Norwegian University of
Science and Technology

Budsjettapplikasjon

Forfattere

Per Arne Drevland
Kristian Dragerengen
Per-Kristian Nilsen
Jardar Tøn

Bachelor i programvareutvikling
20 ECTS
Avdeling for informatikk og medieteknikk
Norges teknisk-naturvitenskapelige universitet

18.05.2016

Veileder Tom Røise

Sammendrag av Bacheloroppgaven

Tittel:	**Budsjettapplikasjon**
Dato:	18.05.2016
Deltakere:	Per Arne Drevland Kristian Dragerengen Per-Kristian Nilsen Jardar Tøn
Veiledere:	Tom Røise
Oppdragsgiver:	Totens Sparebank
Kontaktperson:	Dag Einar Steinsåker
Nøkkelord:	Budsjett, Mobil, Hybrid, Ionic, AngularJS, NodeJS, JavaScript
Antall sider:	158
Antall vedlegg:	10
Tilgjengelighet:	Åpen

Sammendrag:

Mange unge mennesker i Norge har problemer med sin privatøkonomi. På oppdrag fra Totens Sparebank har vi utviklet en mobil budsjettapplikasjon for å bidra med en løsning til dette problemet. Denne rapporten beskriver prosjektets utviklingsprosess og arkitektur. Applikasjonen kan generere et ferdig oppsatt budsjett som kan endres etter behov, eller et som kan settes opp på egenhånd. Budsjettet kan deles med andre brukere, slik at familier, lag, grupper eller mindre organisasjoner kan benytte seg av budsjettet. Applikasjonen kan også benyttes av samme bruker over flere enheter. Brukerne får også mulighet til å sette opp sparemål. Hovedfokus med dette prosjektet var å være innovative ved å bruke nyere rammeverk for å muliggjøre en god felles løsning som fungerer på kryss av flere plattformer.

Summary of Graduate Project

Title:	**Budsjettapplikasjon**
Date:	18.05.2016
Authors:	Per Arne Drevland Kristian Dragerengen Per-Kristian Nilsen Jardar Tøn
Supervisor:	Tom Røise
Employer:	Totens Sparebank
Contact Person:	Dag Einar Steinsåker
Keywords:	Budget, Mobile, Hybrid, Ionic, AngularJS, NodeJS, JavaScript
Pages:	158
Attachments:	10
Availability:	Open

Abstract:

Many young people in Norway have trouble keeping control over their personal finances. As a contribution to solving this problem, we have developed a mobile budget application commissioned by Totens Sparebank. This bachelor's thesis describes the architecture and development process of the project. The application provides the user with an automatically generated budget. They may also choose to customize it to suit their needs, or create their own budget from scratch. The fact that the budget can be shared with other users makes it possible for families, teams, groups or smaller organizations to make use of a common budget. The application can also be used by a single user over several mobile devices. There's also functionality for keeping track of savings. The main focus of the project was being innovative through the use of modern frameworks, allowing the development of well structured multi-platform solution.

Forord

Vår oppdragsgiver:

Totens Sparebank er en sparebank i Eika gruppen. De har 9 kontorer (Lena, Skreia, Raufoss, Eina, Gjøvik, Hamar, Eidsvoll, Råholt, Feiring) i mjøsregionen og har tilsammen 130 ansatte (per 25.04.2016). Våre kontaktpersoner i Totens Sparebank er Dag Einar Steinsåker og Geir Sindre von Schantz Nyborg.

Takk til:

Vi ønsker å takke alle som har vært delaktiv i dette prosjektet. Takk til Dag Einar Steinsåker og Geir Sindre von Scahntz Nyborg for selve oppgaven. Takk til Tom Røise som har vært veilder under dette prosjektet. Takk til Eivind Arnstein Johansen og alle som har bidratt under brukertesting for hjelp til utbedringer av applikasjonens brukergrensesnitt. Takk til Simon McCallum for veiledning innen kodekvalitet. Takk til Intrum Justitia for å dele rapporten Consumer Payment Report 2015 med oss (vedlegg G).

Innhold

Figurer

Tabeller

Kodeutsnitt

1 Introduksjon

1.1 Omfang

Intrium Justitia la i 2015 ut en rapport (vedlegg G) om norske forbrukeres hverdag. De ser på hvilke utgifter og hvilken evne de har til å håndtere egen månedlig husholdningsøkonomi. Der trekker de frem som et av hovedfunnene at *31 prosent sier at de noen ganger har problemer med å få pengene til å strekke til. Blant respondentene i aldersgruppen 24 år eller yngre, svarer hele 41 prosent at dette er et problem. I samme aldersgruppe er 26 prosent enig i påstanden om at de ikke har penger nok til en anstendig tilværelse.*

Videre i rapporten sier 18,5% av de spurte mellom 18-34 år at de har lånt penger til andre formål en bolig, hvor mesteparten har lånt penger fra familie, mot 6,5% av de spurte over 34 år. 39,5% under 34 år melder at årsaker for å ikke betale regninger i tide er manglende evne, mot 29,5% for de over 34 år. 46,5% under 34 år kan betale en uforutsett regning på 26 000 kroner mens 61,5% av de spurte over 34år hadde den evnen.

Tendensen i rapporten gir et inntrykk av at unge mennesker i Norge ikke gir hverdagsøkonomien nok oppmerksomhet. De har dårligere evne til å betale, er mindre opptatt av å lære økonomi til sine barn og har mer bekymringer angående sin egen økonomi.

NRK melder i 2015 at ungdom mellom 18 og 26 år har til sammen 1 milliard norske kroner i kredittkortgjeld [1]. De gir foreldrene skylden for å ikke ha lært ungdommen nok om privatøkonomi. Jan Græsvik sier til nettavisen.no at han ønsker at privatøkonomi skal bli en del av skoleverket [2]. Jesper Foss sier til nrk.no at det er spesielt småforbruket som de unge ofte har problemer med [3]. *Hvis du kjøper en kaffe om dagen, 45 kroner, er ikke det i seg selv mye penger. I løpet av ei skoleuke utgjør det 225 kroner. I løpet av et helt år er det snakk om 11.700 kroner.* Han ser på god norsk økonomi som årsak til dette. *Norge er jo i en situasjon der økonomien har vært veldig bra lenge. Pengene har sittet litt løst, og har en lagt seg til slike forbruksvaner må man jobbe litt for å snu det igjen*

Det er tydelig at norsk privatøkonomi er i forfall. Stor økonomisk vekst og betalingsevne har gitt mange dårlige vaner blant norske forbrukere og tiltak trengs for å snu en stadig dårligere trend.

