er en abstraksjon som tilbyr generisk funksjonalitet som kan endres selektivt ved å skrive egen kode som samhandler med rammeverkets grensesnitt.

Front-end: Den delen av webapplikasjonen som kjører i nettleseren til brukeren. I konteksten til dette prosjektet består front-enden av en SPA utviklet ved hjelp av rammeverket Angular 2, programmeringsspråket TypeScript, stilarkspråket Sass, og markeringsspråket HTML. TypeScript transpileres til JavaScript, og Sass transpileres til CSS.

Back-end: Den delen av webapplikasjonen som kjører på serveren. I konteksten til dette prosjektet består backend-en av en kodebase skrevet i C#, der .NET Core er rammeverket og runtimen. Back-enden fungerer som et API for front-enden.

Transpilering: Transpilering vil si å oversette f.eks. et programmeringsspråk til et annet programmeringsspråk. I websammenheng gjør man ofte dette for å komme seg rundt begrensninger i de etablerte webteknologiene. JavaScript og CSS f.eks. har begrensninger og problemer som kan ha en negativ effekt på utviklingen av en applikasjon, men i og med at alle nettlesere støtter dem og ikke noe annet har man ikke noe annet valg enn å kjøre kode skrevet i disse språkene i nettleseren. Det er dog én måte å komme seg rundt dette på, og det er å transpilere et annet språk - f.eks. TypeScript til JavaScript, eller Sass til CSS - slik at man kan skrive all koden sin i et bedre språk som til slutt blir til f.eks. JavaScript som kan kjøres i nettleseren.

Det er også mulig å transpilere en nyere versjon av et programmeringsspråk til en eldre versjon (dette er vanlig praksis i forbindelse med front-endutvikling, siden man ikke kan forvente at brukeren kjører den nyeste versjonen av nettleseren sin.)

Versjonskontroll: Versjonskontroll (*eng.* version control) er et system som holder styr på versjoneringen av kodebasen. Det gjør det mulig å få en full oversikt over hvordan prosjektet har endret seg over tid, og det gjør det lettere for flere utviklere å jobbe på samme prosjekt samtidig. Vi tok i bruk versjoneringssystemet git i forbindelse med dette prosjektet.

1.7 Eksisterende løsninger

Før vi kunne begynne med å utvikle webapplikasjonen måtte vi finne ut av hva applikasjonen vår skulle erstatte. Det viste seg at det ikke fantes et eksisterende IT-system for problemstillingen. Hver gang en ansatt skulle låne et gitt utstyr måtte han notere det i en bok. Hvis en annen ansatt som kommer litt senere ikke finner det utstyret kan han slå opp i boken for å se hvem som lånte det. Problemet med dette systemet er at den boken befinner seg ved utstyret, så man kan ikke sjekke utlånsstatusen med mindre man befinner seg der boken er. Et annet problem er at folk iblant glemmer å notere at de har lånt et utstyr i boken.

1.8 Lignende løsninger

Webapplikasjonen vi lagde er ikke akkurat banebrytende med tanke på funksjonalitet. Ethvert bibliotek har nok et lignende system. Saken er at systemet vi har laget er skreddersydd og ment for et veldig spesifikt formål, og det er et system som kommer til å brukes internt i en bedrift. Vi tror at webapplikasjoner som ligner på det vi har laget mest sannsynlig også vil være skreddersydde løsninger som brukes av bedrifter internt, fremfor å være offentlige tjenester man kan finne med et enkelt Google-søk. Vi har forsøkt å lete etter lignende systemer, men vi klarte altså ikke å finne noe som var noe særlig relevant.

2 Metoder

Metodene vi brukte i forbindelse med gjennomførelsen av prosjektet.

2.1 Prosjektstyringsverktøy

Daglig Scrum

Vi møttes daglig og oppdaterte hverandre på hvor langt vi hadde kommet på de oppgavene vi hadde gitt oss selv. Det kom helt naturlig for oss, da vi måtte vite hvor langt den andre hadde kommet for å se hvor en selv kunne fortsette.

Backlog

Før sprintene la vi opp en del oppgaver til hver user story. Disse ble lagt inn i backlogen vår og markert med hvilken sprint vi skulle plukke dem opp i. Alle kortene i denne kategorien skulle dekke alle kode-behovene våre og alt vi kodet gjennom sprintene skulle være basert på disse kortene.

TeamGantt

For å holde styr på hele scrum prosjektet brukte vi TeamGantt. Der var alle sprintene delt inn i prototyping, brukertesting og evaluering med gitte tidsfrister for hvert punkt. Slik kunne vi forholde oss til en tidsramme og fikk en god flyt i prosjektet. TeamGantt fungerte da også som et loggføringsverktøy av arbeidstimer.

2.2 Samhandlingsverktøy

Facebook messenger

For generell diskusjon og kommunikasjon brukte vi facebooks messenger. Hadde det vært noe størrelse på gruppen hadde vi definitivt valgt slack eller noe lignende, men i og med at vi bare var to stykker konkluderte vi at Slack ville vært overflødig.

Skype

For videokonferansemøter brukte vi Skype. Vi brukte særlig Skype da vi skulle kommunisere med veilederen og arbeidsgiveren vår.

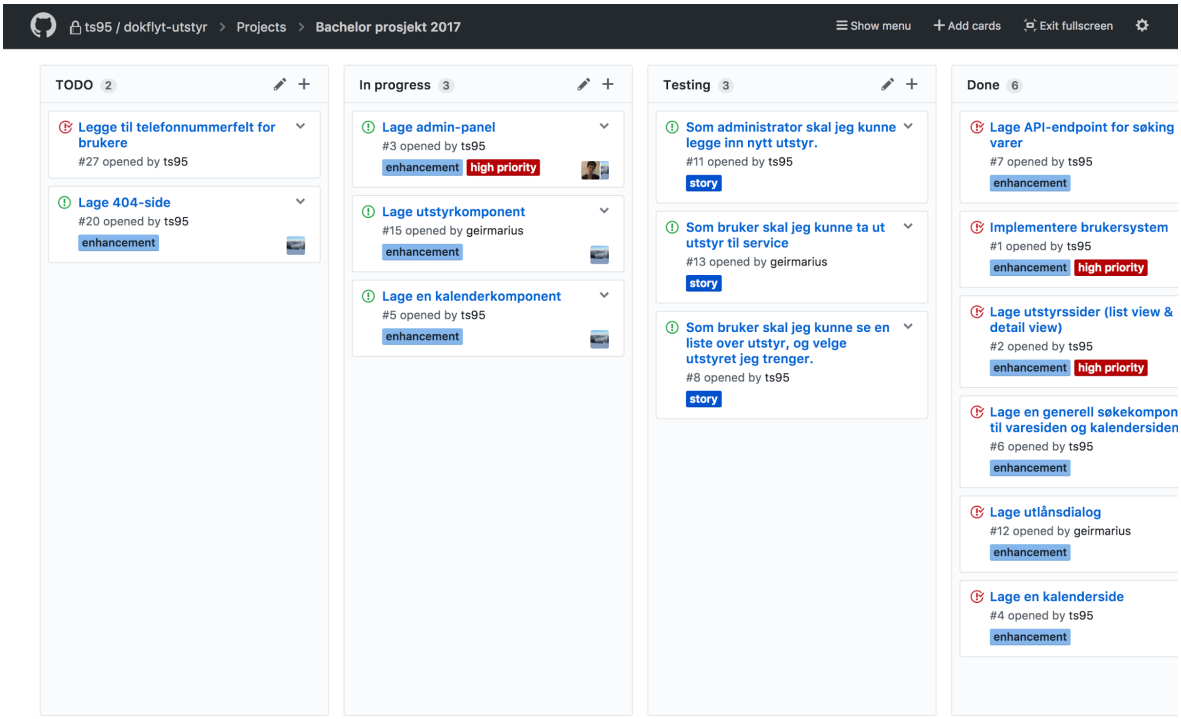
Slack

Slack tok vi i bruk for å kunne kommunisere med oppdragsgiveren vår, men ikke blant hverandre, i og med at vi allerede brukte facebook messenger til det formålet. Å bruke Slack

til å kommunisere med bedriften var veldig praktisk, siden vi kunne få svar raskt i og med at de ansatte i Dokflyt er på Slack hele tiden. Dette gjorde det enkelt å få svar på spørsmål, samt organisere møter.

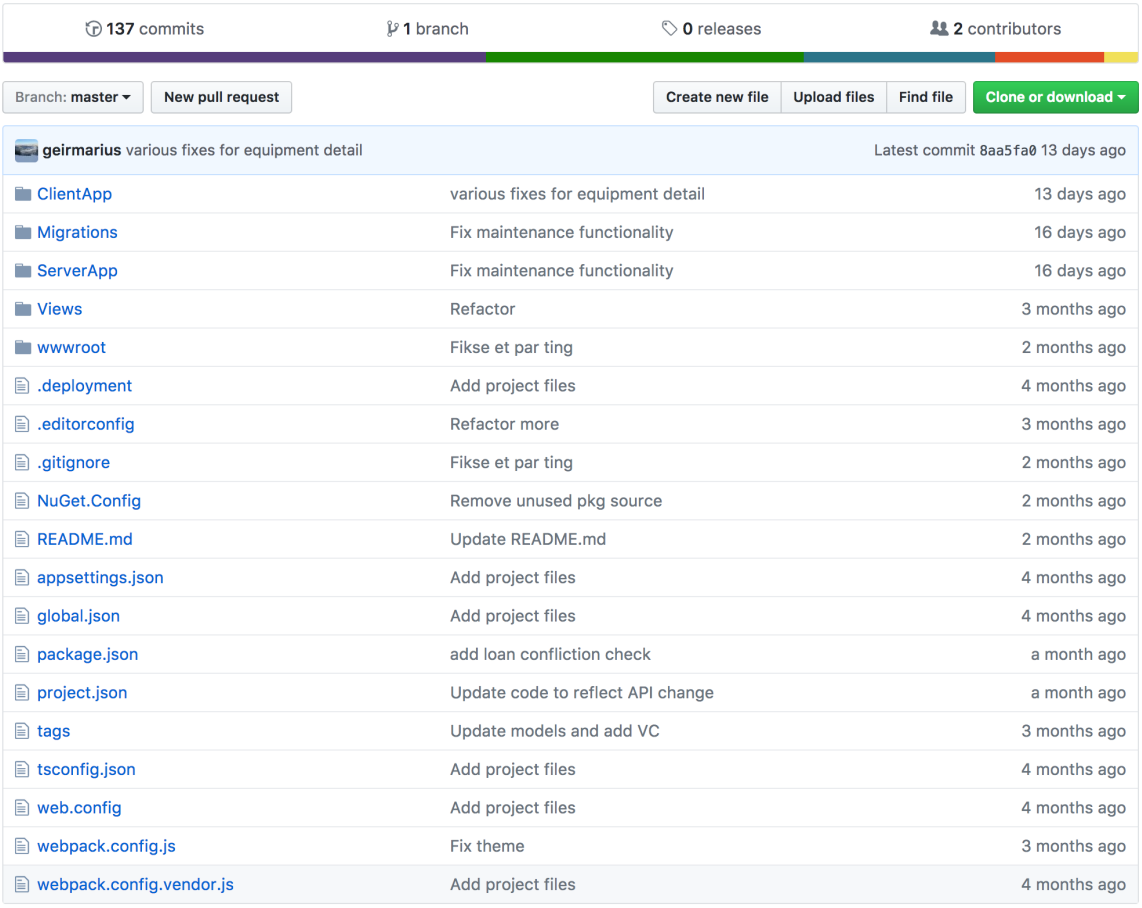
Github

Github spilte en sentral rolle i prosjektet vårt. I tillegg til å fungere som oppbevaringssted for kodebasen brukte vi også Github som et planleggingsverktøy for å organisere gjøremål.



Figur 1: Scrum-board på GitHub

Takket være et prosjektorganiseringsverktøy som er inkludert i Github slapp vi å bruke f.eks. Trello til å lage Scrum-boards. Vi brukte issue-systemet i Github til å informere hverandre om feil eller mangler underveis.



Figur 2: Prosjektet på GitHub

Google Drive

Vi brukte Google Drive til å dele filer med hverandre og veilederen vår. Det er et utmerket verktøy siden det også inkluderer verktøy som lar flere personer jobbe på et dokument samtidig, som var veldig praktisk da vi skulle skrive rapporten.

2.3 Verktøy og rammeverk

Når man skal utvikle en webapplikasjon finnes det mange måter å gå frem på. Dokflyt hadde et ønske om at vi skulle bruke verktøyene og rammeverkene nevnt nedenfor, så det var det vi gikk for. Noen av disse teknologiene var nye for oss, så vi måtte sette av litt tid til å lære hvordan de fungerer før vi kunne komme skikkelig i gang.

2.3.1 Programmeringsspråk

C#

C# er et objektorientert programmeringsspråk egnet til forskjellige formål. Det ble utviklet av Microsoft for .NET-plattformen. Språket kan defineres som et multiparadigma-programmeringsspråk med et strengt typesystem som er imperativt, deklarativt, funksjonelt, generisk, objektorientert og komponentorientert. Syntaksmessig minner C# om C-aktige språk som C, C++ og Java og deler mange likheter med dem.

Da C# først ble lansert var språket kun tilgjengelig på Windows gjennom programvareutviklingsverktøyene til Microsoft. Dette endret seg da Microsoft introduserte .NET Core, som offisielt gjorde C# til et multiplatformspråk som kunne anvendes både på Windows, macOS, og Linux. Dette har en vesentlig betydning i websammenheng, da man ofte distribuerer webapplikasjoner på Linux-maskiner og mange utviklere kjører macOS.

TypeScript

TypeScript er et superset av JavaScript som legger til funksjonalitet som valgfri statisk typing, og klassebasert objektorientert programmering. Fordelen med å bruke TypeScript over JavaScript er at det strengere typesystemet oppfatter visse typer feil som ellers ville gått ubemerket hen i JavaScript, dessuten får man en rekke fordeler man ellers ikke ville fått i JavaScript, nemlig at teksteditoren din kan gjøre en statisk analyse av koden din og oppdage typefeil dynamisk mens du skriver koden, samt komme med nyttig typeinformasjon.

TypeScript kan ikke kjøres i nettleseren uten videre, og må dermed transpileres til JavaScript. Dette kan man gjøre manuelt via kommandolinjen, men i de fleste utviklingsoppsett håndteres denne transpileringen av bundleren (mer om den nedenfor) - som er webpack i vårt tilfelle.

En annen ting som er verdt å nevne er at Angular 2 rammeverket er skrevet i TypeScript.

2.3.2 Markeringsspråk og stilarkspråk

HTML

HTML, akronym for Hyper Text Markup Language, er et markeringsspråk som brukes i forbindelse med å lage websider og webapplikasjoner. Nettlesere mottar HTML-dokumenter fra en webserver eller et lokalt lagringsmedium, og presenterer dem i form av multimedia websider. HTML beskriver strukturen til en webside semantisk.

CSS & Sass

CSS, akronym for Cascading Style Sheet, er et stilarkspråk som brukes til å definere utseende til HTML-dokumentet med tanke på posisjonering av objekter, fonter, farger, og padding/marginer. Sass er et annet stilarkspråk som ble utviklet for å løse visse problemer som oppstår i større prosjekter når man bruker CSS. Det er i utgangspunket det samme som CSS, men med utvidet funksjonalitet som gjør det mer leselig og lettere å jobbe med i store prosjekter. Sass, på lik linje med TypeScript, må transpileres til CSS.

2.3.3 Rammeverk

ASP.NET Core MVC

ASP.NET Core MVC er et MVC-rammeverk for bygging av dynamiske websider. Det er en videreføring av ASP.NET MVC som er blitt redesignet for å passe den moderne weben bedre. ASP.NET Core MVC er altså like godt egnet til å bygge et REST-API som det er til å bygge en fullverdig nettside. I vårt prosjekt brukte vi rammeverket kun til å lage et API, i og med at all renderingen foregikk på klientsiden (front-enden).

Angular 2

Angular 2 er et front-endrammeverk skrevet i TypeScript og utviklet av Google for komplekse webapplikasjoner.

Mot slutten at utviklingsfasen ble den neste versjonen av Angular, nemlig Angular 4, lansert (de hoppet over versjon 3.) I og med at vi ikke hadde mer tid igjen rakk vi ikke å oppdatere kodebasen til den nye versjonen.

2.3.4 Verktøy

Visual Studio Code

Visual Studio Code er en kildekodeeditor utviklet av Microsoft for Windows, Linux, og macOS. Den inkluderer støtte for feilsøking, Git-funksjonalitet, “syntax highlighting”,

“intelligent code completion”, og refaktorering av kode. Det er mulig å tilpasse den ved å endre temaet eller installere utvidelser gjennom utvidelsessystemet til editoren.

Vi var egentlig vant med andre editorer, men vi besluttet å bruke Visual Studio Code på grunn av den gode integrasjonen med C# og TypeScript.

npm
npm (node package manager) er pakkebehandleren til JavaScript-runtime-miljøet Node.js. npm kan behandle pakker som er lokale avhengigheter i et gitt prosjekt, samt behandle globalt installerte JavaScript-verktøy. Når det brukes som en avhengighetsbehandler for et lokalt prosjekt kan npm installere alle avhengighetene til et prosjekt gjennom en package.json med kun én kommando.

Vi brukte npm til å håndtere avhengighetene på front-enden.

teamgantt
teamgantt.com er en nettside der man kan lage gantt-diagrammer og føre inn arbeidstimer, samt legge inn kommentarer på de forskjellige målene man har lagt inn. Vi brukte den i forbindelse med sprintene og andre gjøremål.

2.3.5 Bundlere

Webpack
Webpack er en såkalt “module bundler”. Formålet til en module bundler er å pakke alle de forskjellige ressursene på front-enden sammen til en plassbesparende bunt. Webpack sørger for at all TypeScript koden blir transpilert til JavaScript, som deretter blir minifisert. Minifisering er en prosess der kode som skal kjøre på front-enden blir strippet for unødvendig tekst som tar opp plass. Dette inkluderer blant annet mellomrom, tomme linjer, lange variabelnavn, osv. I tillegg til å transpilere TypeScript til JavaScript transpilerer webpack Sass til CSS, og minifiserer CSS-en.

2.4 Arbeidsflyt

I begynnelsen av semesteret begynte vi med praksisperiodene våre i forbindelse med faget IMT3541 - Veiledet praksis i medie- og informatikkfag. Toni var utplassert i Shortcut AS i Oslo, mens Geir Marius var utplassert i Dokflyt AS i Gjøvik. Vi jobbet begge på torsdager og fredager, og brukte dermed mandagene, tirsdagene, og onsdagene til å jobbe med prosjektet.

Tonis utplassering medførte at han måtte pendle til og fra Oslo én gang i uken, som var en utfordring med tanke på kontinuiteten til prosjektarbeidet. Arbeidet var fordelt sånn at Toni var ansvarlig for back-enden, mens Geir Marius var ansvarlig for front-enden. Vi valgte denne fordelingen siden Toni var den eneste som kunne C# og hadde mer erfaring med back-endutvikling, mens Geir Marius hadde mer erfaring med front-endutvikling. Dette hadde sine fordeler og ulemper. Det var fordelaktig at vi jobbet på to separate deler av applikasjonen, siden det innebar at vi kunne jobbe samtidig og dermed bruke tiden mer effektivt, men et problem var at vi var veldig avhengig av hverandre med tanke på fremgang. Hvis én av oss var opptatt, stoppet utviklingen opp. Vi hadde heldigvis tre sammenhengende dager hver uke som vi kunne bruke til å jobbe sammen side om side, så vi fikk gjort ganske mye. Vi rakk dog ikke å gjøre ferdig alt innen de tre sprintene vi planla, så vi endte opp med å jobbe litt av og på med prosjektet etter sprintene for å gjøre oss ferdig med det.

Rollene var fordelt slik:

Gruppeleder - Toni Sucic

- Scrum master & scrum team
- Ansvar for backenddelen av prosjektet
- Ansvar for visse deler av implementasjonen til frontenden
- Ansvar for å sjekke tidsloggen ved ukeslutt og bekrefte at alle har jobbet nok

Utvikler - Geir Marius Kirkeberg Jansson

- Scrum team
- Ansvar for frontenddelen av prosjektet
- Ansvar for designprosessen
- Booker rom når nødvendig

Oppdragsgiver - Dokflyt

- Scrum product owner
- Ansvar for intervjuobjekter

2.5 Analyser

Før vi kunne sette i gang med prototypene måtte vi lære mer om hva vi skulle lage. Det første spørsmålet man gjerne stiller seg selv før man skal sette i gang med et prosjekt som dette er hvorvidt det finnes lignende løsninger allerede som man kan analysere for å se

hvordan de løste diverse problemer, og om oppdragsgiveren allerede har en løsning på plass, så det var dette vi fokuserte på i begynnelsen.

2.6 Konseptutvikling

2.5.1 Introduksjon

Webappen vår bygges på et helt nytt konsept fra bedriftens side, og bedriften har ingen tidligere løsninger som potensielt kunne gått utfra. Dermed måtte vi tilegne oss informasjon gjennom brukerinervjuer og analyser for å finne ut av hvor fokuset skulle ligge.

2.5.2 Brukerintervjuer

Etter å ha fått grunnleggende informasjon om løsningen vi skulle bygge ønsket vi å ha noen intervjuer med brukere som skulle bruke webappen i fremtiden. I den forbindelse ønsket vi å få avklart hvordan den nåværende løsningen fungerte og problemer de hadde med denne, samt hva som fungerte godt. Dette forenklet arbeidsprosessen, slik at vi kunne fokusere på de rette tingene.

2.5.3 Use-case diagram

Et use-case diagram er en oversikt over relasjonen mellom aktører og funksjoner. Vi kan f.eks. se hvilken funksjonalitet en admin har som en bruker ikke har. Mer avanserte use-case diagrammer viser også relasjoner til en database, men dette brukte ikke vi.

Etter å ha fått en forståelse for hva bedriften ønsket og hva brukerne mente satt vi opp et use-case diagram for å kunne formidle både internt og til bedriften hvordan vi mente løsningen skulle fungere. Det var viktig at all funksjonalitet ble kartlagt og at vi alle var enige om hvem som skulle kunne styre hva før vi begynte å kode løsningen, slik at vi ikke måtte gjøre store og tidkrevende endringer underveis.

2.5.4 Personas og scenarios

Personas er fiktive figurer som representerer målgruppa. Ved bruk av personas kan vi teste løsningen vår opp mot målgruppa under hele utviklingsprosessen uten å måtte konstant være i kontakt med brukere. Dette gjør utvikling mye smidigere.

Scenarios er virkelighetsnære situasjoner hvor webappen vår er aktuell. Det gjør at vi kan se løsningen vår i bruk uten å hente inn folk til brukertesting.

Noen fallemmer ved bruk av personas og scenarios er å helt stenge ute tradisjonell brukertesting. Da bygger du plutselig applikasjonen din kun for personasene, og du får et blindt øye til det store spekteret av folk som faktisk skal bruke webappen. Derfor er det viktig å både beholde tradisjonell brukertesting, og å lage solide personas og scenarios som dekker et bredt spektrum.

2.7 Organisering

For å garantere produktivitet i periodene vi ikke var på jobb i forbindelse med utplasseringen definerte vi noen rutiner og regler som vi skulle følge helt i begynnelsen av prosjektet.

Rutiner:

- Regelmessige fysiske oppmøter i f.eks. grupperom, biblioteket eller andre steder der man kan arbeide. Minimum 3 dager i uka, gjerne mer hvis man vil.
- Logging av arbeidstid. Når man jobber med en oppgave som er definert i Gantt-diagrammet må man passe på å notere hvor mange timer man har jobbet.

Regler:

- Som et minimum må hvert gruppemedlem legge inn 7 timer arbeid daglig minst 5 ganger i uka (4 ganger i uka hvis man fremdeles er utplassert i bedrift). Ved slutten av uka vil gruppelederen undersøke om timeloggen stemmer overens med antall timer man skal jobbe.
- Hvert gruppemedlem må møte opp på campus klokken 9 om morgenen senest og bli der til de 7 timene er unnagjort (dette må gjøres minimum 3 dager i uka). Man jobber altså fra kl 9.00 til kl 16.00.

2.8 Wireframes

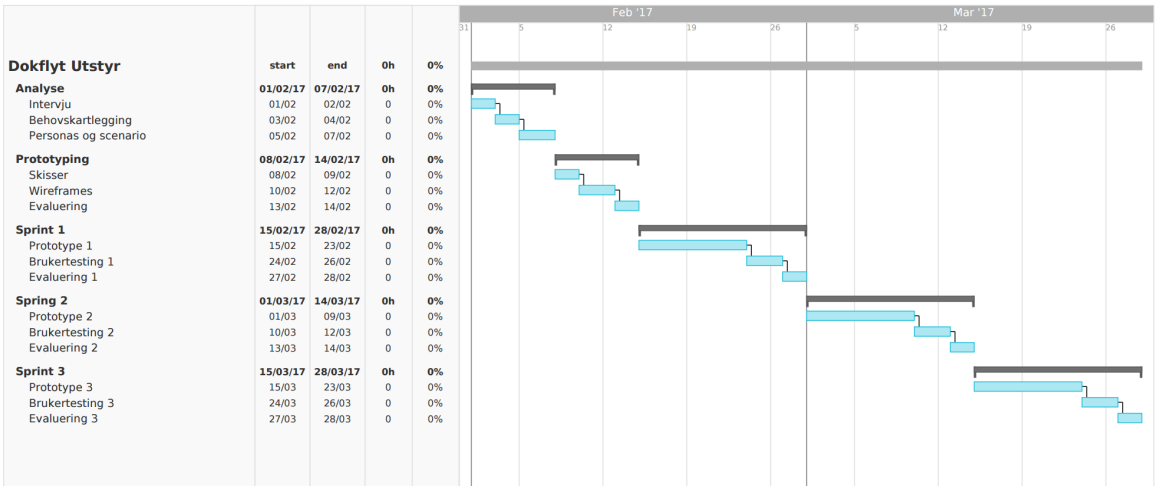
Wireframes er et tenkt oppsett til en applikasjon og lar oss planlegge strukteren i webappen vår før vi begynner med hvordan enkeltgjennstander skal se ut. Vi kan kjapt gjøre endringer i wireframes uten at det skal ha noe sideeffekt eller at vi må gjøre mye arbeid på nytt. Når wireframes er på plass kan vi begynne og tenke på mer detaljert design og starte kodeprossesen, da vi vet hvordan strukturen skal bli.

2.9 Prototype

Prototyper er til for å lage en enkel, provisorisk versjon av applikasjonen. Fordelen med en prototype er at den ikke behøver å være perfekt, og man lærer av utfordringene man støter på underveis i prosessen der man utvikler prototypen. Denne erfaringen kan man ta i bruk senere hvis man skal lage en ny prototype eller en endelig versjon av applikasjonen. Man skiller mellom lo-fi og hi-fi prototyper, og man kan velge om man vil fokusere på bredden i prototypen (horisontal prototype) eller dybden (vertikal prototype).

2.10 Plan for gjennomføring

2.10.1 Gantt-skjema



Figur 3: Gantt-skjema generert av teamgantt.com

Vi lagde oss et gantt-skjema slik at vi kunne få oversikt over fasene til prosjektet, og hvor i tidsforløpet vi befant oss. Hver fase er illustrert med en horisontal søyle som strekker seg mellom startdatoen og sluttdatoen til en fase. En søyle består av oppgaver, hvorav noen avhenger av at andre oppgaver allerede er blitt gjort (man kan f.eks. ikke begynne å lage wireframes uten å la laget ferdig skisser først). Disse avhengighetene er illustrerte med sorte streker som strekker seg fra enden til én søyle til begynnelsen av en annen.

2.10.2 Milepæler

Gjennom prosjektet skulle vi gjennom flere milepæler med et klart mål.

1: Ferdigstille behovskrav

- Blir ferdig med intervju for behovsplanlegging

- Kartlegge alle behov
- Skrive ferdige alle personas med scenarioer

2: Komplett design

- Legge opp alle skisser
- Komme frem til et sett med wireframes vi skal bygge på
- Tegne opp prototypen vi skal gjøre interaktiv

3: Klart produkt

- Fullføre alle sprint sykluser
- “Finpusse” koden

4: Ferdig rapport

- Skrive ferdig rapport
- Se over etter feil og svake deler
- Sette opp referanser

2.10.3 Tids- og ressursplan

Aktivitet	**Tid**	**Kommentar**
Intervju	3t	Intervjue brukere for å lære mer om behovene deres.
Behovskartlegging	7t	Bruke opplysningene fra intervjuene til å kartlegge behovene til brukerne.
Personas og scenarier	7t	Lage personas og scenarier som gjenspeiler brukerne og behovene deres.
Skisser	5t	Tegne skisser av applikasjonen.
Wireframes	14t	Lage wireframes basert på skissene.
Prototype 1	50t	Utvikle den første prototypen.
Brukertesting 1	5t	Teste prototypen på brukerne.
Prototype 2	50t	Utvikle den andre prototypen.

Brukertesting 2	5t	Teste prototypen på brukerne.
Prototype 3	50t	Utvikle den tredje og siste prototypen.
Brukertesting 3	5t	Gjøre den siste brukertestingen.
SUM	201t	Alle timene sammenlagt.

2.12 Objektorientert programmering

Objektorientert programmering (ofte forkortet til OOP) er et paradigma innen programvareutvikling der objekter står sentralt i måten man utformer koden på. Et objekt i en OOP-sammenheng er en representasjon av et objekt som f.eks. kan opptre i virkeligheten. En bok i et bibliotek kan representeres som et objekt. Bøker har visse attributter som definerer dem: tittel, forfatter(e), utgivelsesdato, antall sider, synopsis, ISBN. Man kan lage et objekt som heter “Bok” og definere de tidligere nevnte attributtene på det objektet. Et objekt kan også ha metoder. Metoder er funksjoner / prosedyrer med kode som opererer på objektet. En metode kan modifisere objektets attributter, eller samhandle med et annet objekt. Hvis vi har et “Person”-objekt kan vi f.eks. definere en metode på det objektet som heter “inspiserBok”. Denne metoden kan da ta imot et “Bok”-objekt og avlese dens attributter.

I vårt prosjekt brukte vi to OOP-språk: C# og TypeScript. Objekter var et særlig nyttig konsept i forbindelse med å definere modeller, og modellutveksling mellom front-enden og back-enden. Vi kunne bruke JSON-formatet til å serialisere et C#-objekt på back-enden som kunne sendes over HTTP og konverteres til et TypeScript-objekt på front-enden.

2.13 Dependency injection

“Dependency injection” (avhengighetsinjeksjon) er en programvareutviklingsteknikk som går ut på at et objekt forsyner avhengighetene til et annet objekt. Tradisjonelt sett er ethvert objekt selv ansvarlig for å få tak i referanser til objekter det samarbeider med (dets avhengigheter). Ved å ta i bruk dependency injection får objektene sine avhengigheter når de instansieres av en ekstern entitet som koordinerer hvert objekt i systemet.

Dette var et nytt konsept for oss som vi måtte lære oss i og med at både Angular 2 og ASP.NET Core MVC brukte dependency injection ekstensivt.

3. Resultat

3.1 Design

3.1.1 Målgruppe

Målgruppen til designet vårt er todelt. På én side er det montørene som skal bruke applikasjonen, og på en annen side er det saksbehandlerne.

3.1.2 Brukerintervju

3.1.2.1 Fremgangsmåte

Før vi startet med planleggingen av prosjektet ønsket vi å komme i kontakt med brukere for å undersøke det de brukte og deres behov.

Vi hadde først et brukerintervju med arbeidsgiver, som tidligere var i posisjonen som bruker. Fra ham hentet vi ut mye nyttig informasjon og endret litt på spørsmålene våre til neste intervju.

Vårt andre intervju var med en bruker av systemet og, litt overraskende, fortalte han mer eller mindre helt det samme som arbeidsgiver. Dette ble en bekreftelse for oss på at vi hadde fått de svarene vi trengte.

Senere i prosessen hadde vi intervjuer etter hver sprint - i god scrum ånd. Det ga oss en indikator på hva som var utviklet i riktig retning og hva som ikke funket.

3.1.2.2 Utbytte

De første to intervjuene ga oss svar på en rekke spørsmål vi trengte for å kunne starte. Vi spurte om målgruppen vår, hvordan den eksisterende løsningen fungerte i detalj, hvor ofte de brukte den eksisterende løsningen, og hva de tenkte at en ny elektronisk løsning ville forbedre.

Målgruppen var spredt, men hadde en tung vekt på eldre personer som ikke var noe særlig datakyndige. De fleste hadde telefoner og tilgang til nettbrett, mens svært få brukte PC. Målgruppa er også noen som har brukt “boka” i alle år, og for å erstatte den må det være en overgang de som holder igjen må være med på.

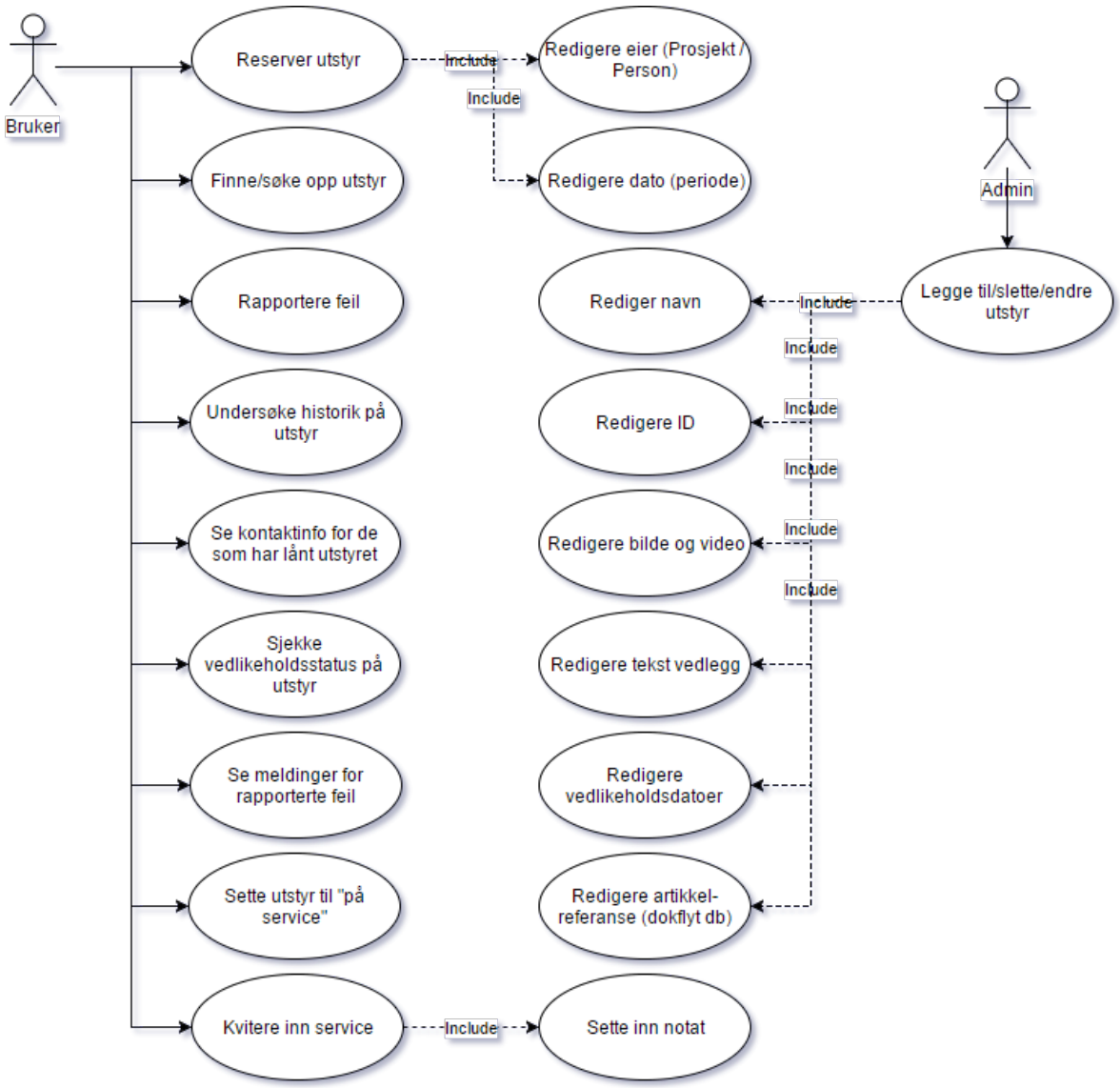
Eksisterende løsninger kunne variere fra excel til en notatbok, men notatbok var definitivt mest brukt. Der skrev de ned når de skulle bruke noe og til hvilken dato. Noen varer er litt utenfor boka og er ikke så lett å se om er utlånt eller ikke. I boka er det viktig å vite hvem som har varen så det går an å få kontakt og ikke *hvor* de er. Av det kom det frem at GPS tracking eller lignende løsninger - som arbeidsgiver først ønsket - ikke var noe behov for brukerne.