1.2 Oppgave

Oppgaven baserer seg på å hjelpe brukere til å kontrollere sin økonomi ved hjelp av en applikasjon til mobile enheter, uavhengig av om de er kunde i banken eller ikke. Oppdragsgiver vil ha en slik applikasjon da dagens applikasjoner er vanskelige å bruke eller har ikke de funksjonalitetene de ønsket. Oppdragsgiver ønsker at applikasjonen skal være tilgjengelig på flere plattformer for å nå ut til så mange som mulig. Applikasjonen skal kunne brukes til å sette opp både enkle og mer avanserte budsjetter. Utgifter og inntekter skal kunne bli lagt til av brukeren fortløpende og sammenlignes med det oppsatte budsjettet.

Brukeren skal kunne sette seg sparemål. Slik at brukeren enkelt kan holde og følge med på hvor mye som spares til ulike formål. Disse sparemålene vil fordeles utover perioden fram mot målet slik at brukeren opplever et stort sparemål som oppnåelig siden det ikke krever mye sparing hver dag. En graf som viser oversikt over sparingen vil gi brukeren en statistikk på hvordan sparingen har gått bakover i tid. Brukeren skal også kunne se oversikt over alle sine utgifter i en utgiftshistorikk og se hvordan dette er i forhold til det oppsatte budsjettet. Oversikt over faste utgifter og inntekter skal også vises. Oppdragsgiver ønsker at applikasjonen skal utformes på en slik måte at det vil være lett å bruke og enkel å forstå, for å motivere brukere til å holde oversikt over sin økonomi.

Applikasjonen skal også inkludere funksjonalitet utover enkeltmannsbruk, slik at familier, grupper eller mindre organisasjoner kan føre et budsjett sammen. Dette skal fungere på en slik måte at all endring av en budsjettpost skal opptre samtidig mellom de aktuelle enhetene. For enheter som er utilgjengelig for øyeblikkelig oppdatering, skal det være mekanismer som sørger for å oppdatere endringer når de enhetene er tilgjengelige igjen. Brukere skal kunne bruke sentral lagring og dermed aksessere sine data på flere enheter. For dette vil det kreves oppretting av brukerkonto for applikasjonen slik at brukere kan autentisere seg. Applikasjonen skal også kunne kjenne igjen brukere, slik at autentisering ikke trenger å bli gjort hver gang applikasjonen er i bruk. Det må også utvikles sikkerhetsmekanismer slik at brukerinformasjon ikke kommer på avveie.

Applikasjonen skal også gi brukeren rask mulighet til å få kontaktinformasjon til banken. Herunder hjemmesiden, kundeservice til banken på telefon og oversikt over kontorer i et kart.

Oppgaven splittes opp i en prioritert liste etter ønske fra oppdragsgiver:

- Bruk for enkeltpersoner.
- Bruk for familie; kommunikasjon mellom brukere.
- Bruk for mindre organisasjoner.

Oppgavene gjennomføres etter rekkefølge nevnt ovenfor. Dette legger til rette for at sluttproduktet tilfredsstiller oppdragsgiver.

1.3 Avgrensing

Intrium Justitia setter opp 10 tips for hvordan bevare en sunn husholdningsøkonomi. Det første punktet deres er *Lag et budsjett, slik at du har kontroll på hvor mye penger du kan bruke. Få grep om husholdningsutgiftene ved å liste opp alle utgiftene du har hver måned, slik at du ikke bruker mer penger enn du har til disposisjon.*

Vår oppdragsgiver, Totens Sparebank, vil lansere en applikasjon som vil hjelpe forbrukere med å sette opp et budsjett som holder orden på privatøkonomien. Det skal være et verktøy for unge brukere for å holde orden på småkjøpene og gi et bilde av forbruket sitt.

Applikasjonen vil ikke ha tilgang til, kommunikasjon med, eller noen annen tilknytning til brukerens bankkonto. En slik tilknytning kunne vært mulig hvis kontoinformasjon hadde tilgjengelig via et API etter OAuth-innlogging, men per dags dato finnes ikke dette.

Oppgavens omfang ville økt betraktelig, trolig forbi hva som er mulig innen våre tidsrammer. Temaet er allikevel interessant, og blir videre diskutert under kapittel 12.4 Videre utvikling og utvidelse.

1.4 Målgruppe

Ved å lage en applikasjon til mobile plattformer ønsket Totens Sparebank å nå ut til en yngre brukergruppe. De hadde sett for seg en kjernebase med brukere på mellom 15-35 år. Dette skulle ikke være noen aldersgrense, verken øvre eller nedre, og oppdragsgiver ønsket også at applikasjonen skulle være tilpasset alle brukere i alle aldersgrupper.

Rapportens målgruppe er alle parter direkte involvert i prosjektet, eventuelle videreutviklere, informatikkstudenter og fremtidige arbeidsgivere.

1.5 Formål

Behovet for en slik applikasjon ser vi på som stor. Kapittel 1.1 Omfang ga oss innblikk i situasjonen i norsk personøkonomi og vi så at trenden er i økende grad negativ. Det er absolutt et rom i dagens marked for en budsjettapplikasjone som er enkel og rask å bruke.

Oppgaven gir store utfordringer

For å tilfredsstille oppdragsgivers krav om å utvikle for flere plattformer, må vi ta stilling til om applikasjonen skal utvikles native for alle tre plattformer, som en webapplikasjon som kjører i telefonens browser, eller om vi skal utvikle en hybrid versjon.

Det er kritisk at vi er strenge når det kommer til kravene våre når det gjelder brukervennlighet. Gjeldende applikasjoner i dag tilbyr mye av den funksjonaliteten vi planlegger å tilby, men mangler mye når det kommer til brukeropplevelse. Mange applikasjoner føles tunge og vanskelig å bruke. På dette området er vi nødt til å skille oss ut i mengden. Vi må balansere konformiteter og innovasjon hvis applikasjonen skal hevde seg i dagens marked.

Delingsmekanismer mellom brukere vil bli svært utfordrende. Vi blir avhengig av en god struktur på serversiden av systemet og vi må være kritiske til vår design av database. I tillegg må tjenesten mellom server og klient fylle dagens krav til sikkerhet.