Utstyr ble lånt ut daglig og enkelte gjenstander som lift var konstant utlånt. Slik at et behov for en fremtidig løsning må gjøre det superenkelt og effektivt å låne, ellers ville det ikke blitt brukt som det skulle.

3.1.3 Kartlegging av behov

Figur 4: Første utkast av use-case diagrammet

Slik vi trodde løsningen skulle se ut (se fig 4) var et ganske mer komplisert bilde enn det arbeidsgiver hadde i tankene. Etter vi hadde delt diagrammet fikk vi det i retur med røde streker og piler overalt.



Figur 5: Siste versjon av use-case diagrammet

Det endte opp med en veldig forenklet utgave av det vi hadde sett for oss (se figur 5) som trolig hjalp på å korte ned tiden vi brukte på utvikling av funksjonalitet. Legg også merke til at mange av use-casene er borte. Det skyldes blant annet at brukerinformasjon håndteres av et eksternt system og det er ikke behov for brukeradministrasjon i vår løsning.

3.1.4 Universell utforming

Det var viktig for oss at universell utforming kom som en naturlig del av designdelen, slik at vi ikke brukte lang tid på å justere oss etter WCAG 2.0 sine spesifikasjoner.

For wireframes sørget vi for at alt som var klikkbart var stort nok og at alt som skulle klikkes lignet på noe klikkbart.

Når vi implemnterte løsningen var det viktig at vi alltid brukte alt-tekst på bildene slik at skjermlesere etc. kan lese det opp. Det var da en beskrivelse av bilde om det sto alene eller hvor bilde linket til om det var i kombinasjon med en link.

Videre sørget vi for at alt av innhold fulgte standard HTML komponenter som hadde innebygd denne standarden og hvor det var lite vi måtte gjøre. I de få tilfellene vi ikke kunne bruke disse, benyttet vi oss av ARIA-attributter og ofte tabindex-attributtet for å legge til rette for tastatur-navigering.

For kontraster fulgte vi det temaet vi jobbet med, men ble noe for fælt ville vi ha endret på fargene.

3.1.5 Personas og scenarios

3.1.5.1 Persona #1

Persona	Bruker av applikasjonen. Yrke: Energimontør
Bilde	
Fiktivt navn	Egil Kristiansen
Demografi	Mann, 40 år gammel, gift, far til tre, bor i Hønefoss
Mål og oppgaver	Egil jobber som energimontør i bedriften Sterkstrøm AS. Hans arbeidsoppgaver består av: ● å vedlikeholde, montere og reparere det elektriske

	nettet, kabler og strømforsyningsanlegg. ● å utføre feilsøking av elektriske anlegg og utstyr ● reise stolper og legge kabler
Miljø	Egil er middels datakyndig. Han har brukt diverse nettsider og nettbaserte applikasjoner tidligere, og har ikke problemer med disse.

3.1.5.2 Persona #2

Persona	
Bilde	
Fiktivt navn	Rune Høgberg
Demografi	52 år, gift, far til en utflyttet sønn
Mål og oppgaver	Rune jobber sine siste arbeidsår i felt som bygningsarbeider. Han har store ansvarsoppgaver som en person med dyp erfaring. Rune er ofte med og lærer opp nykommere til arbeidsplassen. Viser dem hvordan ting gjøres i bedriften og henviser til standarder og retningslinjer. Ellers begynner det å bli lite fysisk tungt arbeid på rune.

Miljø	Rune synes ny teknologi er spennende og leser seg opp på hva som finnes i og litt utenfor hans fagområdet. Når det kommer til å finne disse artiklene syntes rune det ofte er greit med en laptop eller nettbrett. Helst ser han at det er et skikkelig tastatur. Mobil blir alt for lite. Rune bruker kun laptop / nettbrett til å lese artikler og mail.

3.1.5.3 Scenario #1

Rune skal vise noen av de nye gutta hvordan liften deres er, og skal la dem teste den før de begynner å bruke den i arbeid. Rune ringer da til en som er i nærheten av boka og sjekker om den er utlånt. Telefonen blir ubesvart og Rune må selv tusle bort med gutta til boka. Den har ikke vært lånt ut på en uke og skal være inne. Rune skriver seg opp på liften for hele dagen og tusler bort til der liften skal stå.

Når rune kommer frem til lagre der liften skal stå er det ingen lift i siktet. Rune tenker så at noen har satt den fra seg et annet sted og springer rundt å leter etter den med guttene på slep. Etter en halvtimes tid har de fortsatt ikke funnet den.

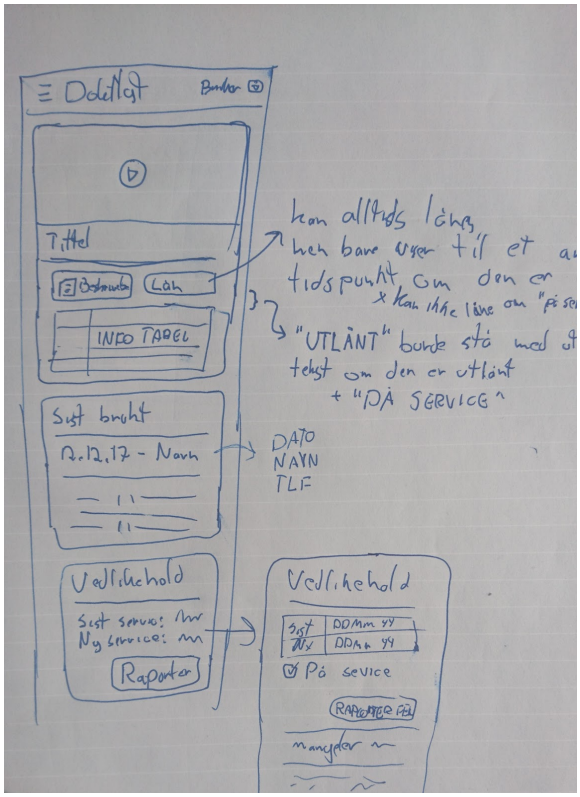
Det viser seg at den sto rett på utsiden av arbeidsplassen og at noen hadde lånt den rett før Rune ankom for å henge opp bedriftens navneplakat på brakkene. Det skulle bare ta 30 minutter og personen som lånte den ut så ikke poenget med å gå helt til der boka var for å registrere 30 min.

3.1.5.4 Scenario #2

Daniel er elektriker og skal installere et skjult anlegg i et hus. For å gjøre dette avhenger han av diverse utstyr som han låner fra bedriften. Han er i bilen sin og vet ikke om utstyret han trenger er lånt ut eller ikke, så han drar frem mobilen og går inn på Dokflyt Utstyr for å sjekke om utstyret hans er blitt utlånt allerede eller ikke. Det viser seg at noen har lånt utstyret han trenger, men at det kommer til å bli ledig igjen om kort tid, så han legger inn et utlån av det utstyret slik at ingen andre skal ta det før ham.

3.1.6 Wireframes

Før valg av endelig layout utviklet vi to forslag - en fra hver av oss. Med to layouts tok vi de beste idéene fra begge og fusjonerte dem til et. Det viktigste under valget av wireframes var hva som lot oss låne ut varer kjappest og med minst mulig rom for forvillelse.



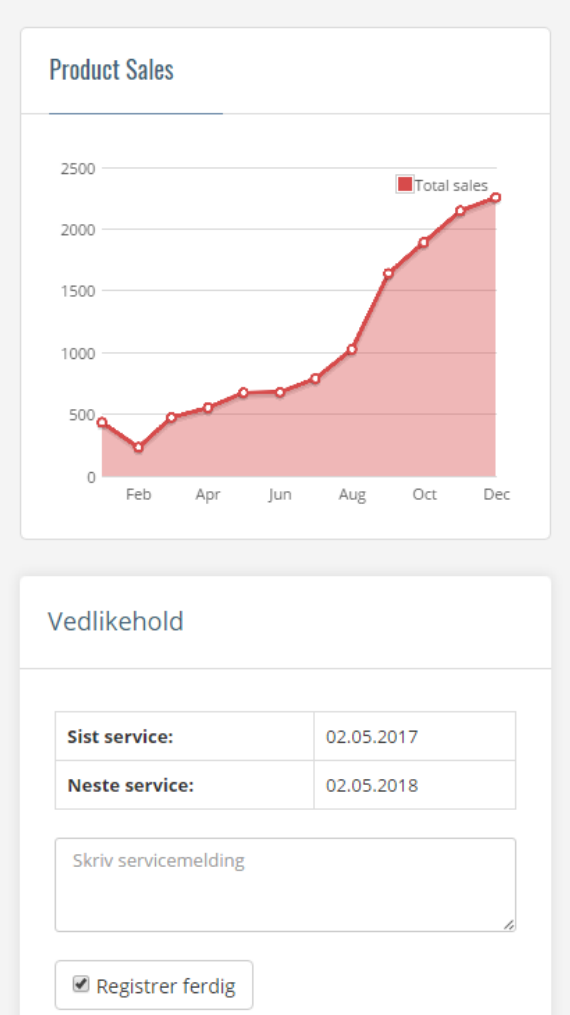
Figur 6: Detaljesiden til et utstyr

Vi endte opp med en snarvei til å ta ut utstyr på nesten alle sider av applikasjonen og selve utlånssiden skulle ha ferdig utfylt informasjon og kreve minimalt med justeringer for å låne en vare.

Andre krav til wireframes var at de oppfulgte alle spesifikasjonene vi hadde i use-case diagrammet vårt.

3.1.7 Retningslinjer

Vi bygde hele applikasjonene på et eksisterende bootstrap-tema. Det var dette temaet som Dokflyt brukte fra før, og ønsket at vi videreutviklet. Dermed hadde vi allerede en del retningslinjer vi måtte følge.



Figur 7: Temaets kort på toppen og våre justeringer på bunn

Kort

Alt av informasjon er gruppert i kort, eller "portlets", som temaet kalte det.

Vi fjernet den skarpe rammen rundt kortene og la på en myk skygge for å senke "strengheten" ved webappen.

Fargebruk

Hovedsakelig er temaet ganske fargeløst med unntak av navigasjonmenyen på toppen som har en mørk blåfarge. Vi endret mange av de andre stedene blåfargen var brukt (Se blant annet figur 6) da vi ønsket at disse ikke skulle trekke så mye oppmerksomhet og skulle gli bedre inn i resten av webappen.

Temaet kommer med en primærfarge som er så godt som rød. Det kan by på problemer da man ofte assosierer fargen med fare. Vi endret ikke på denne, men brukte heller ikke rødfargen noen andre steder. Det viktigste var å fort vise at denne fargen var kun ment for oppmerksomhet, og ikke at noe var farlig. Det oppnådde vi ved at første sted du ser den er det det eneste valget du har for å fortsette. Ellers i applikasjonen er det ingen steder vi ønsket å vise at dette er et "farlig" valg.

Ikoner

Vi brukte en rekke Font Awesome ikoner på front-enden. De står oppført i tabellen nedenfor med Font Awesome class-navnet og i hvilken sammenheng vi anvendte dem.

Ikon	Navn	Bruksområde
	arrow-right	Brukes i “Lån ut” knappen for å indikere at knappen tar deg til siden der man kan låne ut et utstyr.
	remove	Brukes i lukkeknappen i søkedialogen.
	filter	Brukes til å veksle mellom forskjellige filtere i søkedialogen.
	icon-bar x3	Brukes i navbaren for mobile enheter. Den brukes til å åpne og lukke navigasjonsbarens panel.
	arrow-circle-o-down	Brukes på utstyrssiden for PDF-dokumentet man kan laste ned (hvis saksbehandleren lastet opp et PDF-dokument med utstyret)
	chevron-left	Brukes til å gå én uke tilbake i tid i kalenderkomponenten.
	chevron-right	Gjør det motsatte av chevron-left, den går altså én uke frem i tid.

3.2 Prototype

Dette kapittelet tar for seg prototypene vi utviklet, hvilke verktøy vi brukte for å utvikle dem, og utviklingsprosessen.

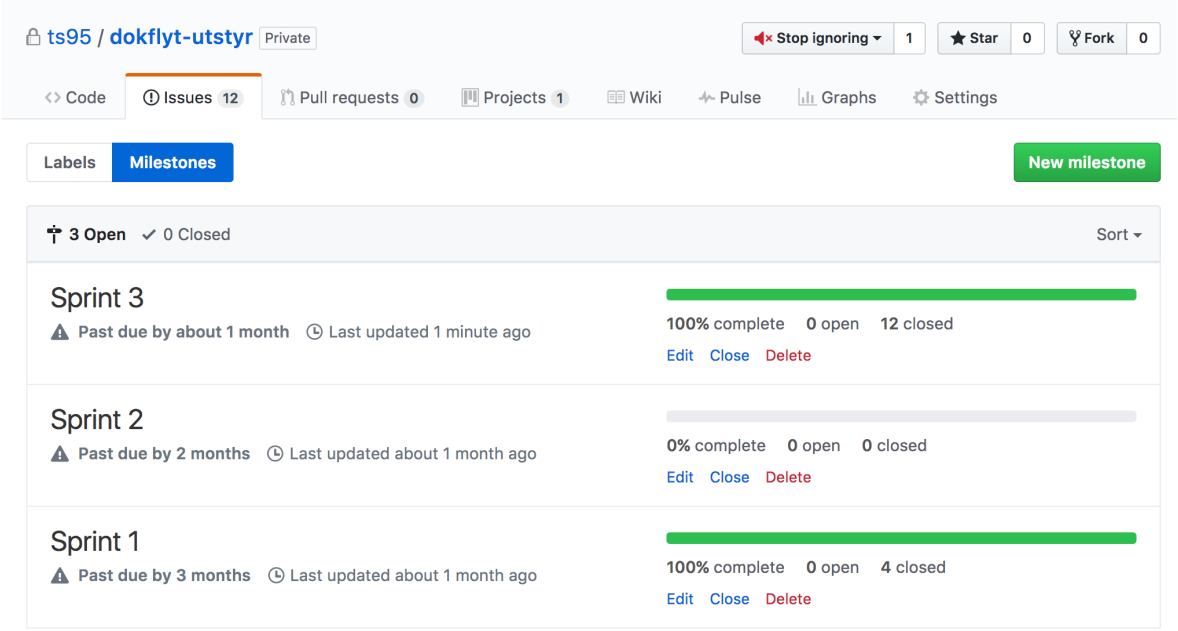
3.2.1 Introduksjon

Vi lagde to prototyper. Den første var en enkel lo-fi prototype bestående av skisser og wireframes. Vi lagde deretter en hi-fi prototype med utgangspunkt i lo-fi prototypen. Vi begynte med å gjøre hi-fi prototypen horisontal slik at vi hadde et utgangspunkt som kunne presenteres til oppdragsgiveren, slik at vi kunne få tilbakemeldinger. Deretter gikk vi mer inn i dybden med henhold til funksjonalitet og lagde en vertikal prototype.

3.2.3 Sprinter

Som en del av Gantt-skjemaet vi satt opp måtte vi gjennomføre tre sprinter. Hvert sprint gikk ut på at vi skulle gjøre ferdig en del av applikasjonen. Et sprint bestod altså av diverse gjøremål som vi måtte gjennomføre i løpet av det sprintet.

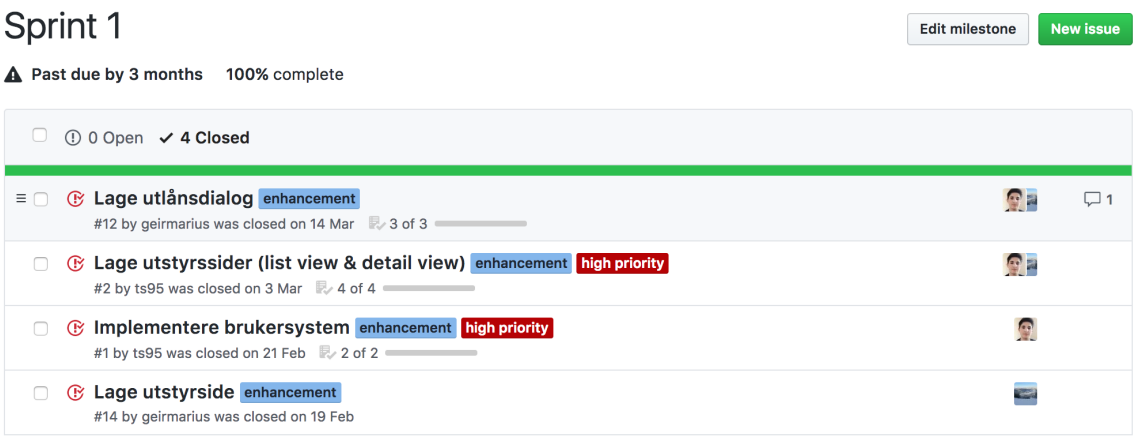
Hvert sprint var delt opp i tre deler: arbeid med hi-fi prototypen (webapplikasjonen), brukertesting, og til slutt evaluering. Vi satte av ni dager til prototypen, tre dager til brukertesting, og to dager til evaluering. Vi kunne dog ikke være produktive hver dag, i og med at vi begge var opptatt med utplassering to dager i uken.



Figur 8: Sprinter på GitHub.

Sprint 1

Det første sprintet gikk ut på å sette opp de grunnleggende delene av applikasjonen. Toni hadde allerede begynt med prosjektstrukturen litt før det første sprintet, slik at vi kunne bruke mer tid på å implementere funksjonalitet. Å lage utstyrssider stod i fokus.