1.6 Rammer

Etter oppstartsmøte i begynnelsen av januar 2016 ble gruppen enige om arbeidsmetoder og grupperegler som skulle være gjeldende gjennom hele prosjektperioden. Disse grunnprinsippene gjelder for det meste metodikk vi skulle følge i vår utvikling av applikasjonen og inkluderer, men begrenses ikke til:

- Bruk av Versjonskontroll
- Kvalitetssikring (QA)
- Dokumentasjon
- Testing

I tillegg til disse utviklingsmetodikkene, ble gruppen enig om hvordan arbeidsmetodikk vi skulle følge. Dette gjelder hvor og hvordan vi skulle jobbe og inkluderer, men begrenses ikke til:

- Arbeidstider som var fra klokken 08.00 til 16.00 tirsdag-torsdag. Mandager fra 10.00 til 16.00. Gruppen ble enig om å jobbe sammen på skolen i disse periodene.
- Daily stand up meeting, et kort møte som oppsummerte arbeid gjort siden sist. I dette møtet skulle også eventuelle problemer/beslutningspunkter tas opp. Disse beslutningspunktene omfattet gjerne prinsipielle utforminger på applikasjonen. Tyngre beslutningspunkter ble tatt opp i retrospektive møter vi hadde etter hver milepæl.
- Milepælsbaserte iterasjoner. Dette ville gjøre utviklingsprosjektet vårt mer fleksibelt når det kommer til timeboxing. Siden vår erfaring med å estimere prosjekter var begrenset, mente vi at dette ville være med på å ikke la tidspress bli en faktor for utviklingen.
- Retrospective meeting, et større møte etter hver fullførte milepæl. På dette møtet tar vi opp hvordan vi følte utviklingen gikk, hva som kunne forbedres, hva som gikk bra og planlegging av neste milepæl.
- Ukentlig møte med veileder. Dette ville bidra til at vi tok et steg tilbake fra vår utviklingshverdag og få et mer helhetlig bilde av prosjektet

1.7 Gruppens bakgrunn

Alle i prosjektgruppen har studert programvareutvikling på fulltid og hatt samme emner og kompetanseområder under utdanningsløpet. Alle bærer kompetanse innenfor informatikk med fordypning innen programmering og systemutvikling. Gjennom fastsatte emner til utdanningen og valgfag har alle tilegnet seg følgende kunnskap som har vært spesielt nyttig for dette prosjektet:

- Webutvikling. Innebærer bruk av HTML, CSS og JavaScript som har gitt en grunnleggende forståelse av hvordan nettsider blir laget.
- Utvikling for mobile enheter. Har gitt en introduksjon til hvordan programmering må tilpasses mobile enheter.
- Ergonomi i digitale medier. Dette har gitt en forståelse av hvordan brukere samhandler med brukergrensesnitt.

Jardar, Per-Kristian og Kristian har tidligere arbeidet som gruppe i flere emner gjennomgående i hele studiet.

1.8 Tidligere arbeid

Et utviklingsprosjekt for deler av denne oppgaven er allerede blitt utført i emnet IMT3672 Mobile Development Project hvor en uferdig prototype ble utviklet til bruk for Android. Det ble parallelt utført et systemutviklingsarbeid for denne prototypen i emnet IMT3102 Objektorientert systemutvikling.

Tidligere utført arbeid har gitt en pekepinn på hvordan problemer i oppgaven kan løses og hvordan utseende på brukergrensenittet kan se ut. For prototypen som ble utviklet er det ikke gjenbrukt noe kildekode eller funksjonalitet. Dette på grunn av omstrukturering av store deler av applikasjonen som gjorde det lite lønnsomt å ta i bruk allerede eksisterende kode. For dokumentasjon rundt prosessen som inngår i systemutvikling er noe gjenbrukt. Der er det ingen helt like deler av som er å finne igjen, men enkelte deler er tatt med og blitt forbedret. Dette gjelder blant annet use case og backlog. Når det gjelder oppbygning av brukergrensesnittet er dette relativt likt som tidligere prototype. Hvis de to applikasjonene sammenliknes, er det lett å se at den ene er basert på den andre. Her ble det gjenbrukt mye innen form og farge, samt metode for å represen-

tere informasjon til brukeren.

Den tidligere prototypen har også mer preg av å være en 'spareapplikasjon' hvor brukeren kan registrere sine utgifter og inntekter for å holde oversikt over disse uten noen form for budsjett.

1.9 Roller

Under prosjektets planleggingsfase ble det tatt en avgjørelse om at rollene som ble tildelt ikke skulle være bindende. Grunnen til dette var at medlemmene skulle få prøve seg på forskjellige områder av applikasjonen og finne ut hva de ønsket å fokusere på. Dette førte til at medlemmene falt naturlig inn i sine roller gjennom utviklingsfasen. Mange deler av applikasjonen er utviklet under samarbeid mellom to gruppemedlemmer. Dette har ført til at medlemmer har delt på enkelte roller.

Ansvarsområder som klart skilte seg ut på de forskjellige medlemmene i gruppen var:

- **Kristian** Gruppens leder. Hadde ansvaret for å styre gruppen, sørge for at det var fokus på arbeidet og å sette retningslinjer. Han var også gruppens kontaktpunkt opp mot oppdragsgiver. I tillegg sørget han for at prosjektets administrative oppgaver ble gjort, som for eksempel å forberede gruppen på kommende møter. Han hadde ansvaret for sparemålsmodulen, brukertesting og utbedringer av brukergrensesnittet med dets funksjonalitet.

- **Per Arne** var gruppens nestleder. Han hadde ansvaret for at vedtak gruppen gjorde ble fulgt opp. Tok ansvaret for generelle varige endringer i applikasjonens stilsett, som for eksempel konvertering av ikoner som ble gitt av oppdragsgiver og stilsetting av farger. Hadde ansvaret for budsjett- og home- modulen og model-implementering, herunder pålogging av bruker med forskjellige autentiseringstjenester.

- **Per-Kristian** Var gruppens kontaktpunkt for veileder. Hadde ansvar for kommunikasjon og forberedelser til møte med veileder, herunder også skrive møtereferat. Tok ansvaret for kvalitetssikring og tester. Hadde ansvaret for transactionmodulen, gjentagende transaksjonstjeneste, og var ansvarlig for å kvalitetssikre brukeropplevelsen.

- **Jardar** Hadde ansvaret for å sette opp prosjektmiljøet med de nødvendige rammeverkene. Hadde ansvaret for det administrative rundt prosjektet og utviklingsprosessene. Satte opp testregimer og retningslinjer for bruk av Git sammen med Per-Kristian. Tok seg av hoveddelen av alle QA-oppgavene som kom. Hadde ansvaret for utvikling av serversiden og introduksjonsmodulen.

1.10 Ordliste

Oppdragsgiver - Totens Sparebank ved Dag Einar Steinsåker og Geir Sindre von Schantz Nyborg
Transaksjoner - Fellesbetegnelse for inntekt og utgift.
MVC - Model View Controller. Design pattern brukt i applikasjonen.
Modell - Model i MVC som modellen.
Kontroller - Controller i MVC som kontroller.
View - View i MVC som view -et, -ene osv.
History-stack - En stack over sidene, i en tab, som en bruker har måtte navigert seg gjennom for

å komme til den siden han står på nå. Det er en history-stack per tab.