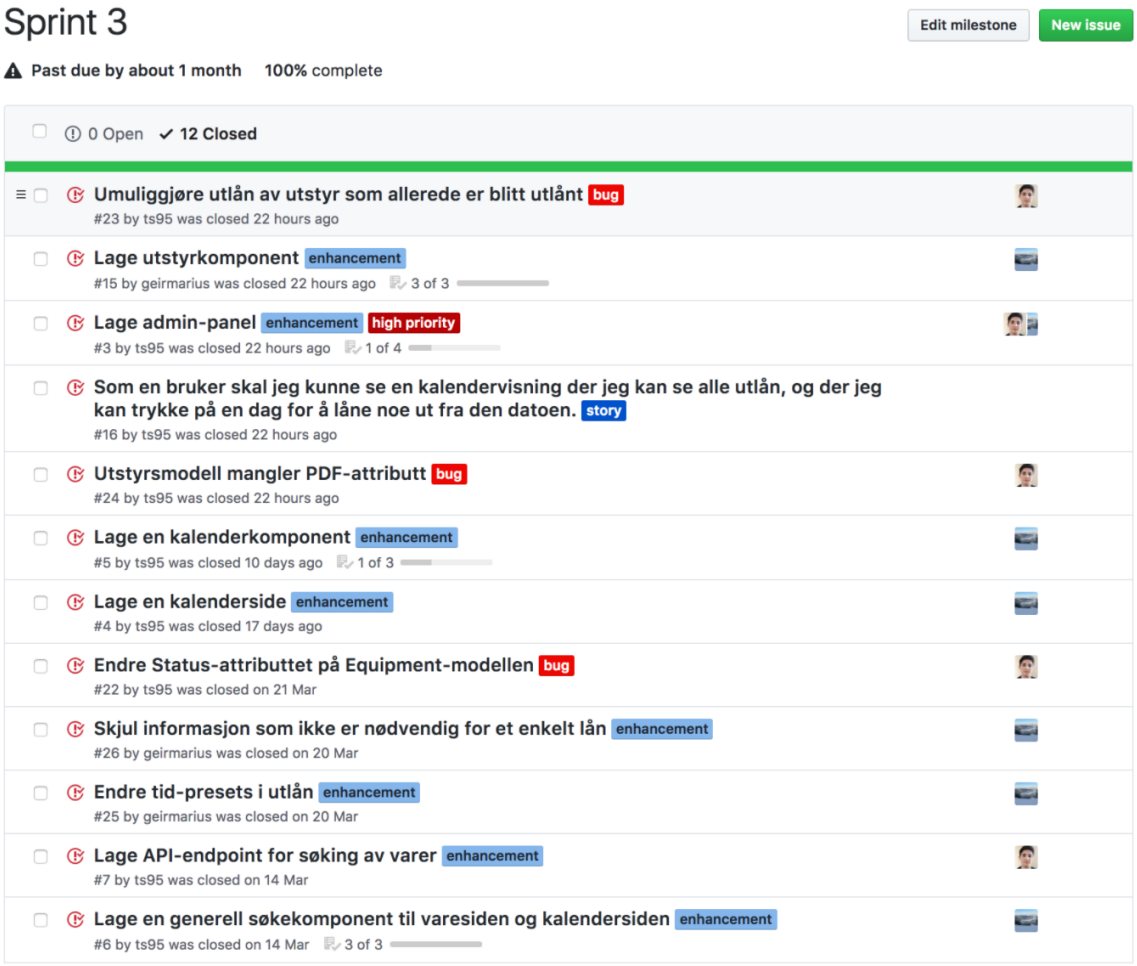
Figur 9: Det første sprintet på GitHub.

Sprint 2 og 3

I de siste to sprintene skulle vi bare fortsette med å implementere funksjonalitet til prototypen. Vi rakk ikke å bli helt ferdige med alle gjøremålene på det andre sprintet, så vi besluttet å flytte dem over til det tredje sprintet. Vi ble heller ikke helt ferdige med det tredje sprintet i perioden vi hadde satt av, så det endte opp med at vi jobbet litt av og på når vi hadde tid slik at vi skulle rekke å lage ferdig en funksjonell prototype.

Som man kan se på figur 9 og 10, så brukte vi noen “tags” til å kategorisere arbeidsoppgavene:

- **enhancement:** en forbedring eller utvidelse av funksjonalitet.
- **bug:** en feil eller mangel.
- **story:** en beskrivelse av noe en bruker skal kunne gjøre i applikasjonen; brukt i forbindelse med scrum-boardet.



Figur 10: Det tredje sprintet på GitHub, med arbeidsoppgaver fra det andre sprintet.

3.2.4 Sprintevalueringer

3.2.4.1 Evaluering av sprint #1

Mandag, 27. februar 2017.

Toni:

Første sprint gikk greit. Jeg hadde allerede satt opp en prosjektstruktur før den første sprinten slik at vi heller kunne bruke tid på å utvikle selve webapplikasjonen. Det var dog et par ting som vi slet litt med i begynnelsen, nemlig å få greie på hva kunden vår ønsket at vi skulle bygge. Det viste seg at vår oppfatning av hva som skulle bygges ikke helt stemte overens med det de ville ha. Dette førte til at de første dagene av sprintet ikke var spesielt produktive. Heldigvis så ble vi fort enige om hva vi skulle lage, så vi satte i gang og rakk å kode en vesentlig del av webapplikasjonen.

Jeg fokuserte mest på serversiden til applikasjonen, men skrev også litt av koden på klientsiden - hovedsakelig kode som har med brukerautentisering å gjøre.

Hva som var lett:

- Å implementere koden for oppretting, endring og sletting av utstyr gikk greit. Det var ikke noe spesielt vanskelig, og jeg støtte ikke på noen store problemer.

Hva som var vanskelig:

- Å modellere databasen. Det er alltid litt vanskelig i begynnelsen å vite nøyaktig hva man trenger med tanke på databasemodeller. Dette ble komplisert ytterligere med tanke på at vi fikk beskjed av kunden om at de ikke ønsket at vi skulle implementere vårt eget innloggingssystem, siden de allerede hadde ett. Av tekniske grunner kompliserer dette ting ganske mye, og vi ble til slutt enige om at jeg bare skulle implementere et brukersystem i vår egen database siden det ville spare oss veldig mye tid.

Vi rakk ikke å bli ferdige med alle storyene vi la inn i første sprint, så vi blir nødt til å flytte dem over til det neste sprintet. Selv om webapplikasjonen har begynt å ta form er den fremdeles ikke funksjonell i dette stadiet.

Geir Marius:

Første sprint gikk med til å lage alle essensielle delene til et interface hvor låne prosessen var tilstede. Jeg jobbet med å sette opp alle *view*sene, med dynamisk kode via angular. Alt som kunne gjøres uten kontakt med server ble forsøkt gjort. Litt gjenstår fremdeles da de bydde på uventede komplikasjoner.

Hva som var lett:

- Sette opp html med bootstrap CSS og MVP temaet. Selv om det til tider tok tid å finne frem til riktig komponent for hva jeg lagde, gikk det aller meste på skinner og var veldig greit å slå opp i dokumentasjonen. For temaet måtte jeg gå inn i kildekoden til eksempeldokumentet.
- Angular sine dynamiske variabler enn bruker i tekst og attributter i HTML var veldig greit å bruke.

Hva som var vanskelig:

- Angular hadde noen løsninger som jeg trengte som var dypt begravet i en nokså rotete dokumentasjon. Det var ofte vanskelig å finne frem til det jeg ønsket å gjøre.

Det var mange runder på stackoverflow og leting i dokumentasjonen. Ofte var det angular 1 som kom opp ovenfor angular 2 som vi bruker.

Brukertesting

Vi var litt sent ute med brukertestingen og fikk ikke gjort den før 28.02.17. Heldigvis fikk vi positive tilbakemeldinger på det vi hadde gjort.

3.2.4.2 Evaluering av sprint #2

Mandag, 13. mars 2017.

Toni:

Mens første sprint hovedsakelig gikk ut på å sette opp grunnlaget for prosjektet gikk det andre sprintet ut på å bygge videre på det grunnlaget. Jeg støtte ikke på noen særlige problemer under dette sprintet, men det var et par små problemer som dukket opp - hovedsakelig databaseproblematikk.

Hva som var lett:

- Å lage API-endepunktene var ikke spesielt vanskelig.

Hva som var vanskelig:

- Til tider var det litt vanskelig å vite nøyaktig hvordan databasen skulle struktureres.
- På grunn av begrensninger i SQLite måtte vi gå over til MySQL. Dessverre hadde MySQL-driveren en del problemer som medførte litt tapt tid.

Geir Marius:

Sprint nummer to gikk med til å sette opp alle komponenter mot server. Hente ut data fra APIet vårt og sende data til samme API.

Hva som var lett:

- Å hente ut data fra vårt eget API var veldig lett takket være god kommunikasjon mellom front-end og backend.
- Konvertere statiske lister til å fungere med dynamisk innhold fra APIet gikk veldig greit, da de var designet til å kunne fungere med det i fremtiden.

Hva som var vanskelig:

- YouTube sitt SDK fungerte dårlig med angular sin SPA implementasjon.

3.2.4.3 Evaluering av sprint #3

Tirsdag, 28. mars 2017.

Toni:

Dette sprintet bydde på en del problemer med tanke på forholdet mellom modellene. Utstyrsmodellen skulle ha et attributt som spesifiserer om utstyret er lånt ut eller ikke. Dette var overraskende vanskelig å implementere, siden det finnes en egen modell for utlån som er "usynlig" for utstyrsmodellen.

Hva som var lett:

- Å gjøre noen småjusteringer på API-endepunktene

Hva som var vanskelig:

- Løse problemer med databasemodellene.
- Ordne status-attributtet på utstyrsmodellen.

Marius:

Kalender komponenten ble forskjøvet til sprint 3, så den sto i fokus store deler av sprintet. Ellers så var det justeringer på alt som ble tatt opp under testingen fra sprint 2. Dette inkluderte bla. endringer i hvordan utlånssiden var strukturert.

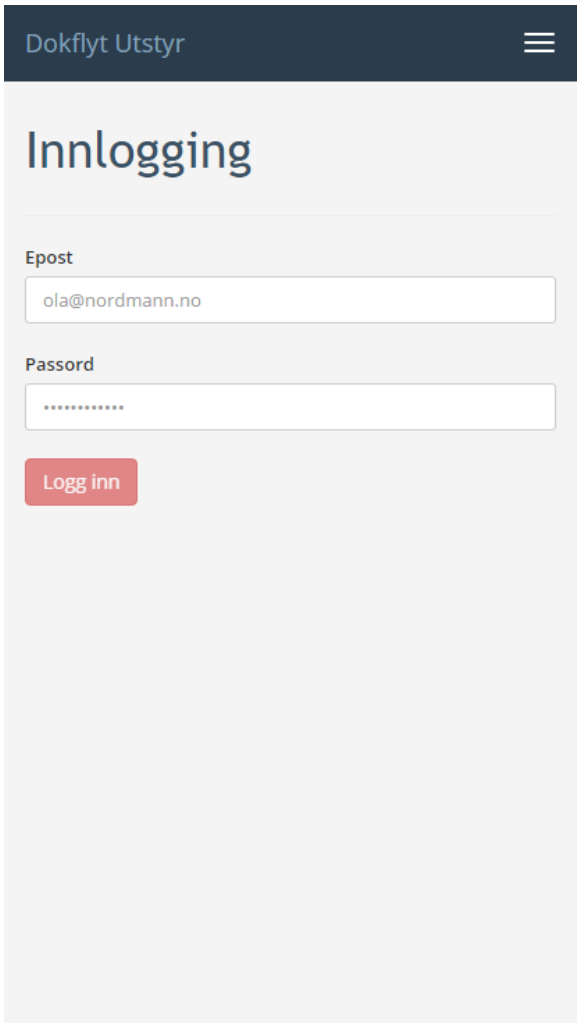
Hva som var lett:

- Sette opp visningen til kalenderen.

Hva som var vanskelig:

- Fikse bugs i utlånssiden hvor datoer ikke ble registrert riktig og hvor visse elementer hadde vanskelig for å bli disabled.
- Registrere hvor på kalenderen en trykket viste seg å ikke være rett frem med slik kalenderen var satt opp via CSS grid.

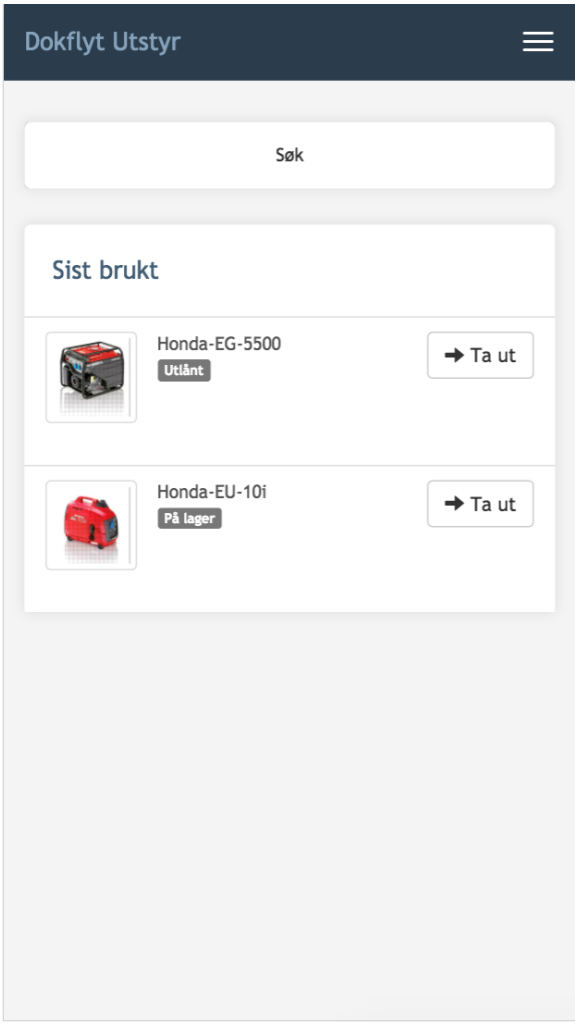
3.2.5 Sidene i webapplikasjonen



Figur 11: Innloggingssiden

Innloggingssiden er en enkel form med to tekstfelt og en knapp. “Logg inn”-knappen er deaktivert inntil brukeren har fylt ut begge feltene.

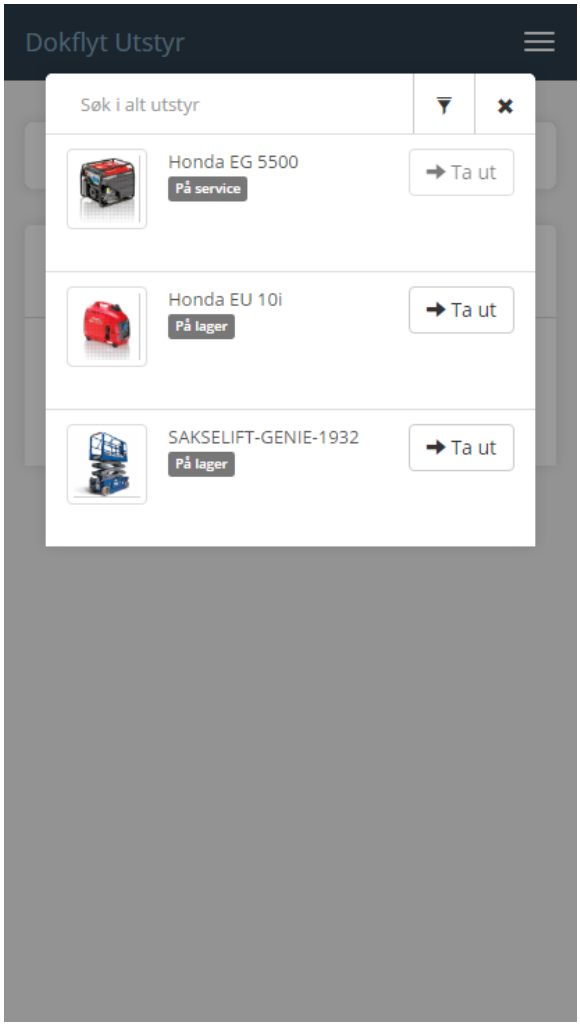
Når man trykker på “Logg inn”-knappen vil formen gi brukeren en tilbakemelding dersom innloggingsopplysningene ikke er korrekte, i form av en feilmelding under tekstfeltet.



Figur 12: Hovedsiden

På hovedsiden får man oversikt over utstyr man sist har brukt slik at man kan finne frem til det raskere neste gang man vil låne det.

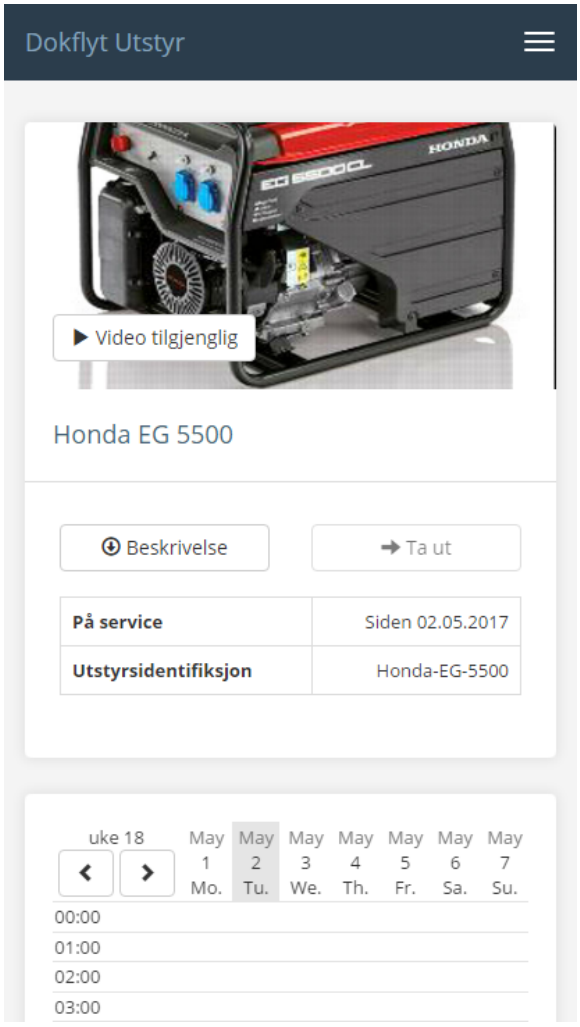
Hvis man trykker på søkefeltet får man opp en søkedialog som man kan bruke til å søke etter utstyr i systemet.



Figur 13: Søkedialog på hovedsiden

Knappen til venstre for x-en øverst til høyre på dialogen veksler mellom forskjellige søkefiltere.

For å søke behøver man bare å skrive i "Søk i alt utstyr" feltet. Listen under vil automatisk oppdateres for å reflektere teksten man har skrevet inn.