1.11 Dokumentets struktur

1. **Introduksjon** Avklaringer om oppgaven og prosjektet.
2. **Kravspesifikasjon:** En detaljert oversikt over alle krav som stilles til produktet og som skal inkluderes i utviklingsprosessen.
3. **Produkt:** En grunnleggende introduksjon til produktet og dets funksjonalitet.
4. **Native, hybrid eller web:** Diskusjon rundt grunnlaget for applikasjonstype.
5. **Utviklingsmodell:** Forklaring om hvordan utviklingsprosessen har blitt gjennomført. Overordnet modell for systemutvikling og gang i utviklingsløpet.
6. **Systemarkitektur:** Oversikt over hvilke komponenter systemet består av.
7. **Frontendarkitektur:** Oversikt over hvilke komponenter applikasjonen består av.
8. **Design:** Forklaring over hvilke designvalg som er foretatt, herunder grafisk- og databasedesign.
9. **Kvalitetssikring:** Forklaring av hvilke prosesser og verktøy som er benyttet for å kvalitetssikre produkter.
10. **Verktøy:** Beskrivelse av hva som er brukt for å støtte opp under utviklingen innen prosjektstyring, dokumentasjon og automatiserte løsninger.
11. **Sikkerhet:** Gjennomgang av hvilke sikkerhetstiltak som er benyttet og begrunnelse av disse.
12. **Sammendrag:** Oppsummering av prosjektet i sin helhet, evaluering av eget arbeid og videreutvikling.

2 Kravspesifikasjon

Kravspesifikasjonen innledes med overordnede krav som er utformet av oppdragsgiver. Videre kommer funksjonelle krav, krav til utforming og en PACT-analyse som dekker produktets innhold og utforming. Til slutt ligger operasjonelle krav, krav til sikkerhet og krav til utviklingen som dekker rammene rundt produktet.

2.1 Overordnede krav

- Skal kunne brukes på iOS. Det er også ønskelig for Android og Windows, men ikke et krav.
- Norsk språk. Det er også ønskelig med engelsk og nynorsk, men ikke et krav.
- Mulighet for å legge ut applikasjonen på operativsystemets applikasjonsbutikk.
- Applikasjonen skal være tilpasset en yngre aldersgruppe.

2.2 Funksjonelle krav

- Tilby forslag til forhåndsdefinert budsjett fra SIFO (Statens institutt for forbruksforskning) referansebudsjett[4]. Kilden henvises til referansebudsjett for 2016, men dette revideres årlig. På grunnlag av årlige revidering av budsjettet skal det også være mulig å endre dette i applikasjonen slik at budsjettet som tilbys alltid er oppdatert i forhold til siste versjon av SIFO referansebudsjett.

- Legge til, fjerne og endre budsjettposter, sparemål samt inntekter og utgifter i budsjettet. Alt som brukeren legger til skal dermed kunne endres eller slettes.

- Få oversikt over budsjett med transaksjoner i de forskjellige budsjettpostene. Det skal vises en totaloversikt som gir brukeren en oversikt over over de totale utgiftene i forhold til inntekt. Brukeren skal også kunne få en status i forhold til hver enkelt budsjettpost. Videre skal også brukeren kunne gå inn på en konkret budsjettpost for å se en detaljert oversikt og alle inntekter eller utgifter som er tilknyttet denne posten.

- Tilgang til bankes kundeservice og hjemmeside. Brukeren skal på en enkel måte kunne finne frem kontaktinformasjon til banken. Dette skal ligge tilgjengelig som både en lenke til hjemmesiden og et telefonnummer til kundeservice som kan ringes direkte.

- Dele budsjett med andre brukere. Det skal foreligge funksjonalitet slik at en budsjettpost kan benyttes og administreres av flere brukere samtidig. Det skal også være tilgangsstyring slik ikke hvem som helst kan få tilgang til andre brukeres budsjett.

- Ha budsjettet tilgjengelig på flere enheter via en brukerkonto. Ved å opprette en brukerkonto i applikasjonen skal budsjettinformasjon kunne videreformidles til en annen enhet hvor brukeren også er autentisert. Hvis en bruker eksempelvis bruker applikasjonen på et nettbrett en dag og på sin telefon en annen dag, skal dette fungere sømløst ved at oppdatering

av budsjettinformasjon blir gjort på begge enheter.

2.3 Krav til utforming

- Utforming i henhold retningslinjer etter designhåndbok fra Totens Sparebank. Dette innebærer bruk av bankens farger, fonter og ikoner.
  Applikasjonen profil skal være basert på bankens fargepallett som er blå og gul:

   RGB: 0 154 199

   RGB: 250 183 25

  Hovedvekt med blåfarger og bruk av gulfargen hvor dette lar seg gjøre. Banken har også 130 forskjellige ikoner som kan brukes i applikasjonen. Det er ikke et krav om at alle brukes, men de mest hensiktsmessige skal inkluderes hvor det er mulig.

  Ved valg av fonter skal Lucida Sans brukes til brødtekst og tekst av mindre størrelser. Ved større tekster skal Eika Bold brukes hvis passende.

- Utforming med hensyn til retningslinjer for universell utforming. Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger[5] er begrenset til å gjelde nettløsninger og automater. Applikasjoner inngår i denne forskriften hvis applikasjonen må laste ned oppdatert informasjon over internett for å fungere[6]. Det vil altså si at denne applikasjonen ikke inngår i forskriften. Det skal allikevel tas hensyn til universell utforming ved å ha som mål å opprettholde en A i henhold til WCAG 2.0 standarden [7].

- Viktige tall som fremkommer i applikasjonen skal fremstilles grafisk.

2.4 PACT Analyse

PACT-analysen er brukt til å definere behovet for design i applikasjonen[8]. Dette vil også fungere som krav som ble fulgt under utviklingsprosessen. Dette inkluderer krav for brukervennlighet, utforming og universell utforming.

2.4.1 People

Demografi

Applikasjonen er målrettet for en yngre brukergruppe som defineres innenfor en alder fra 15 til 35 år. Selv om en ung brukergruppe er målet skal applikasjonen også være utviklet på en slik måte at den kan brukes av eldre.

Kunnskapsnivå

Det skal ikke kreves opplæring for å kunne ta i bruk applikasjonen. Hele applikasjonen skal være selvforklarende og logisk. Fokus på å bruke operativsystemets særpreg er sentralt, slik at applikasjonen fungerer på samme måte som mobiltelefonen i seg selv gjør.