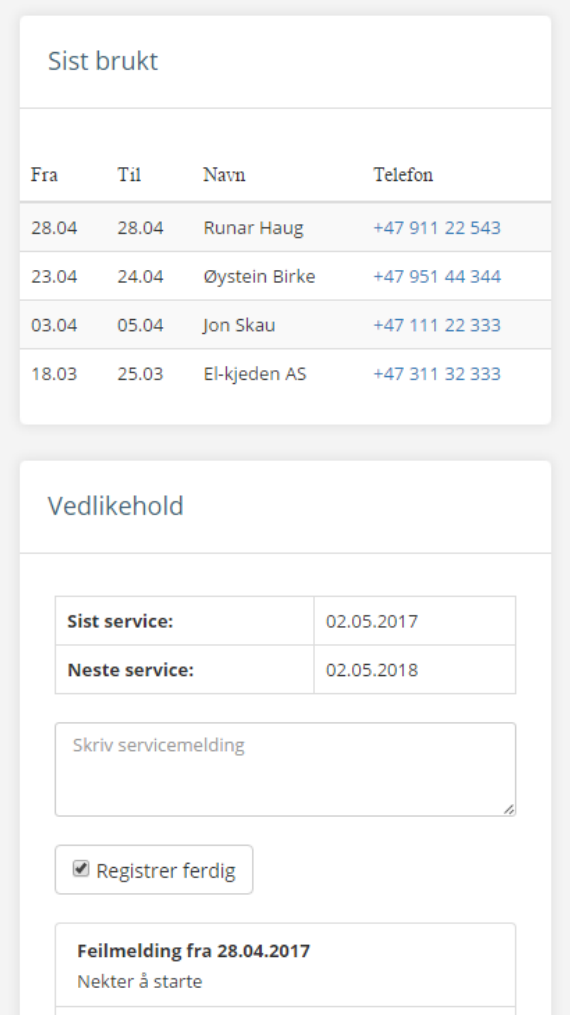
Figur 14: Toppen av utstyrssiden

Utstyrssiden viser et lite bilde av utstyret. Hvis det er en informasjonsvideo tilgjengelig kan man trykke på “Video tilgjengelig”-knappen for å spille av videoen.

“Beskrivelse”-knappen åpner et PDF-dokument med informasjon om utstyret dersom man trykker på den.

I tabellen under “Beskrivelse” og “Ta ut” -knappene ser ser man serviceinformasjon og identifikasjonen til utstyret.

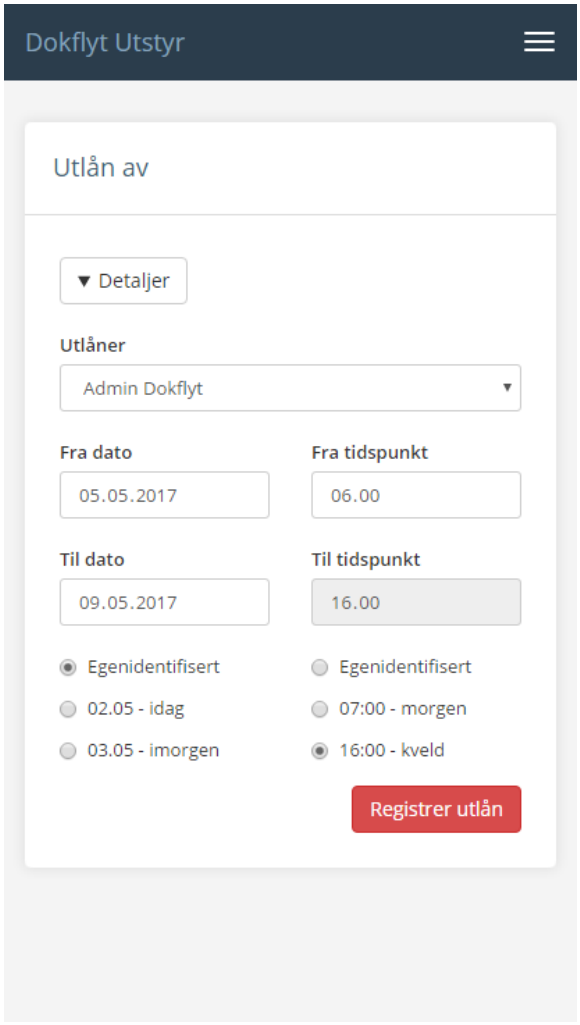
Helt nederst ser man en kalender som viser om utstyret er ledig eller opptatt i et gitt tidsrom.



Figur 15: Bunnen av utstyrssiden

Bunnen av utstyrssiden viser en liste over dem som sist brukte utstyret, med navn og telefonnummer slik at man kan ringe dem. Hvis man er på nettsiden på en mobiltelefon kan man ganske enkelt trykke på telefonnummeret for å ringe det.

Helt nederst ser man informasjon om vedlikehold, slik den vises når utstyret er på service. Tekstboksen vil si «Skriv feilmelding» og du vil se en «post feilmelding» knapp om utstyret ikke er på service.



Figur 16: Registrering av utlån

Dette er siden man kommer til etter å ha trykket på “Ta ut”-kappen på hovedsiden eller utstyrssiden.

Her kan man spesifisere hvem som skal låne utstyret, og i hvilket tidsrom det skal lånes.

«Utlåner», «Fra dato» og «Fra tidspunkt» er hjemt under detalje knappen når man først åpner denne visningen og er da satt til ditt navn og datoen akkurat nå.

Hvis man prøver å låne et utstyr i en periode der det allerede er lånt ut får man opp en rød boks med en feilmelding som informerer om at utstyret er tatt i den perioden.

3.3 Implementasjon av hi-fi prototypen

Hvordan vi gikk frem for å implementere webapplikasjonen i henhold til kravspesifikasjonen vi ble gitt og ekstra opplysninger vi fikk gjennom brukerintervjuer med oppdragsgiveren.

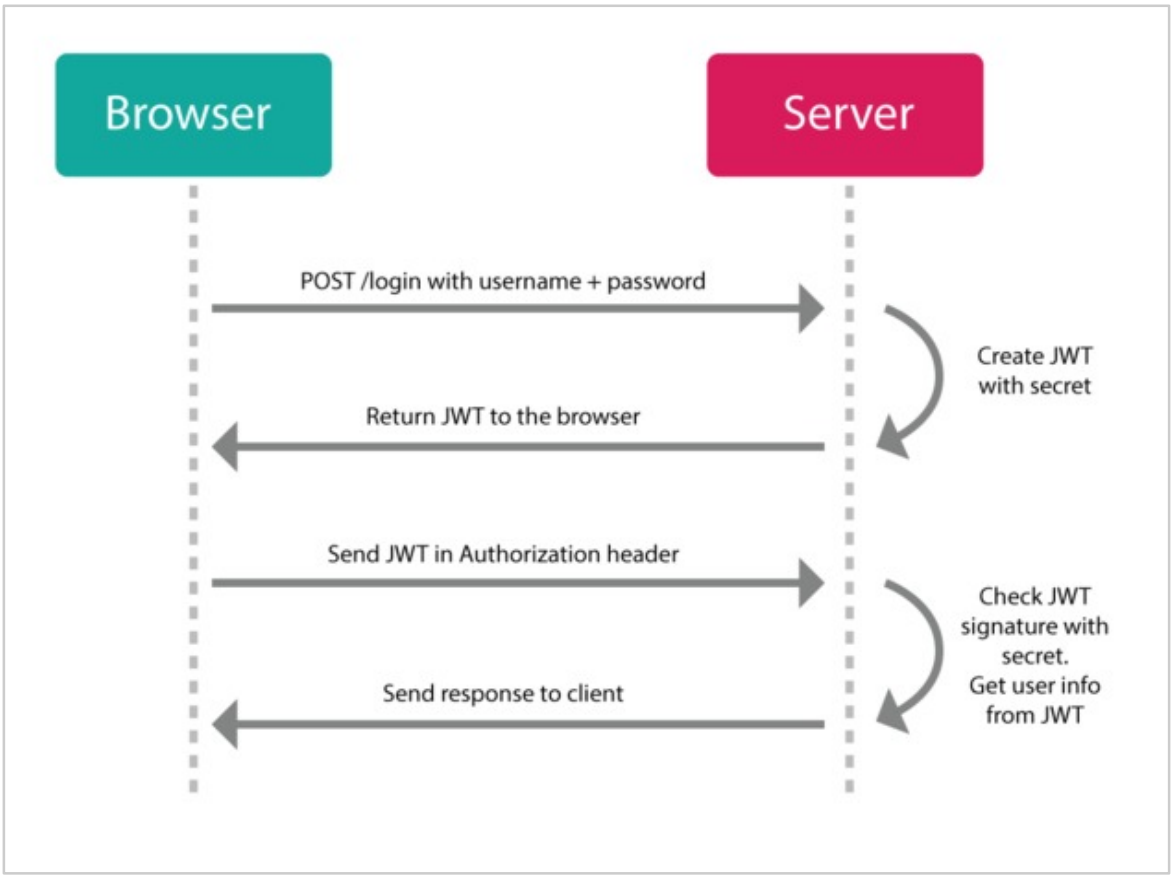
3.3.1 Prosjektmal

Da vi skulle begynne med utviklingsfasen fikk vi en prosjektmal av Dokflyt som de har brukt til sine interne prosjekter. Denne prosjektmalen var bygd opp med teknologier som vi ikke hadde erfaring med tidligere, nemlig Angular 2, TypeScript, og .NET Core. Toni hadde heldigvis tidligere erfaring med programmeringsspråket C#, og vi hadde begge erfaring med både front-end og back-end rammeverk. Dette gjorde det til en smal sak å lære seg Angular 2 og .NET Core.

3.3.1.1 Utfordringer med prosjektmalen

Da vi satte i gang med utviklingen oppdaget vi noen fordeler og ulemper med C# som back-endspråk. Fordelen er at det strenge typesystemet, og de gode feilsøkingsverktøyene, gjør det enklere å oppdage og håndtere feil i koden. Ulempen er at utvikling i C# kan oppleves som litt tregere enn for skriptespråk som f.eks. Python, Ruby eller JavaScript, i og med at språket må kompileres til bytekode som kjøres av .NET-plattformen. Dette gjorde utviklingsprosessen litt tregere, siden man måtte vente på at C# koden skulle kompileres hver gang man gjorde den minste endring i kildekoden. Hadde vi brukt f.eks. Python ville koden blitt kjørt umiddelbart uten noe ekstra ventetid siden Python ikke krever det ekstra kompilasjonssteget.

Helt i begynnelsen av utviklingen til webapplikasjonen oppstod det en del utfordringer med tanke på oppsettet til prosjektmalen. Prosjektmalen tok utgangspunkt i en isomorfisk SPA. At den er isomorfisk vil si at all koden som kjører på front-enden også blir kjørt på back-enden rett før serveren serverer siden til klienten (nettleseren). Dette var problematisk i forhold til hvordan vi satte opp autentisering. Vi brukte JWT til autentisering av brukerne.

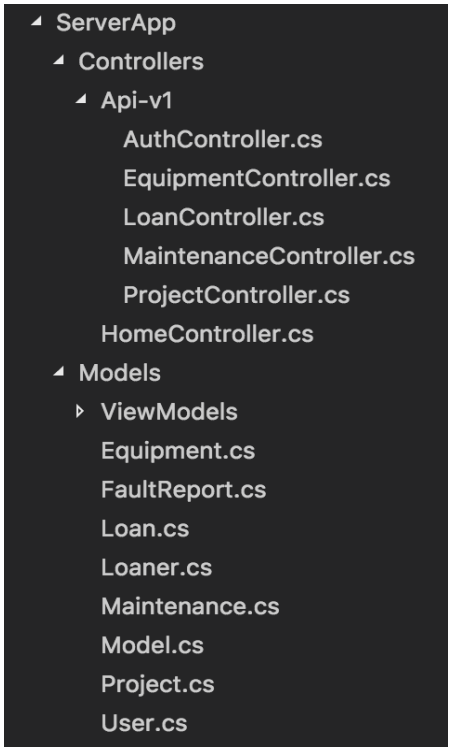


Figur 17: Illustrasjon som viser hvordan JWT fungerer.

Som fig 17 illustrerer funger JWT på den måten at brukeren etterspør en token fra serveren. Det er front-enden som er ansvarlig for å huske denne tokenen, slik at at den kan bevise at brukeren er autentisert ved påfølgende etterspørsler til serveren. For å kunne gjøre dette må tokenen lagres på front-enden på et vis. Den anbefalte måten å gjøre dette på er å bruke local storage API-et i nettleseren. Local storage gjør det mulig å lagre informasjon i nettleseren som kommer til å forbli der også etter at brukeren har lastet inn siden på nytt, eller lukket fanen. Det er her problemet oppstår når man introduserer en isomorfisk arkitektur. Når serveren prøver å forhåndsrendere nettsiden kan den umulig vite om brukeren er innlogget eller ikke, siden tokenen er lagret i nettleseren og ikke på serveren. I og med at serveren ikke vet om brukeren er innlogget eller ikke, kan den ikke rendere siden riktig, dermed mister man den viktigste fordelen med en isomorfisk arkitektur.

For en tradisjonell nettside pleier man å oppbevare autentiseringsopplysningene på serveren fremfor i nettleseren. Vi tenkte at vi kunne prøve å gjøre det på den måten, men det krevde at vi gikk over til et annet autentiseringssystem, og med tanke på at vi allerede hadde sløst en del tid på denne problemstillingen besluttet vi at vi bare skulle gå for en standard SPA som ikke er isomorfisk. En ansatt i Dokflyt anbefalte oss for øvrig å gjøre akkurat dette, siden de også støtte på dette problemet og ikke fant en god løsning.

3.3.2 Utforming av back-enden



Figur 18: Fil- og mappestrukturen på back-enden.

På back-enden brukte vi .NET Core med C# som programmeringsspråk. I og med at webapplikasjonen er en SPA fungerte serveren kun som et API som front-enden kunne bruke til å utføre bl. a. CRUD-operasjoner. Alle kontrollerne befinner seg i en mappe som heter Api-v1. Tanken med dette oppsettet er at det skal være lett å lage en ny versjon av API-et i fremtiden. I og med at vi utviklet en prototype er det nødvendig å lage en ny versjon i fremtiden, siden prototypen vår ikke var designet for produsjonsbruk. Hver kontroller inneholder en klasse med metoder som reflekterer hver API-route. API-routen `/v1/Equipment/List` f.eks. er definert i en metode i `EquipmentController.cs`. I `Models`-mappen ligger de forskjellige databasemodellene. Disse fungerer i samspill med Entity Framework; ORM-en vi brukte. Modellene utgjør grunnlaget for databasetabellene, som genereres gjennom en databasemigrasjon.

3.3.2.1 Problemer med databaseutformingen

Da vi satte i gang med utviklingen av webappen var vi ikke helt sikre på hva vi skulle lage, og det viste seg at vår oppfatning av hva som skulle lages ikke stemte overens med oppdragsgiverens oppfatning. Vi visste dermed ikke hvordan databasen skulle bygges opp i begynnelsen av prosjektet, og måtte ta en del antagelser. Etter noen møter med oppdragsgiveren klarte vi å oppklare forvirringen så vi ble mer bevisste på hva vi skulle lage, men da hadde vi allerede lagd en databasestruktur. Denne strukturen måtte endres på flerfoldige ganger i løpet av tiden vi jobbet med prosjektet. Etterhvert som databasekompleksiteten grodde støtte vi på en del begrensninger i Entity Framework som gjorde det vanskelig å uttrykke visse forhold mellom modeller og attributter på modeller som vi trengte. Disse forholdene og attributtene ville vært uproblematiske i rå SQL, men i og med at vi befant oss bak abstraksjonen til Entity Framework, så kunne vi ikke ta i bruk vanlig SQL. Dette medførte at en del stygg kode ble skrevet for å tvinge frem funksjonaliteten vi trengte. Man burde helst unngå å skrive slik kode, men i og med at dette er en prototype har det ikke så mye å si.

Et annet problem med databasen var driveren vi valgte. For å kunne bruke entity framework må man installere en driver for en gitt SQL-dialekt. I begynnelsen besluttet vi å bruke SQLite, en enkel SQL-løsning, siden prosjektet vårt bare skulle være en prototype, men på grunn av begrensninger i SQLite måtte vi gå over til noe annet. Vi bestemte oss for å bruke MySQL siden vi hadde erfaring med det tidligere. Vi klarte å finne en MySQL-driver, men det var en driver utviklet av en tredjepart istedenfor Microsoft, og den hadde noen mangler med tanke på noe veldig spesifikk funksjonalitet i Entity Framework. Vi endte opp med å prøve et

par forskjellige MySQL-drivere, men falt til slutt tilbake på den første vi fant siden den så ut til å være den mest funksjonelle.

Entity Framework har en standard måte å generere tabeller på ut fra forholdene man har definert mellom modellene sine. Hvis man har en abstrakt klasse og to andre klasser som arver fra den abstrakte klassen vil Entity Framework generere kun én tabell med en såkalt "diskrimineringskolonne". Metadataen til denne tabellen vil inneholde alle feltene i begge subklassene. Denne metadatagenerasjonen skaper en del redundans, siden det innebærer at en rad i tabellen til enhver tid vil ha masse ubrukte kolonner.

```
abstract class Animal
{
    public string AnimalId { get; set; }

    public string Name { get; set; }

    public int MaxGroundSpeed { get; set; }
}

class Dog : Animal
{
    public string FurColor { get; set; }
}

class Bird : Animal
{
    public string FeatherColor { get; set; }

    public int MaxAirSpeed { get; set; }
}
```

Figur 19: Eksempel på en abstrakt klasse i Entity Framework som har to subklasser.

Modellkoden i fig 19 er et eksempel på hvordan man kan lage to forskjellige klasser med én felles baseklasse i Entity Framework. "Animal" beskriver attributter som skal være felles for alle dyr, mens "Dog" og "Bird" klassene beskriver attributter som er spesifikke for de dyrene.

Som standard vil entity framework generere en tabell for dette som ser slik ut:

AnimalId	**Discriminator**	**Name**	**Max Ground Speed**	**Fur Color**	**Feather Color**	**Max Air Speed**
1	Dog	Fido	50	Brown	*NULL*	*NULL*
2	Bird	Birdie	5	*NULL*	Blue	70

Her kan man se redundansen som oppstår ganske tydelig. Siden en hund hverken kan fly eller har pels må de to siste kolonnene stå tomme. I dette eksempelet er det ikke så ille siden det bare er to subklasser og hver av dem har få attributter, men hvis det hadde vært fem subklasser med seks attributter hver hadde redundansen blitt ganske stor.

Entity Framework tilbyr faktisk en alternativ måte å generere tabellene på som ikke er like redundant. Hadde man brukt den generasjonsmetoden fremfor den ovenfor ville man endt opp med tre tabeller istedenfor én, der subklassene er egne tabeller med fremmednøkler som peker til én tabell som fungerer som den abstrakte baseklassen. Vi ville helst bruke denne metoden, men driveren vi brukte hadde dessverre ikke implementert støtte for det.

Det er dog en fordel med å kun ha én tabell: spørringene blir mindre komplekse. Når man har flere forskjellige tabeller må disse forenes. Det skaper økt kompleksitet, som medfører at spørringene blir tregere.