Språk

Applikasjonens språk bestemmes av språket til operativsystemet. Hvis operativsystemet er norsk, skal applikasjonen også være på norsk. For alle andre språk, skal applikasjonen være engelsk.

Fysiske egenskaper

Det skal tas hensyn til personer som enten er fargeblinde, døv eller stum. Dette skal gjøres ved å følge retningslinjer for universell utforming. Applikasjonen vil ikke bruke lyd eller tale sentralt.

Kognitive Egenskaper

Ta hensyn til brukere som kan ha konsentrasjonsproblemer, problemer med hukommelse, resonnering. Fokus på enkelhet i applikasjonen.

Hyppige og sjeldne brukere

Applikasjonen skal være utformet på en slik måte at det er irrelevant om brukeren benytter applikasjonen hyppig eller sjelden. Brukeren må ta i bruk applikasjonen for hver gang det gjøres et varekjøp, men dette er for å holde ved like applikasjonens innhold og ikke for å kunne gjenkjenne brukesmetode.

2.4.2 Activities

Sjeldne operasjoner

Det skal ikke være skille mellom operasjoner som utføres ofte eller sjeldent, disse skal fremstille på samme måte. Det skal ikke være vanskelig eller forvirrende å bruke funksjoner som ikke brukes ofte.

Avbrytelser

Avbrytelser skal håndteres utifra kontekst. Dersom brukeren er i en aktivitet hvor arbeid kan gå tapt som følge av en avbrytelse, skal applikasjonen fortsette på samme skjermbilde når bruk av applikasjonen tas opp igjen. Hvis applikasjonen avsluttes skal applikasjonen returnere til hovedbildet.

2.4.3 Context

Omgivelser

Applikasjonens fargeprofil skal brukes på en måte som gjør grensesnittet tydelig under sterke lysforhold. Dersom applikasjonen må gi brukeren en viktig beskjed vil det tas i bruk operativsystemets støtte for varslinger.

Sosial kontekst

Applikasjonen skal aldri være sjenerende for brukeren i en sosial kontekst. Dette innebærer at applikasjonen ikke skal tiltrekke seg oppmerksomhet fra andre mennesker enn brukeren.

Brukermiljø

Siden dette er en mobil applikasjon kan miljø rundt bruker ha innvirking på bruken av applikasjonen. For eksempel at en bruker er i bevegelse utendørs. En av de typiske oppgavene en bruker vil gjøre er å legge inn en utgift i forbindelse ved et kjøp. Dette gjøres av brukeren selv og krever ikke noe kommunikasjon med andre systemer eller nettverk. I hovedsak vil denne oppgaven bestå av å trykke på en knapp for å legge til utgift, skrive inn en sum og velge en budsjettpost for utgiften. Hvis bruker ønsker kan han eller hun også legge inn en beskrivelse. Dette skal ta

relativt lite tid og kan dermed utføres selv om bruker er i bevegelse. En annen oppgave som er relevant og gjøre mens brukeren er på farten er å sjekke status for budsjettet totalt og for en enkelt budsjettpost. Dette skal være lett tilgjengelig. Resten av oppgavene en bruker gjør som å legge til sparemål og administrere faste utgifter/inntekter er mest realistisk å gjøre når brukeren har satt av tid og er i roligere omgivelser.

Brukervennlighet

Oppdragsgiver ønsker stor fokus på brukervennlighet. Dette er noe som ofte kan gå på bekostning av godt design, men løsningen vil kunne tilfredsstille begge disse kravene. Dette vil gjøres ved å fremstille informasjon og valgmuligheter så enkelt som mulig. Dette innebærer å ha minimal funksjonalitet og informasjon per skjermbilde, noe som fører til at løsningen vil ha mange forskjellige skjermbilder dedikert til enkeltfunksjonalitet. Samtidig vil skjermbilder med relativt lik funksjonalitet ha lignende grafisk brukergrensesnitt, slik at brukeren kan kjenne seg igjen, selv i nye skjermbilder.

2.4.4 Technology

Nettverk

Applikasjonen vil i første omgang ikke ta i bruk internett eller andre kommunikasjonskanaler. Dette vil inkluderes ved å ta i bruk tilleggstjenester i applikasjonen som å dele budsjettposter med andre, men vil ikke være behov ved enkeltmannsbruk.

Feil

Applikasjonen skal være konstruert slik at brukerfeil ikke skal ha alvorlige følger. Designet skal forhindre at brukerfeil oppstår. Om brukeren gjør noe feil i applikasjonen, skal det tydelig komme frem hva som er feil, hvor feilen ligger samt en løsning på problemet.

Visning

Applikasjonen skal hovedsaklig utviklet for å brukes på en mobiltelefon. Her inngår ikke nettbrett. Applikasjonen skal fungere på samme måte på enheter av forskjellig størrelse. Applikasjonen trenger ikke støtte for landskaps-modus da det ikke er behov for dette.

2.5 Krav til sikkerhet

- Det skal ikke samles inn eller lagre sensitive personopplysninger i applikasjonen. Det settes ikke egne krav til personvern, da applikasjonen ikke skal benytte seg av personopplysninger ut over navn og alder.

- Brukerdata som blir lagret i applikasjonen for innlogging skal oppbevares på en sikker måte. Bruker-id kan lagres i klartekst, men passord skal alltid hashes før lagring. Minimumskrav er SHA-256 eller bruk av SHA-2 familien, da disse metodene ikke er knekt eller har større svakheter[9].

- Innloggingssesjoner som blir foretatt skal kunne bevares slik at brukeren slipper innlogging neste gang applikasjonen er i bruk. Dette skal ikke være gyldig mer enn 60 dager før brukeren må identifisere seg på nytt.

- All kommunikasjon skal utføres på en slik måte at kritisk informasjon overføres sikkert. Kommunikasjon skal alltid utføres på sikre kanaler som er kryptert.

2.6 Operasjonelle krav

- 'Offline first': Applikasjonen skal kunne fungere uten å være tilkoblet internett eller utenfor dekningsområdet til mobiloperatør. Dette skal heller ikke påvirke delingsfunksjonaliteten. Deling vil ikke fungere uten tilkobling til internett, men mekanismen kunne fortsette i det telefonen kommer på nett igjen.

- Det settes ikke noen formelle krav til oppetid for applikasjonen, dette kommer an på hvilke driftsløsninger som blir iverksatt.