3.3.2.2 Problemer med autentiseringssystemet

Dokflyt ville helst at vi skulle overlate autentisering til dem, og implementere en slags “placeholder” istedenfor å implementere et fullverdig autentiseringssystem. Vi kunne brukt tid på å lage en slags abstraksjon som gjør det mulig “å plugge inn” et nytt autentiseringssystem i fremtiden, men det ville kostet oss mye tid så vi besluttet å ikke gjøre det. Vi endte opp med å lage vårt eget autentiseringssystem. Dessverre ble dette noenlunde ufullstendig siden vi ikke hadde nok tid til å implementere det fullstendig. Systemet ble designet med tanke på at det skulle være mulig for en leder av et prosjekt å låne utstyr for det prosjektet, men vi rakk ikke å gjøre ferdig den delen.

3.3.3 Utforming av front-enden

Hele front-ended til applikasjonene er bygget på rammeverket angular 2. Det vil gjøre at all navigasjon er gjort på klient-siden og at du ikke henter inn et nytt HTML dokument hver gang du går imellom de forskjellige sidene i webappen. Videre lar det oss dele opp alle sidene inn i komponenter, slik at store og tunge sider er bygget ved å sette sammen mindre komponenter. Det gjør det mer oversiktlig og jobbe med de forskjellige sidene og lar oss bruke samme komponent over flere sider som oppfordrer til gjenbruk av kode.

3.3.3.1 Utfordringer

Angular har et sterkt forhold til typescript, og for å kunne lage gode angular komponenter er det nødvendig med kompetanse i typescript. Begge disse var relativt nye for oss begge og bydde på en del utfordringer i form av både ny syntaks og nye ukjente måter å tenke på.

Typescript sitt konsept var vi veldig glad i å tok veldig kort tid å lære seg. Det inkluderer å lage interfaces for objekter, altså å kunne skrive hva et objekt kan og ikke kan inneholde, men også typesystemet som lar oss si hvilken datatype en variabel skal ha. F.eks. kan en variabel som er satt til `String` aldri være et nummer.

En angular komponent (enten en hel side eller en mindre modul), begynner med javascript koden `class MyComponent {}` som er kjent kode for både javascript og nesten alle andre språk. Men med angular og typescript ønsker man å benytte seg av “decorators” som, for javascript utviklere, er et relativt ukjent begrep og byr på litt annerledes tankegang. I javascript er man også vant til å bruke metoder på en klasse og all koden som skal kjøres når en ny instans av klassen opprettes i `constructor` metoden. I angular er denne nesten helt fjernet og brukes kun for å gi importerte klasser et lokalt navn. Typescript har også noen helt spesielle regler for hvordan en klasse kan brukes. I Javascript er det ikke mulig å sette en variable på en klasse og man bruker “getters” og “setters” som metoder på klassen. Typescript lar deg skrive variabler rett på klassen som er helt i strid med hvordan javascript klasser fungerer.

Generelt sett byr dette på nye tankemønstre som gjorde at koding var litt mer saktegående i starten, men som løste seg etterhvert når vi forsto litt mer om hvordan typescript og angular fungerte.

Større problemer kom når vi skulle begynne å bruke angular sine proprietære løsninger, som angular sin “http” og “route” modul. Http-modulen lignet veldig på javascript sin egen løsning

“fetch”, men endte opp med å fungere på en ganske så annerledes måte. Det var forvirrende hvor like de var, men hvor forskjellige de oppførte seg. Både http- og route-modulen introduserte et nytt konsept hvor du må abonnere på det du ønsker å hente ut for å få noe svar i det hele tatt - selv om du bare skulle ha et eneste svar. Det var også veldig viktig å huske å avslutte abonnementet når komponenten koden kjørte i ikke lenger var bruk, men for og legge til litt mer kompleksitet er det enkelte abonnenter som avslutter seg selv. For å bli klok på det var vi nødt til å gjøre et google-søk hver gang vi startet et abonnement.

Et problem vi knotet lenge med var skjemaer. Å bruke HTML-standarden sitt `<form>` element med `name` attributten på alle datafelt som skulle sendes er ikke kompatibelt med angular. Angular har bygget et helt rammeverk rundt skjemaer som er ganske komplisert og tungt å sette opp - spesielt om det er første gang du prøver deg på det. Det krever sine egne attributter på både `<form>` elementet og alle datafelt. All dataen er deretter håndtert via javascript med en av angular sine mange metoder for å sette opp en form på. Hvor vi burde bruke hvilken metode ble vi aldri kloke på.

For å finne ut av hva angular lot oss gjøre og hva de hadde proprietære løsninger for var angular sin dokumentasjon ofte nevnt som et svar. Problemet med dokumentasjon-siden var at den sikkert var veldig fin for folk som var godt inne i angular-miljøet, men den var ekstremt tunglest for oss som ikke var så godt kjent. Det tok lang tid å tyde hva som sto der og ofte endte det opp med at vi måtte gå andre steder for å få et komplett svar.

4 Diskusjon

I dette kapittelet skal vi diskutere prosjektet vi gjennomførte, drøfte resultatene av det, gjøre rede for arbeidsmåten vår, og henvise til problemer som oppstod underveis, samt vurdere om noe kunne blitt gjort annerledes.

4.1 Metoder

4.1.1 Verktøy og gjennomføring

Vi følte vi hadde en rimelig god plan for å gjennomføre prosjektet og planla og bruke alle verktøyene vi hadde til rede til sitt fulleste, men så er det jo gjerne slik at bruk av enkelte verktøy synker gradvis når en ikke ser noe verdi i det. Det skjedde med enkelte av planene vi hadde som f.eks. teamgantt sin timelogging funksjon. Det var rett og slett ikke av noen stor verdi for oss.

Sammen med milepælene våre og tidsplanene vår ble det opprettet et gantt-diagram som var en ekstremt god ressurs og hjalp oss daglig med å organisere oppgaver til oss selv. Sammen med github issues og github milestones var det ekstremt lett å se hva som måtte prioriteres og hva som måtte bli forskjøvet til et senere tidspunkt lenge før det egentlig skulle være ferdig. Det ga oss gode planleggingsevner - spesielt i kodeperioden.

At alt av bachelorrelaterte dokumenter og filer lå på google drive og github gjorde at vi aldri hadde problemer med å finne frem til eldre dokumenter og bilder som vi trengte på et senere tidspunkt. Alt var bare å hente ut i strukturerte google drive mapper og alt av kode og kodeprosjekt relatert som backlogg og issues lå i et enkelt github repository. Det er den mest problemfrie løsningen vi kan se for oss.

4.1.2 Konseptutvikling

Vi gikk frem med brukerintervju og use cases på en effektiv måte. Vi fikk gått igjennom mye av konseptutviklingen over et veldig kort tidsperiode og kom kjapt i gang med kodebiten av prosjektet. Litt kritikk er kanskje på sin plass og det skal sies at ting endret seg i løpet av prosjektet, men disse endringene er fullt mulig også ville oppstå med en mer grundig konseptutviklingsfase. Det ryddet også plass for en ekstra kode-sprint som var sårt trengt, men på den andre siden var det kanskje derfor vi trengte den «fjerde sprinten».

4.2 Resultater

Resultatene er det vi sitter igjen med etter å ha gjennomført prosjektet. Dette omfatter analyser, intervju, testing, og prototype.

4.2.1 Analyser

Vi gjorde litt research før vi satte i gang med prosjektet, og hadde noen intervjuer med oppdragsgiveren får å finne ut av hva vi skulle lage, og om det allerede fantes en lignende løsning der ute. Vi konkluderte med at dette ikke fantes en løsning der ute som er akkurat som vår, men at det fantes mange lignende løsninger der ute, dessuten blir applikasjoner som dette hovedsakelig brukt internt i bedrifter som gjør det vanskelig å finne dem med et enkelt Google-søk.

4.2.2 Brukerintervju

Det vi fikk ut av brukerintervjuene var ekstremt verdifullt for applikasjonen da uten dem ville applikasjonen sannsynligvis hatt et helt annet fokus. F.eks. hadde vi fra starten av fokus på sporing av utstyr, men i intervjuene viste deg seg å ikke være viktig for brukerne i det hele tatt.

Det stemmer bra med hva vi har sett fra andre rapporter, hvor også der brukerintervjuene har endret helt retning på prosjektet.

4.2.3 Testing

Ved slutten av hvert sprint testet vi applikasjonen for å avduke problemer, feil, eller mangler. Vi avtalte også et par møter med oppdragsgiveren slik at de kunne teste applikasjonen. Siden Toni hadde sin egen server kunne han laste opp applikasjonen på den slik at den kunne testes av oppdragsgiveren og andre personer.

4.2.4 Prototypen

Vi har utviklet en utlånsløsning (prototypen) i form av en webapplikasjon som kan brukes av montører i en bedrift til å låne utstyret i bedriften. Applikasjonen holder styr på alle utlån slik at montørene kjapt og enkelt kan sjekke om noe er lånt ut eller ikke, samt oppgi feil eller mangler på utstyret. Prototypen er funksjonell, og vi har implementert nok til at den kan bli testet på ekte brukere for å finne ut av om en slik løsning vil være effektiv. Den er dog ikke komplett nok til å kunne taes i bruk uten å gjennomgå noen feilrettinger og utvidelser først.

4.3 Videre arbeid

Prosjektet vi overleverer til Dokflyt AS er en fungerende webapplikasjon som fremstiller en potensiell implementasjon av kravspesifikasjonen vi fikk av oppdragsgiveren slik den ble tolket av oss, med tilleggsopplysninger vi fikk gjennom intervjuer med oppdragsgiveren. Det er dog kun en prototype, og er dermed uegnet til bruk i et produksjonsmiljø. Vi ønsket i utgangspunktet å utvikle en ferdig løsning som kunne brukes i et produksjonsmiljø, og som ikke krevde så mye ekstra arbeid, men grunnet problematikken som oppstod i utviklingsfasen og mangelen på tid, rakk vi ikke å bli helt ferdige.

Vår anbefaling til Dokflyt AS er at de lager en ny versjon av det eksisterende API-et som tar utgangspunkt i den første versjonen, og at de gjør en refaktorering av koden på front-enden slik at den blir mer konsistent. Å lage en ny versjon av API-et burde gå greit siden ASP.NET Core MVC gjør dette enkelt.

Selve designet og funksjonaliteten har vi implementert bra, og den behøver bare noen få omjusteringer, mener vi.

4.4 Evaluering av gruppearbeid

Vi har samarbeidet godt i løpet av denne perioden. Vi klarte å utnytte tiden der vi ikke jobbet med utplasseringen godt nok til å gjennomføre gjøremålene vi hadde definert for oss, og vi møtte opp på b.la. biblioteket regelmessig slik at vi kunne jobbe sammen.

Vi har vært på samme gruppe før, og det har gått veldig bra siden vi har veldig kompatible ferdighetssett. Toni kunne ta seg av back-endutviklingen, mens Geir Marius kunne håndtere mesteparten av front-endenutviklingen, særlig komponentene.

4.5 Organisering

Vi prøvde å følge et sett med rutiner og regler for å sørge for at ting ble gjort og at vi hadde styr på hvordan vi lå an.

4.5.1 Rutiner

Regelmessige fysiske oppmøter i f.eks. grupperom, biblioteket eller andre steder der man kan arbeide. Minimum 3 dager i uka, gjerne mer hvis man vil.

Dette fikk vi til. I og med at vi begge jobbet to ganger i uka (torsdager og fredager) var det gunstigst for oss å jobbe på mandager, tirsdager, og onsdager. Det ble sjelden noe mer enn de tre dagene, da vi helst ville slappe av i helgen slik at vi kunne ha energi til den kommende uken.

Logging av arbeidstid. Når man jobber med en oppgave som er definert i Gantt-diagrammet må man passe på å notere hvor mange timer man har jobbet.
Tanken med dette var å få oversikt over hvor mye vi jobbet. I begynnelsen innførte vi timer i teamgantt, men etter en stund sluttet vi med det siden det var litt tungvint og vi ikke tjente noe særlig på det.

4.5.2 Regler

Som et minimum må hvert gruppemedlem legge inn 7 timer arbeid daglig minst 5 ganger i uka (4 ganger i uka hvis man fremdeles er utplassert i bedrift). Ved slutten av uka vil gruppelederen undersøke om timeloggen stemmer overens med antall timer man skal jobbe.
Formålet med denne regelen var å sørge for at vi ville jobbe regelmessig med prosjektet istedenfor å sluntre unna. Det gikk dessverre ikke så bra. I begynnelsen fikk vi det til, men etter en stund ble vi utbrente, og på noen dager var det ikke mulig. Toni reiste tilbake til Gjøvik fra Oslo på mandag morgen, og kunne dermed ikke være på skolen før 12-tiden. Vi måtte også ha timer med veilederen i blant som tok tid, så dette resulterte i at vi som regel ikke jobbet mer enn 5 timer på en typisk dag. Gruppelederen (Toni) undersøkte timeloggen i begynnelsen, men sluttet å gjøre det etter at vi sluttet å føre timer.

Hvert gruppemedlem må møte opp på campus klokken 9 om morgenen senest og bli der til de 7 timene er unnagjort (dette må gjøres minimum 3 dager i uka). Man jobber altså fra kl 9.00 til kl 16.00.
I og med at arbeidsdagene våre i forbindelse med praksisplassene våre varte fra kl 9.00 til 16.00, tenkte vi at vi kunne gjøre det samme med gruppearbeidet, men grunnet Tonis pendling og faktumet at ingen håndhevde denne regelen endte det opp med at vi som regel jobbet fra kl 10.00 eller 12.00 til kl 16.00. Vi klarte i det minste å gjøre det tre ganger i uken gjennom hele prosjektet.

4.6 Fordeling av arbeid

Arbeidet var fordelt slik at vi jobbet med tingene hver av oss kunne best. Toni har erfaring med både front-end og back-end, mens Geir Marius hovedsakelig har erfaring med front-end, dermed ble arbeidsfordelingen slik at Toni skrev all koden på back-enden og hjalp til litt

på front-enden - hovedsakelig de delene av front-enden som skulle knyttes opp mot back-enden. Geir Marius utformet omtrent alt designet på front-enden og implementerte de forskjellige komponentene som søkedialogen og kalenderen.

4.7 Problemstilling

Oppgaven vår var å utvikle et utlånssystem i form av en webapplikasjon for Dokflyt AS med et design som skal være brukervennlig, intuitivt, og mobiltilpasset. Vi har gjennomført en analyse av applikasjonen og situasjonen den skal brukes i, lagd skisser og wireframes, samt utviklet en fungerende prototype i form av en webapplikasjon. Tilbakemeldingene vi fikk tyder på at besvarelsen vår av problemstillingen virket lovende for oppdragsgiveren. Designet vårt er elegant, oversiktlig, og lett å ta i bruk – også for dem som ikke er særs datakyndige.

Utlånsfunksjonaliteten i applikasjonen er tilgjengelig fra hvor som helst i applikasjonen, slik at man ikke må rote i menyer og verdilister for å låne noe, og applikasjonen husker de siste lånene man har gjort slik at man kjapt kan låne utstyr man pleier å låne ofte.

Det er dog rom for forbedringer. Det ville vært fordelaktig om applikasjonen hadde hatt mer automatisering i form av sporingsbrikker som man kan feste på utstyret for å sjekke om det er på lageret eller om det er utlånt allerede. Vi ønsket å prøve oss på dette i begynnelsen, men vi skjønte fort at det ville være for tidkrevende for oss.

5 Konklusjon

I dette kapittelet konkluderer vi rapporten og drøfter hvordan det har gått med prosjektet.

5.1 Resultatet

Vi utviklet en fungerende prototype i form av en webapplikasjon for bedriften Dokflyt AS. Den kan brukes til å teste hvorvidt en slik løsning vil fungere i praksis, og hvilke effekter den har. Dersom den viser seg å være en suksess skal det være lett å videreutvikle prosjektet slik at det kan taes i bruk.

Rapporten vi har skrevet beskriver gjennomførelsen av prosjektet fra start til slutt, og presenterer hvordan det har vært å gjennomføre prosjektet med tanke på planlegging, organisering, problematikk, og utbytte. Rapporten viser hvilke metoder vi har anvendt i gjennomførelsen, og hvordan vi har anvendt dem.

5.2 Samarbeidet

Vi hadde et godt samarbeid fra start til slutt. Vi kom tidlig i gang med utviklingen, og rakk å lage ferdig en fungerende, brukbar versjon av prototypen. På dagene der vi ikke jobbet med utplasseringen møttes vi for å jobbe sammen på skolen. I og med at vi klarte å gjøre det hver uke gikk prosjektutførelsen ganske bra.