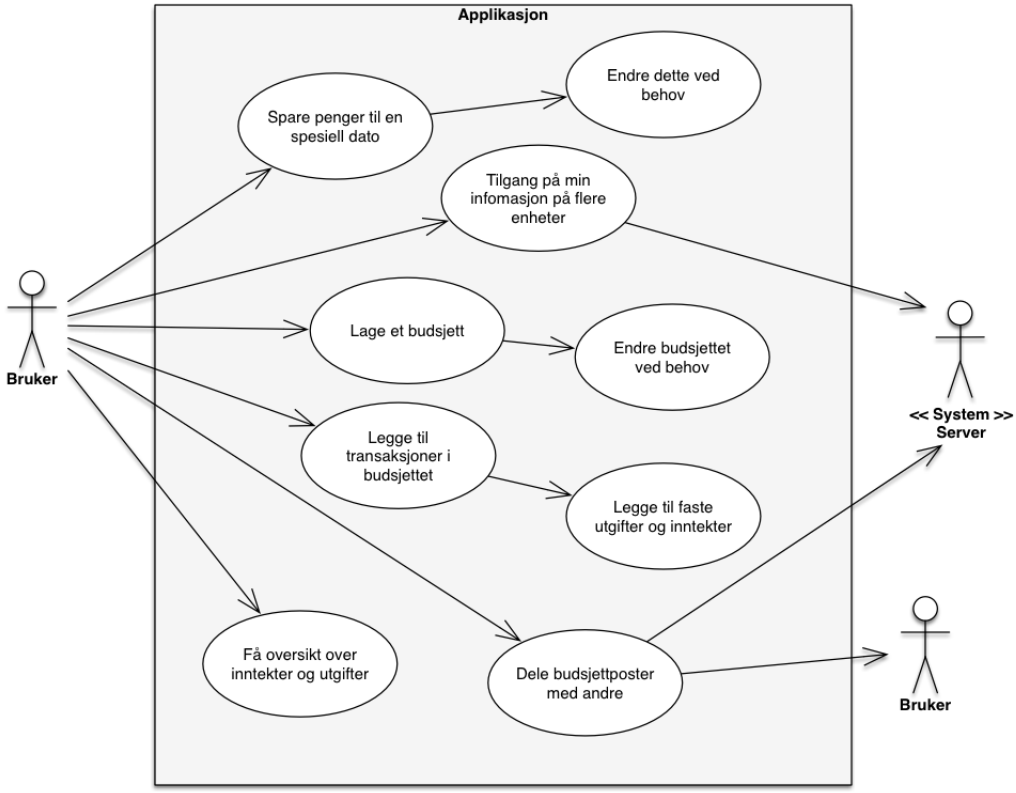
2.7 Krav til utvikling

- Applikasjonen skal utvikles på en slik måte at den skal inneholde så få feil som mulig. For å kvalitetssikre kodens kvalitet og virkemåte settes det et krav om regresjonstester for all viktig funksjonalitet, slik at eventuelle endringer et sted ikke ødelegger for annen kode som har denne som en avhengighet. Det stilles i tillegg et krav om 80% testdekning for hele kodebasen.

- Applikasjonen skal utvikles på en slik måte at det enkelt kan endres på koden for å ha muligheten til å foreta endringer i applikasjonen. Dette er for å ta hensyn til videreutvikling.

- All kode som blir skrevet skal alltid revideres av en annen person enn den som har skrevet den. Personen som reviderer koden skal kommentere feil, men også eventuelle endringer som kan gi mer effektivitet i koden. Alle endringer skal gjøres av personen som selv har skrevet koden.

3 Produkt



Figur 1: Use-case-diagram for applikasjonen

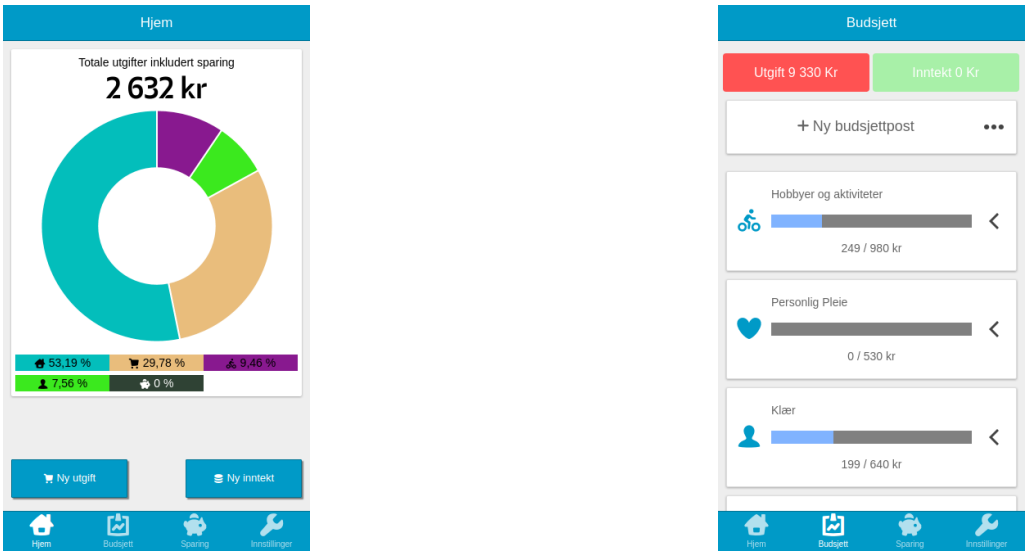
Figur 1 viser en oversikt for funksjonalitet i applikasjonen sett fra brukerens ståsted. Ved å presentere et grensesnitt med de funksjonalitetene som vil bli brukt mest som de mest fremtredene, fremstår applikasjonen som brukervennlig. Engangsfunksjonaliteter, eller funksjonaliteter som blir brukt sjeldent er plassert på steder i applikasjonen som er mindre fremtredende, men intuitivt å finne frem til. Dette gjelder for eksempel pålogging og registrering.

Ved førstegangs oppstart av applikasjonen blir brukeren presentert en introduksjon, hvor applikasjonens hovedfunksjonalitet blir forklart og fremvist. Brukeren får her et valg om å logge inn eller registrere en ny brukerkonto. Velges det å ikke opprette en brukerkonto, er det mulig å gjøre dette senere. I tillegg til dette gis brukeren et valg om å få et ferdig oppsatt budsjett, som så kan endres i ettertid.

Etter gjennomgang av introduksjon er applikasjonen klar for bruk. Brukeren kan da registrere sine inntekter og utgifter etter hvert som de forekommer, sparemål kan opprettes og budsjettet kan styres som brukeren selv ønsker. Dersom brukeren er innlogget, er det også mulighet for å

dele en eller flere budsjettposter med andre brukere. Begge parter kan da registrere inntekter eller utgifter i den samme budsjettposten.