6 Referanser

1. WCAG Overview ◦ Web Accessibility Initiative ◦ W3C [Internett]. W3C Web Accessibility Initiative (WAI). 2017 [sitert 8. mai 2017]. Tilgjengelig på: https://www.w3.org/WAI/intro/wcag

2. Binstock A. Excellent Explanation of Dependency Injection (Inversion of Control) [Internett]. JavaWorld. 2017 [sitert 8. mai 2017]. Tilgjengelig på: http://www.javaworld.com/article/2071914/excellent-explanation-of-dependency-injection--inversion-of-control-.html

3. C Sharp (programming language) [Internett]. Wikipedia. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

4. TypeScript [Internett]. Wikipedia. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://en.wikipedia.org/wiki/TypeScript

5. ASP.NET MVC [Internett]. Wikipedia. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://en.wikipedia.org/wiki/ASP.NET_MVC

6. Angular (application platform) [Internett]. Wikipedia. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://en.wikipedia.org/wiki/Angular_(application_platform)

7. Visual Studio Code [Internett]. Wikipedia. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://en.wikipedia.org/wiki/Visual_Studio_Code

8. Npm (software) [Internett]. Wikipedia. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://en.wikipedia.org/wiki/Npm_(software)

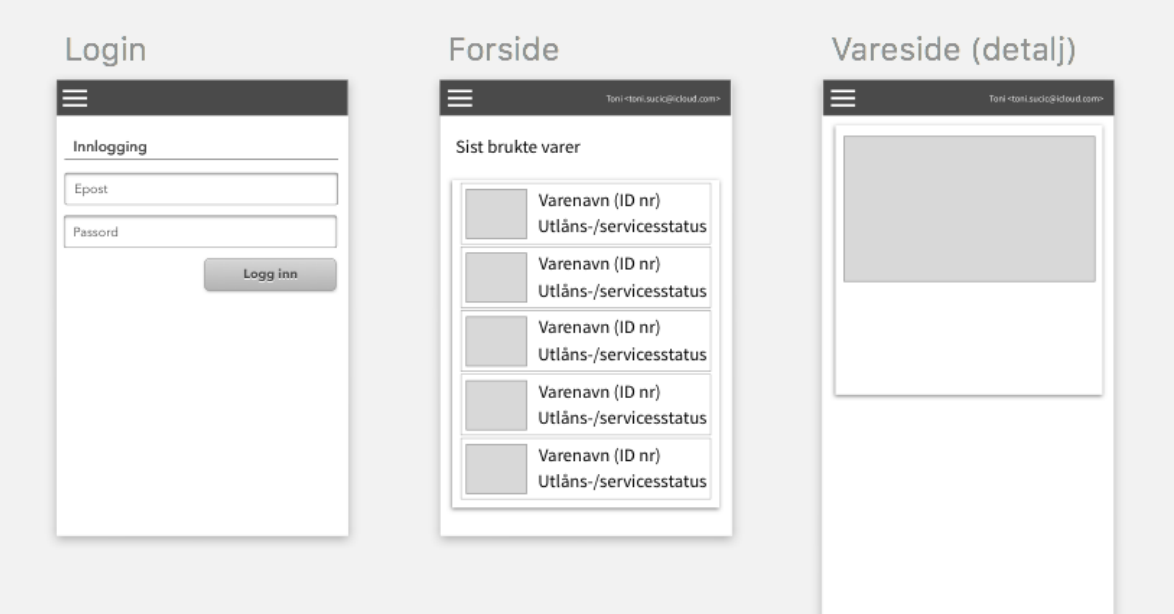
9. Webpack: An Introduction - Wisdom Geek [Internett]. Wisdom Geek. 2017 [sitert 3. mai 2017]. Tilgjengelig på: https://www.wisdomgeek.com/web-development/webpack-introduction/

10. JWT.IO - JSON Web Tokens Introduction [Internett]. Jwt.io. 2017 [sitert 8. mai 2017]. Tilgjengelig på: https://jwt.io/introduction/

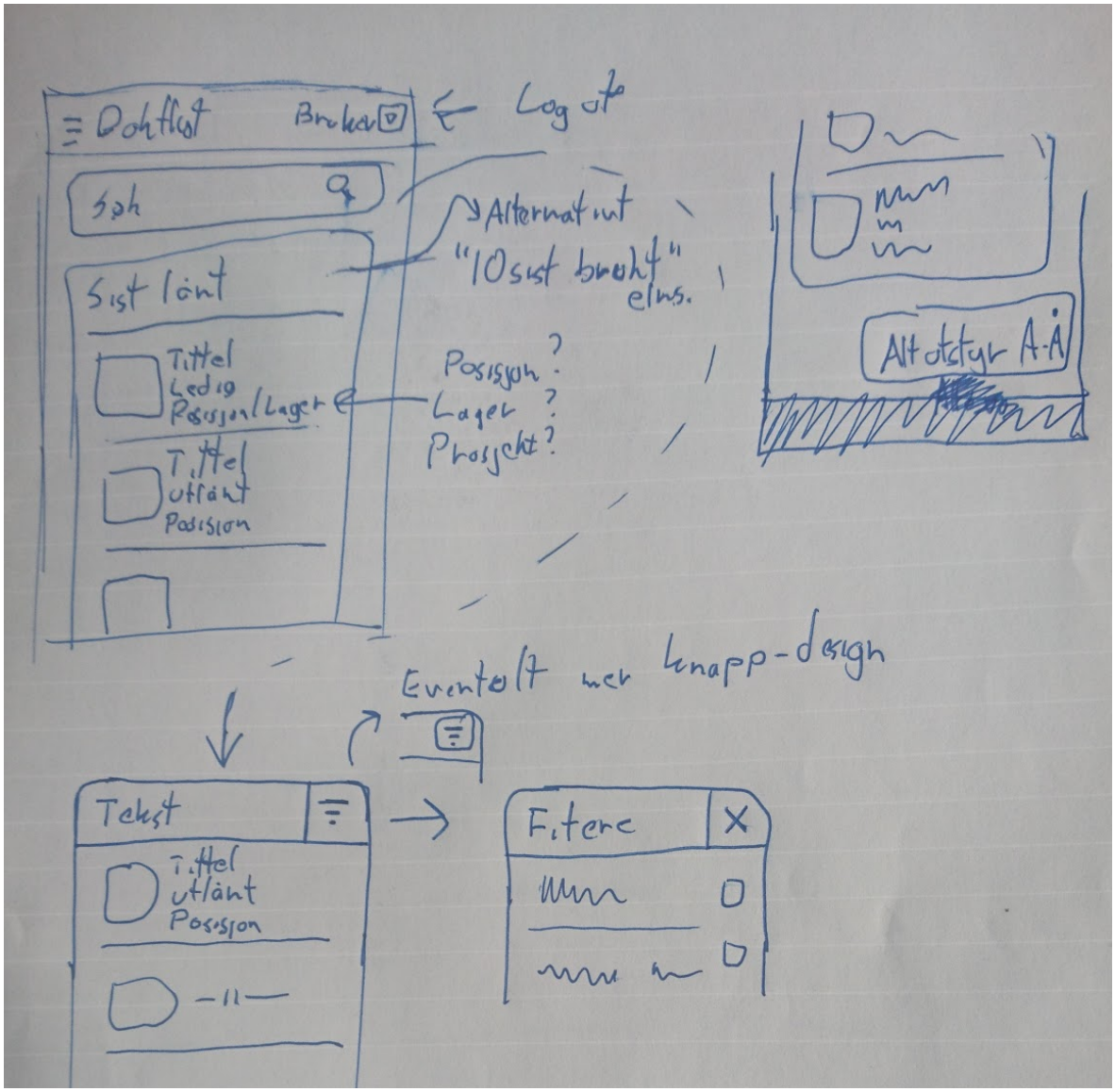
11. Brehm S. Isomorphic JavaScript: The Future of Web Apps – Airbnb Engineering & Data Science – Medium [Internett]. Medium. 2017 [sitert 8. mai 2017]. Tilgjengelig på: https://medium.com/airbnb-engineering/isomorphic-javascript-the-future-of-web-apps-10882b7a2ebc

7 Vedlegg

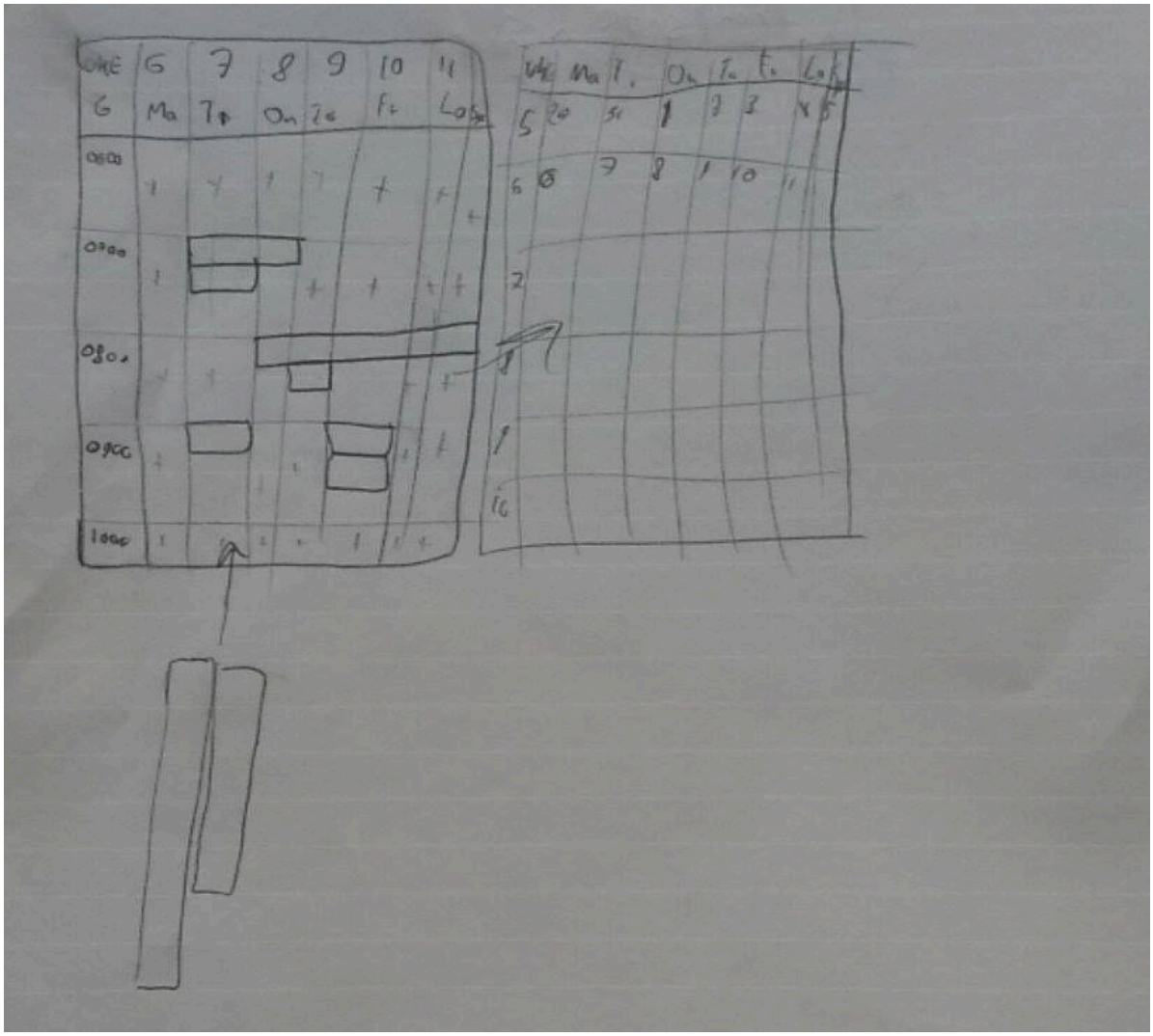
A Skisser



Figur 20: Wireframes laget med programvaren Sketch.



Figur 21: Wireframes fra hjemmeskjermen.



Figur 22: Tidlig utkast av kalenderkomponenten.

B Notater fra intervju med oppdragsgiveren

Spm. til Dokflyt-intervju

- Hvordan foregår loggføring av utstyr nå?
- Hvor stor kontroll ønsker du at det er på utstyret? (Hvor nøyaktig posisjon, tid etc.)
- Hva slags type utstyr er det?
- Hvor stor kontroll ønsker du å ha på det du loggfører? Ønsker du selv å sette posisjon, tid, varenummer etc.?
- Hvor ofte blir utstyr lånt ut og logget?
- Hva skal denne applikasjonen erstatte?
- Hvilke behov kan denne applikasjonen tilfredsstille, og hvilke behov ønsker dere at den skal tilfredsstille?
- På hvilken måte kan applikasjonen bidra til tidsbesparelser og økt produktivitet blant brukerne?
- Hvor gammel vil den gjennomsnittlige brukeren være?
- Hvor IT-kyndig vil den gjennomsnittlige brukeren være?

Møte #1

SPM 1: Blir loggført i en bok. (kalender) Skriv opp når du skal ha. (klarer ikke skrive seg opp i boka! >:( ). Når du kommer på lageret er varen borte fordi noen lånte den uten å skrive opp. Mindre varer blir lånt ut via et excel ark (logg inn).
Vanskelig og ha oversikt.
Noen bruker korktavle! … og noen bruker kalender i outlook. Boka er nok mest vanlig.

SPM 2: Det viktigste er at det er der når du henter det. Greit å vite hvor det er. “Er det på lager eller ikke?”. Kommer ikke til å kjøre å leite etter det. “Hvem har det?”

Spontant spm. (Toni): Kategorier: Det er to klasser utstyr: Stort (henger, lift) og mindre utstyr som passer i baksetet på bilen. Kan kanskje eies av en person og ikke bedriften. En bok for hver stor ting.

SPM 4: Minst mulig. Boka er enkel. Skrive navn, dato og nummer. Folk skriver ikke i boka. Hadde vært fint om det gikk automatisk. Varsel om du ikke henter utstyr på reservert utstyr. “Hvorfor tok du ikke ut varen idag?” glemte du å endre dato? Også varsel på at utstyr ikke er levert inn etter tiden. (Vil ikke ha det på småting sa han, men store ting veldig viktig)

SPM 5: Minst en bruker som låner daglig. Kommer litt an på hvor mye du legger inn. Utstyr kommer til å være ute nesten konstant. Skal vi legge inn 15 eller 50? Kommer litt an på hvor god løsningen være.

SPM 7: Noen må alltid gå bort å skrive seg inn i boka. Låner du utstyr også er det ikke der når du skal ha det fordi en annen dude ikke så i boka. Viktig å vite hvor det er om en slik situasjon skjer i fremtiden.

Spontant spm. (Toni): abonner på varslinger på spesifikt utstyr: Ikke noe for meg.

Spontant spm. (Toni): Låner folk det samme om igjen uten å gå til noe nytt utstyr: Det jeg lånte sist kommer jeg til å låne igjen. De siste utlånene mine blir nok en liste over favoritter.

SPM 8: 40-50 år er gjennomsnittet. Neida. Joda. Det er et stort sprik her. Det er nok i det øvre sjiktet de fleste er. Det er å legge seg på enkelt UI. Skal det være brukt må det være enkelt. De har helst lyst til å bruke boka - boka funka i alle år.

SPM 9: Liten grad. De har blitt trykka ned på et nettbrett. De bruker allerede dokflyt.

Spontant spm. (Toni): Bruker folka smarttelefon: Ja, nettbrett for noen og smartphones for alle. De som er ansvarlig sitter på PCen.

EKSTRA DIALOG:

Masse penger på at noen går over utstyret jevnlig (1-2 ganger i året) Istedenfor at det bare detter sammen plutselig.

Første jeg har lyst til å se er det du lånte sist. Kan jeg se kalenderen til utstyret om jeg klikker på den? Ellers vil jeg søke. Vil også gjerne se hvem som låner mitt utstyr.

Admin legger inn utstyr, der han også legger inn bruker.

Tre bedrifter kan dele på varer. Da skal utstyr fra et annet firma koste penger.

Burde kanskje være admin som bestemmer hvem som bestemmer hva slags fellesutstyr en kan låne, slik at dyrt utstyr ikke blir lånt ut? Ikke helt sikker på dette enda.

Når du gir fra deg utstyr, hva har du gjort av vedlikehold. Sjekk av på en sjekkliste hva du har gjort. Eksempel: Når du leverer den tilbake, er dekka ok? Funker lysa? Er det i samme stand som det var før? En regn rutine: 2 - 3 spm. Må kunne registrere feil - et slags kommentarsystem. En vare burde ha en status. (I drift, til reparasjon, til service)

Møte #2

Bruker dokflyt i et år. Det er litt utfordringer med nettbrett og mobildekning

Når jeg går inn på applikasjonen vil jeg se et oversiktsbilde over alle varer og hvor de er om de er utlånt. “Oh den vinsjen trenger jeg nå. Ja den er på lager da kan jeg reise ned og hente den”.
Som en slags forside.

Favoritter - vil du låne samme utstyr om igjen? Tror ikke det er noe kritisk for oss.

Søkefunksjon

Kan også brukes av de som driver med vedlikehold. Sånn at de kan se hvor lenge siden ting var på service. Varsler om at denne varen må på service (1mnd før i tid). Det må være slik at servicemannen setter inn ny dato etter hver gang.
Noe utstyr har service årlig, annet har det ikke. Kommer an på bruken.

Boka ligger inne på lageret. ATVer etc. står litt tilfeldig. Vet ikke om ATVer er lånt ut.

Det er sånn ca. 50 stk annsat.

Vi er ikke avhengige av å vite hvor utstyret er, men hvem som har det.

Strekkode - butikk. Eller en unik ID. (QR-kode kanskje?)

Hvem legger inn utstyr? Lagersjef har ansvar og er admin. For at han skal gjøre jobben må folk si ifra at utstyr er ødelagt.

Kalender? Ja, vi setter av utstyr. Sånn det fungerer idag er at vi må sette opp en papplapp på utstyret med når vi trenger det.

Lærer lagermannen opp til å gjøre kontroll av dvs. Utstyr selv. Så alt av utstyr kan gå inn i dokflyt.

De fleste er komfortable med nettbrett. Telefon har litt liten skjerm, men gjerne ha det på telefon også.

Utstyr blir lånt ut hver dag. Liften er ute konstant. Vi har konteinere. Vi har agregat.

På større varer kan det være lurt med et varslingssystem om du ikke låner ut ting, men vet ikke om det blir litt mye stress. Er ikke det som står øverst på ønskelista.

Støtte for flere avdelinger som admin kan legge inn (avdeling velges når admin legger inn nytt utstyr).

C Projektplan

1. MÅL OG RAMMER

1. Bakgrunn

Dokflyt er en bedrift skapt av to moder-bedrifter: Flyt IT og Infraflyt. De gikk sammen inn for prosjektet Dokflyt-intra. Siden Dokflyt-intra har de drevet med flere løsninger for dokumentering på arbeidsplassen.

En ny løsning dokflyt skal begynne på er registrering av varer som er lånt ut internt i bedriften, og vi skal bygge prototypen for den.

Hele løsningen bygger på teknologier vi har spesialisert oss på lenge, og / eller lar oss bruke mye av det vi har lært gjennom 3 års skolegang: HTML, CSS og JS, med Angular 2 på front-end og .NET Core på back-end og den skal ligge tilgjengelig via nettleseren til Dokflyts kunder. Applikasjonen følger også Dokflyts design-språk hvor vi var nærmest oppfordret til å komme med innspill og forslag til forbedringer.

2. Oppgavebeskrivelse

Oppgaven går ut på at vi skal bygge en ny "modul" for dokflyt. En modul er en applikasjon som eksisterer for å gjøre hverdagen enklere for en gitt målgruppe. Modulene må ikke nødvendigvis henge sammen. Vår modul, Dokflyt Utstyr, skal være en selvstendig webapp, og målgruppen for den skal være byggearbeidere som jobber ute i felten.

Dokflyt Utstyr-modulen skal dekke følgende tre behov:

- Oversikt over utstyr man eier, og hvor det befinner seg.
- Mulighet for å kunne reservere utstyret, og ha en logg på hvor utstyret har vært.
- En enkel løsning for å sørge for vedlikehold av utstyret, og melde feil på disse.

Applikasjonen skal ha en meny bestående av tre deler:

- En oversikt – tabellvisning over alle varene.
- Et kart – kartvisning over alle varene.
- En kalender – varene vises i sammenheng med en kalender.