Vi ønsket å gjøre applikasjonen så brukervennlig som mulig. Ved å bruke 'divide and conquer'-prinsippet til å gruppere funksjonalitet i ulike deler av grensesnittet, klarte vi å oppnå en struktur som er lett å forstå for brukere. Mer om dette kan leses under kapittel 9.4 Brukertesting.



(a) Hjem skjerm

(b) Budsjett skjerm

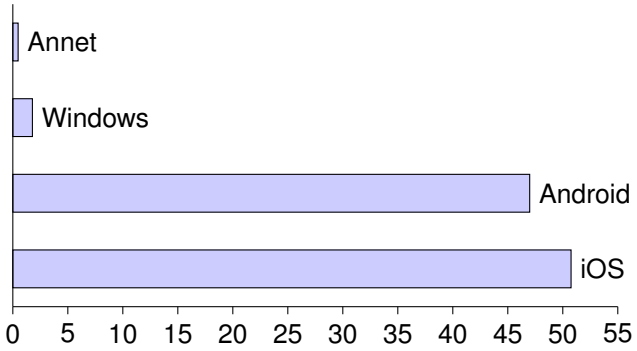
Figur 2: Applikasjonens utseende

Dette er implementert med en menystruktur som er fremvist på alle skjermer slik at brukeren alltid har mulighet til å gå direkte til andre oppgaver. Dybden i de forskjellige skjermene er begrenset til maksimalt tre nivåer med viktig og mer brukt funksjonalitet først. Dette er i tråd med hva Alan Cooper skriver i sin bok About Face[10] om dype lagdelinger: *"Tidligere menyer var hierarkiske, men abstrakte hierarkier gjør det vanskelig for brukere å navigere igjennom ... "*. Han underbygger dette med *"... hierarkier er ikke et naturlig konsept for de fleste mennesker ... Programmerere er veldig komfortable med nesting av systemer ... Andre mennesker finner dette konseptet vanskelig å forstå"*. Det er videre utviklet metoder for å sørge for at den informasjonen brukeren ser på skjermen alltid er den oppdaterte informasjonen uansett hvordan brukeren navigerer gjennom applikasjonen. Dette gjelder også for endringer på delte budsjettposter, hvor det er lagt inn mekanismer som oppdaterer informasjonen med en gang endringen skjer. Egne løsninger innloggingsløsninger er lagt inn for å sikre god brukeropplevelse ved at brukeren ikke trenger å vente unødvendig lenge før brukeren får logget på serveren.

4 Native, hybrid eller web

Ifølge kravspesifikasjon og ønsker fra oppdragsgiver var det en prioritet å ha en applikasjon for iOS. Dette var på grunn av interne undersøkelser hos oppdragsgiver som tilsier at deres kundemasse har en overvekt av brukere med iOS i forhold til andre plattformer. Siden dette er en applikasjon som ikke bare er tiltenkt bankens kunder, så gir ikke disse interne undersøkelsene et riktig svar for den ønskede brukermassen. Vi har derfor valgt å se på den generelle fordelingen av operativsystemer på mobiltelefoner i Norge.

Her vises en statistisk oversikt over andelen enheter og tilhørende operativsystem i Norge fra mai 2015 til april 2016 [11].



Figur 3: Bruk av plattform på mobiltelefoner i Norge

Med bakgrunn i denne statistikken vil det være lønnsomt å utvikle applikasjonen for både Android og iOS for å nå den store masse av brukere.

Videre står valget om applikasjonen skal være native, hybrid eller en webapplikasjon.Ved å velge en native løsning kreves det at applikasjonen programmeres to ganger; platformene bruker forskjellige programmeringsspråk. For iOS kan dette være Objective-C eller Swift og for Android kan dette være Java [12]. Ved å velge en løsning for web eller hybrid vil produktet kunne brukes hos flere plattformer da det utvikles i HTML5, CSS og JavaScript.

	Native	**Web**	**Hybrid**
Utførelse	Rask	Treg	Treg
Tilgang til sensorer	Ja	Nei	Ja
Distribuering	Appstore	Web	Appstore
Lagring	Database/Disk	Database	Database/Disk
Look and feel	Native	Emulert	Emulert
Varslinger	Ja	Nei	Ja
Tilkobling	Online/Offlive	Online	Online/Offline

Tabell 1: Ulemper og fordeler: Native, Hybrid, Web

Tabell 1 [12] viser en liste over nødvendige funksjonalitet og hvordan disse vil fungere på ved forskjellige applikasjonsvalg

Når det gjelder native- kontra hybridapplikasjon, ville valget vært enklere for noen år siden. Selv om hybrid ville gitt en mye bedre kryssplattform-kapabilitet var ikke teknologien utviklet nok til å gi en god nok brukeropplevelse. Facebook prøvde seg før de gikk over til native, og sa på «TechChrunch Disrupt» i september 2012 “Vår største tabbe var å legge for mye lit til HTML5”[13]. De tok steget fra en webapplikasjon som var kryss-plattform til native. Det var en hel del merarbeid for Facebook, men de fikk det som de ville når det gjaldt brukeropplevelse og ytelse. Det er lenge siden 2012 og teknologien har blitt langt bedre enn den gang.

Alle på gruppen hadde erfaring med native utvikling for Android etter emnene Mobile Development Theory og Mobile Development Project. Denne erfaringen kunne spart gruppen for mye tid dersom løsningen hadde endt opp som en native applikasjon for Android. Derimot hadde ingen av oss erfaring med utvikling for iOS. Dette betyr at en del tid ville gått til å sette seg inn i programmeringsspråket Swift og tilhørende rammeverk. I tillegg ville selve utviklingen av applikasjonen for flere operativsystemener tatt lengre tid. Allikevel hadde gruppen som mål å oppfylle så mange av oppdragsgiverens ønsker som mulig.

For å kutte tid og øke effektiviteten rundt produktet ville det ikke være lønnsomt å gjennomføre utviklingsprosessen to ganger ved å programmere løsningen for to plattformer. En native løsning ville gi best ytelse på hver plattform, men basert på funksjonaliteten definert i kravspesifikasjonen ville det ikke være nødvendig å ha en native løsning. Dette er fordi krav som budsjett-funksjonalitet, design, internettilkobling og lokal lagring ikke krever mye av enheten. Figur 1 viser at både native, hybrid og web gir muligheten for å nå kravene.

Webapplikasjoner har en fordel i at de ikke trenger å installeres på enheten. Dette fungerer ved at brukeren besøker en nettside, lagrer et bokmerke, og lager en snarvei til bokmerket på hjemskjermen. Slike applikasjoner kjøres i nettleseren. Dette gjør at applikasjonens funksjonalitet vil være låst til hva nettlesere kan tilby. Samtidig vil ikke denne typen applikasjoner kunne legges tilgjengelig i App Store og Google Play, noe som var et krav for oppgaven. Dette medførte at webapplikasjon ikke ville være riktig retning å ta.