I tillegg vises det 2 stk tabeller: én som viser alle brukerne i et firma og én som viser alle aktive prosjekter til firmaet.

3. Prosjektmål

Effektmål

I løpet av første kvartal 2018 skal applikasjonen fullstendig erstatte den eksisterende løsningen.
Det skal ikke lenger være nødvendig å ha notatbøker for varene som skal lånes ut, som også vil medføre at folk ikke vil måtte ringe hverandre for å sjekke om opplysningene i notatbøkene stemmer.
Varer med feil skal kunne fikses raskere, og feil vil kunne rapporteres raskere enn før.

Resultatmål

Utvikle en webapplikasjon som skal gi arbeiderne bedre oversikt over reservasjoner og effektivisere prosessen med å reservere utstyr til prosjekter. Tidsramme: 3-4 måneder fra oppstartsdato.
Gjøre applikasjonen brukervennlig gjennom gjentatt brukertesting.
Inkludere et system for rapportering av feil eller mangler hos varene.

4. Rammer og avgrensning

Dokflyt har gitt oss muligheter til å velge hvor vi vil sette rammene. De ønsker en del funksjoner, men ønsker ikke sette begrensninger om det er enten for lite eller for mye.

Intervjuobjektene våre er de som jobber i felt og bruker dokflyt. Da får vi mest mulig igjen for intervjuene da det er akkurat samme gruppe vi lager produktet for.

2. PROSJEKTORGANISERING

1. Ansvarsforhold og roller

Basisansvar og roller, men også mer spesifikke punkter på hver av deltakerne av bacheloren.

Gruppeleder - Toni Sucic

- Scrum master & scrum team
- Ansvar for backenddelen av prosjektet
- Ansvar for visse deler av implementasjonen til frontenden
- Ansvar for å sjekke tidsloggen ved ukeslutt og bekrefte at alle har jobbet nok

Utvikler - Geir Marius Kirkeberg Jansson

- Scrum team

- Ansvar for frontenddelen av prosjektet
- Ansvar for designprosessen
- Booker rom når nødvendig

Oppdragsgiver - Dokflyt

- Scrum product owner
- Ansvar for intervjuobjekter

2. Rutiner og regler i gruppa

For å garantere kontinuerlig produktivitet er det viktig at det er klart definerte rammer for rutiner og regler som gruppemedlemmene må følge.

Følgende rutiner skal følges:

Regelmessige fysiske oppmøter i f.eks. grupperom, biblioteket eller andre steder der man kan arbeide. Minimum 3 dager i uka, gjerne mer hvis man vil.
Logging av arbeidstid. Når man jobber med en oppgave som er definert i Gantt-diagrammet må man passe på å notere hvor mange timer man har jobbet.

Følgende regler skal følges:

Som et minimum må hvert gruppemedlem legge inn 7 timer arbeid daglig minst 5 ganger i uka (4 ganger i uka hvis man fremdeles er utplassert i bedrift). Ved slutten av uka vil gruppelederen undersøke om timeloggen stemmer overens med antall timer man skal jobbe. Hvert gruppemedlem må møte opp på campus klokken 9 om morgenen senest og bli der til de 7 timene er unnagjort (dette må gjøres minimum 3 dager i uka). Man jobber altså fra kl 9.00 til kl 16.00.

Å opprettholde en normal døgnrytme er viktig for produktiviteten.

3. PLANLEGGING, OPPFØLGING OG RAPPORTERING

1. Valg av utviklingsmodell

Vi har valgt scrum som utviklingsmodell selv om vi bare er to personer. Vi kom frem til at scrum kan tilpasses to personer og fortsatt beholde den effektiviteten agile development har. Vi har arbeidet med scrum flere ganger før og syntes det fungerer godt, selv i på-kanten små team.

Da vi er to personer for vi ikke noe fulltids scrum master, men det har vi funnet ut at ikke nødvendigvis er nødvendig. Begge to skal utvikle løsningen i like stor grad, da scrum master ikke er eneste ekstra ansvar vi trenger i prosjektet.

Sprintene våre er på 2 uker per og dekker en cycle med både koding og testing, samt evaluering før neste sprint.

2. Prosjektfaser med aktiviteter

Vi har delt prosjektet opp i en rekke faser som prosjektet kommer til å bevege seg gjennom i løpet av prosjektutførelsen.

Analyse

Den første fasen handler om å analysere i hvilken sammenheng vi skal bygge applikasjonen. Vi må samle data og lære om brukerne våre slik at vi kan danne oss et bilde av hva brukerne våre faktisk trenger. Som regel holder det ikke bare å få en kravspesifikasjon. Man må gjerne også bli kjent med brukerne slik at man vet hva de egentlig vil ha. Informasjonen vi trenger kan tilegnes gjennom brukerintervju. Etter intervjuene kan vi kartlegge behovene til brukerne, og derfra kan vi utforme personas og scenarier.

Prototyping

Neste fase handler om å lage lo-fi (low fidelity) prototyper. Vi begynner med å tegne noen skisser for hånd basert på opplysningene vi tilegnet oss gjennom analysefasen. Når vi har noen skisser vi er fornøyde med kan vi gå videre til å lage wireframes basert på dem. Disse vil utgjøre grunnlaget for sprintene, som er neste fase.

Sprint 1 til 3

Den siste fasen består av tre sprinter. I løpet av en sprint skal vi utvikle en iterasjon av applikasjonen. Ved sprintens slutt skal vi gjøre brukertesting og bruke tilbakemeldingene som grunnlag for nye arbeidsoppgaver i neste sprint.

3. Metoder

Intervjumetodikk

Intervjuene våre skal hovedsakelig være kvalitative. Vi ønsker å stille spørsmål til de fremtidige brukerne for å finne ut av om det er ting de gjerne ønsker seg som ikke er blitt nevnt i kravspesifikasjonen, og vi ønsker helst ikke å intervjue mange personer, da vi mener at det er unødvendig for en applikasjon som dette.

Personas og scenarier

Etter at vi har dannet oss et bilde av hvem den typiske brukeren er gjennom intervjuprosessen kan vi begynne med utforming av personas og scenarier. En persona skal gjenspeile en typisk bruker av applikasjonen, f.eks. en byggearbeider som jobber ute i felten. Med personas kan vi skildre de viktigste egenskapene til brukerne, og denne informasjonen kan brukes til å ta de rette avgjørelsene i design og utviklingsfasene. Scenariene skal være fiktive situasjoner og hendelser som utspiller seg i felten. De skal gi oss et innblikk i hverdagen til brukerne, og gi oss en forståelse for hvilke behov de har.

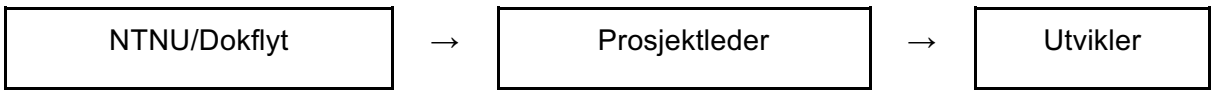
Skisser og wireframes

Før vi kan begynne med programmeringen og high fidelity prototypene må vi lage noen low fidelity prototyper, altså skisser og wireframes. Hver av oss lager flere skisser slik at vi har flere alternativer å velge mellom. Når vi blir fornøyde med skissene kan vi bevege oss over på å lage wireframes, som vil være mer tidkrevende. Wireframesene vil være mer strukturerte, da de i utgangspunktet nesten skal beskrive det eksakte oppsettet til applikasjonen vi skal implementere senere.

Prototyper

Til slutt lager vi high fidelity prototyper basert på low fidelity prototypene vi lagde tidligere. I det første sprintet vil vi fokusere på å bygge en fungerende løsning. Når den er ferdig vil vi ha en prototype som kan brukertestes. Tilbakemeldingene fra brukertestingen vil brukes til å forbedre applikasjonen i neste sprint.

4. Organisasjonskart



Prosjektet er styrt av veileder fra NTNU og prosjektleder fra Dokflyt. Veilederen fra NTNU hjelper til med at prosjektet blir styrt i riktig retning og vi kan søke hjelp underveis når vi trenger det. Dokflyt står for oppgaven og hjelper til med å skaffe oss relevant data for å komme i mål.

Under disse går det rett til prosjektleder som passer på at timer blir fulgt opp og at alt arbeid er gjort innen frister.

Under prosjektleder finner vi utvikler 2/2. Prosjektleder og utvikler jobber på samme nivå gjennom hele prosjektet, men med ulikt ansvar.

4. ORGANISERING AV KVALITETSSIKRING

1. Dokumentasjon (organisering og lagring)

Google Drive

Vi bruker Google Drive som vårt delte lagringsmedium. Her kommer alt av dokumenter fra notater til rapporter ligge. Det gjør det veldig lett for alle parter og lett ta en titt på hvordan vi ligger an. Google Drive bruker vi også som skriveverktøy da det lar oss jobbe sammen i sanntid fra hver vår PC.

GitHub

Selve kildekoden til prosjektet og instrukser på hvordan det skal bli satt opp vil ligge på GitHub. Det gir oss flotte verktøy til version control samt en god oversikt over hvordan kodedelen av prosjektet ligger an.

GitHub er også vårt valg av verktøy for å håndtere oppgaver. Vi vil bruke github issues i kombinasjon med github projects for å opprette en backlog vi kan jobbe fra. Med issues kan vi dele ut oppgavene til en eller flere person, og også kommentere eller legge til notater.

TeamGantt

For å logge arbeidstimer bruker vi TeamGantts prosjektverktøy hvor vi følger et ganttskjema lagt opp for hele prosessen. Det gjør det lett for oss å holde hverandre på plass og motiverer oss til å følge oppsatt arbeidsplan.

2. Risikoanalyse

Hva kan gå galt?

Serveren kan svikte, som medfører at applikasjonen ikke vil fungere lenger.

- **Sannsynlighet**: Lav. Noen ganger går servere offline som følge av strømbrudd eller tekniske problemer. Disse forekommer som regel ganske sjelden.
- **Konsekvenser**: Alvorlig. Dersom serveren går offline vil det medføre at applikasjonen ikke vil være tilgjengelig i det hele tatt inntil den kommer online igjen.
- **Forebyggende tiltak:** Velge en hosting-provider som har få problemer.

En bug kan medføre at brukeren ikke kan få brukt applikasjonen.

- **Sannsynlighet**: Lav. En bug som er så alvorlig at brukeren ikke vil kunne bruke applikasjonen vil mest sannsynlig ikke inntreffe.
- **Konsekvenser**: Alvorlig. Hvis den mot formodning skulle inntreffe vil konsekvensen være at brukeren ikke vil kunne bruke applikasjonen.
- **Forebyggende tiltak**: Skrive unit-tests for å fange opp bugs tidlig. Gjøre brukertesting regelmessig slik at brukerne kan rapportere bugs.

Et hackerangrep kan medføre at sensitive opplysninger kommer på avveie.

- **Sannsynlighet**: Veldig lav. Applikasjoner med dårlig sikkerhet er mer utsatte for hackerangrep, og vår applikasjon skal kjøre en ny teknologi med gode sikkerhetsmekanismer. Utviklerne vil også ta hensyn til sikkerhet. Dessuten er det usannsynlig statistisk sett at en hacker skulle ønske å hacke en tjeneste som dette.
- **Konsekvenser**: Liten. Applikasjonen kommer ikke til å inneholde data som kommer til å være av mye verdi. Alle passord vil være godt sikret (salt + hash), og de gjenstående opplysningene er ikke noe særlig sensitive.
- **Forebyggende tiltak**: Pen-teste applikasjonen for å se etter svakheter.

En bruker vil ikke kunne bruke applikasjonen pga. uegnet enhet.

- **Sannsynlighet**: Lav. Selv om brukerne våre kommer til å være i aldersgruppen 40-50 så har så og si alle av dem en smarttelefon eller tablet, og siden det er 2017 i skrivende stund kan man regne med at enhetene de eier vil være egnede.
- **Konsekvenser**: Alvorlig. Manglende enhet betyr ingen tilgang på applikasjonen.
- **Forebyggende tiltak**: Melde fra til brukerne om at applikasjonen vår forutsetter at man eier en smarttelefon (eller et nettbrett / en PC).

En bruker vil kunne slite med å forstå seg på hvordan applikasjonen skal brukes.

- **Sannsynlighet**: Moderat. I og med at brukerne kommer til å være litt eldre individer som ikke er noe spesielt IT-kyndige så kan man regne med at noen av dem vil slite litt med å bruke applikasjonen.
- **Konsekvenser**: Moderat. Dette kan medføre at brukeren ikke får noe utbytte av å bruke applikasjonen.
- **Forebyggende tiltak**: Bruke velkjente designmønstre slik at brukeren intuitivt kan finne ut av hvordan applikasjonen navigeres. Ta hensyn til ergonomi. Gjøre brukertesting og bruke tilbakemeldingene til å gjøre applikasjonen mer brukervennlig.

3. Kommunikasjonsverktøy

Facebook messenger

Valg av chat-program faller naturlig på messenger. Hadde det vært noe størrelse på gruppen hadde vi definitivt valgt slack eller noe lignende, men da vi er to følte vi slack ble for mye. Det vi trenger fra en chat program er muligheten til å enkelt sende meldinger og diskutere. Trenger vi sende kodesnutter blir det enkelt sent med en link til jsfiddle / github gist etc.

Skype

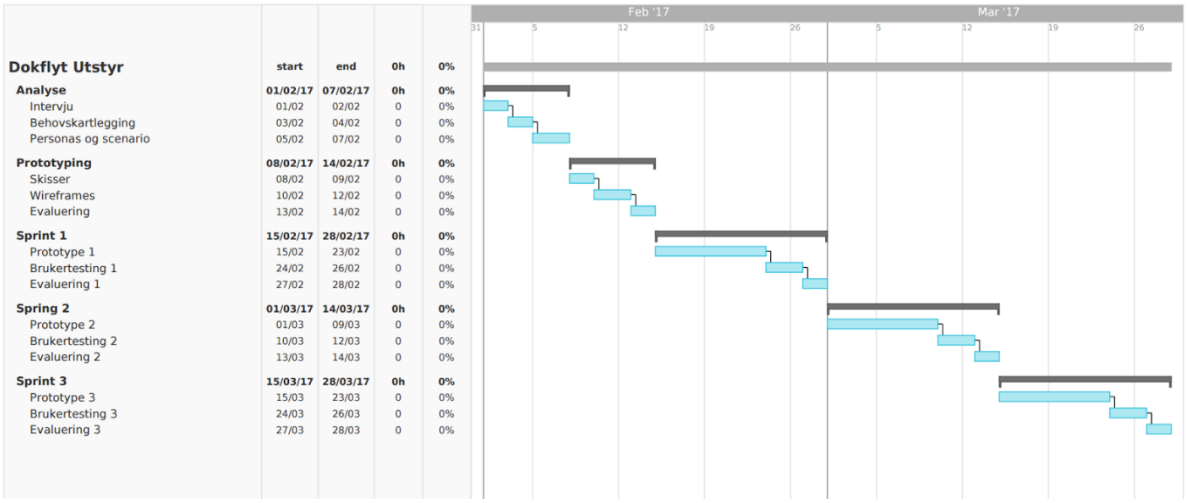
For møter bruker vi gjerne Skype siden det er et greit verktøy for videochat som de fleste er kjent med.

Slack

For tekstbasert chat med prosjekteier er vi invitert som gjest i Dokflyt sin egen slack. Vi vil bruke denne til å sende spørsmål utenfor møter og planlegging av skype-møter.

5. PLAN FOR GJENNOMFØRING

1. Gantt-skjema



Dokflyt Utstyr	start	end	0h	0%
Analyse	01/02/17	07/02/17	0h	0%
Intervju	01/02	02/02	0	0%
Behovskartlegging	03/02	04/02	0	0%
Personas og scenario	05/02	07/02	0	0%
Prototyping	08/02/17	14/02/17	0h	0%
Skisser	08/02	09/02	0	0%
Wireframes	10/02	12/02	0	0%
Evaluering	13/02	14/02	0	0%
Sprint 1	15/02/17	28/02/17	0h	0%
Prototype 1	15/02	23/02	0	0%
Brukertesting 1	24/02	26/02	0	0%
Evaluering 1	27/02	28/02	0	0%
Spring 2	01/03/17	14/03/17	0h	0%
Prototype 2	01/03	09/03	0	0%
Brukertesting 2	10/03	12/03	0	0%
Evaluering 2	13/03	14/03	0	0%
Sprint 3	15/03/17	28/03/17	0h	0%
Prototype 3	15/03	23/03	0	0%
Brukertesting 3	24/03	26/03	0	0%
Evaluering 3	27/03	28/03	0	0%

Vi har et gantt-skjema for å gi oss en oversikt over fasene til prosjektet og hvor i tidsforløpet vi er. Hver fase er illustrert med en horisontal søyle som strekker seg mellom datoene som utgjør perioden for nevnte søyle. En søyle består av oppgaver. Noen av oppgavene avhenger av at andre oppgaver er blitt gjort. Man kan f.eks. ikke begynne å lage wireframes uten å ha laget skisser først. Disse avhengighetene er illustrerte med sorte streker mellom søylene.

2. Milepæler

Gjennom prosjektet skal vi igjennom flere milepæler med et klart mål.

1: Ferdigstille behovskrav

- Blir ferdig med intervju for behovsplanlegging
- Kartlegge alle behov
- Skrive ferdige alle personas med scenarioer

2: Komplett design

- Legge opp alle skisser
- Komme frem til et sett med wireframes vi skal bygge på
- Tegne opp prototypen vi skal gjøre interaktiv

3: Klart produkt

- Fullføre alle sprint sykluser
- Finpusse koden

4: Ferdig rapport

- Skrive ferdig rapport
- Se over etter feil og svake deler
- Sette opp referanser

3. Tids- og ressursplan

Aktivitet	Tid	Kommentar
Intervju	3t	Intervjue brukere for å lære mer om behovene deres.
Behovskartlegging	7t	Bruke opplysningene fra intervjuene til å kartlegge behovene til brukerne.
Personas og scenarier	7t	Lage personas og scenarier som gjenspeiler brukerne og behovene deres.
Skisser	5t	Tegne skisser av applikasjonen.
Wireframes	14t	Lage wireframes basert på skissene.
Prototype 1	50t	Utvikle den første prototypen.
Brukertesting 1	5t	Teste prototypen på brukerne.
Prototype 2	50t	Utvikle den andre prototypen.
Brukertesting 2	5t	Teste prototypen på brukerne.
Prototype 3	50t	Utvikle den tredje og siste prototypen.
Brukertesting 3	5t	Gjøre den siste brukertestingen.
SUM	201t	Alle timene sammenlagt.