Dermed ble hybridapplikasjon det beste alternativet for dette prosjektet siden det ville la oss

utvikle for flere plattformer samtidig og installere programvaren som en applikasjon på enheter.

5 Utviklingsmodell

5.1 Smidig utvikling

Valget ved å ha en smidig utviklingsmetodikk var basert på 12 prinsipper som heter 'The Agile Manifesto'[14]. For dette prosjektet var følgende punkter avgjørende:

- Tidlig og kontinuerlig levering av viktig funksjonalitet.
- Hyppige leveranser og kommunikasjon mot oppdragsgiver.
- Alltid mulighet for endringer.
- Selve utviklingen er det viktigste for prosjektets progresjon.
- Mål om å hele tiden bedre effektiviteten.

I henhold til kravspesifikasjon er det viktig at disse punktene blir overholdt.

Ved å ta i bruk en smidig utviklingsmetodikk vil hyppigheten på kommunikasjon mellom utviklingsgruppen, oppdragsgiver og veileder bli ivaretatt ved at kommunikasjon er en del av selve prosessen. Det vil også sørge for at produktet utvikles på en slik måte at kontinuerlig levering er fokus. Hvordan kode blir skrevet er også viktig, slik at dette gjør både endring og gjenbruk lettere.

5.2 Kanban

Det ble bestemt å bruke Kanban som utviklingsmodell. Kanban ble hovedsaklig valgt på grunn av en flekskibel og dynamisk metodikk som gjør det mulig å endre på prosesser underveis. Kanban fremtrer som er mer passende metode for dette prosjektet fremfor eksempelvis Scrum, XP og RUP som også er smidige utviklingsmetoder, men inneholder mer rammer enn Kanban [15].

- RUP (120+)
- XP (13)
- Scrum (9)
- Kanban (3) [15]

Valget innebærer ikke mengden regler prosjektet må forholde seg til, men innholdet i modellenes regler som ikke passer inn. Kanban baserer seg hovedsaklig på:

- Visualisere arbeidsflyt ved å dele oppgaver inn i mindre deler, navngi dem og sette dem på en tavle (herunder taskboard).
- Sette tydelige grenser for mengden oppgaver som skal være med i hver iterasjon.
- Fokus på effektivitet. Herunder å ha kontroll på tiden som brukes for så å effektivisere prosessen så mye som mulig underveis. [15]

Det er ikke fastsatt større mengder regler som må følges, men er derimot åpent for tilpasning. Dette lar oss som en liten gruppe kunne gjøre tilpasninger av arbeidsmetode som passer best for gruppen og prosjektet.

En iterasjon i kanban kan inneholde utvikling av en egen definert mengde av oppgaver. Vi kaller

dette videre for milepæler. Fremgangen kan rapporteres direkte til veileder og oppdragsgiver som sikrer kvaliteten av produktet ved å hele tiden ha en løpende kommunikasjon for å verifisere hver leveranse. Det sikerer også at utviklingen er i tråd med oppgavebeskrivelsen og krav som er satt, slik at oppdragsgiver får det som er ønsket.

Det defineres en løsere "time-box" for milepæler. Disse milepælene blir definert før utviklingen starter og inneholder et sett med oppgaver som må være ferdigstilt før milepælen kan ansees som nådd. Alle oppgaver som defineres innenfor en milepæl ligger i produktets backlog (Se vedlegg A Backlog).

For å kunne evaluere utviklingen underveis i prosjektet vil det være hensiktsmessig å avholde retrospektive møter. Møtet vil holdes etter at fastsatte milepæler er fullført. Dette medfører at estimering må inkluderes i utviklingsmodellen. Dette for å holde kontroll på lengden av milepæler og mengden arbeid som ligger i hver milepæl.

5.3 Milepæler

Milepæler er brukt for å definere en mengde oppgaver som skal utføres innen en bestemt tid. Å definere en mengde oppgaver som passer til en leveranse ble ansett som mer fleksibelt og bedre for oss enn å jobbe etter en standard med faste tidsrammer som krav. Et krav om tid vil alltid foreligge, men fokuset har vært å ha en mengde oppgaver som passer sammen fremfor velge oppgaver basert på en tidsramme. Den største grunnen til å bruke en slik tilnærming med løsere krav om tid er hovedsaklig på grunn av oppgaven som skal løses i nye rammeverk som ingen i gruppen tidligere har jobbet med.

I tillegg er det lettere å håndtere levereanser i henhold til møter med oppdragsgiver. Isteden for å avtale møter med oppdragsgiver for å presentere hva gruppen hadde fått utviklet innenfor en tidsramme, kan det heller presenteres med en allerede avtalt bulk funksjonalitet.

Forskjellige typer av oppgaver som milepæler inneholder:

- Ny funksjonalitet: Oppgaver som er hentet rett ut ifra Backlog. Dette er grunnleggende funksjonalitet for applikasjonen.
- Generelle oppgaver: Andre tilleggsoppgaver som tilkommer når det er noe som må gjøres men er for lite til å defineres som funksjonalitet.
- Feilretting: Feil i koden som må rettes opp.
- Forbedringer: En løsning som allerede er gjort, men må forbedres eller gjøres på en annen måte.

Det ble totalt gjennomført tre milepæler i løpet av utviklingsperioden.

5.4 Estimering

Estimeringsprosessen ble gjennomført ved å velge ut hvilke oppgaver som ville passe inn i en milepæl. Tidsperspektivet for estimeringen ble vanskelig grunnet ny teknologi og rammeverk.

Ved første milepæl ble det kun hentet ut en mengde oppgaver fra backlog som skulle løses,

og ikke satt noe konkret tidsramme på dette. Målet var å dekke grunnleggende budsjettfunksjonalitet i første milepæl.

Det å kunne kunne bedømme tid på oppgaver ble lettere når mengden oppgaver for andre milepæl skulle bestemmes. Da hadde alle vært gjennom en opplæringsprosess og en milepæl med programmering som gav et godt grunnlag for videre arbeid. Andre milepæl ble igjen basert på et sett med oppgaver for applikasjonen hvor det var bedre kjennskap til tid. Oppgaver for tredje milepæl ble også tilpasset på samme metode.

Grunnen til at det ikke er definert et konkret tidsrom for hver milepæl eller for hver oppgave, er på grunnlag av at dette er gjort tidligere ved utvikling av prototypen. Estimeringen ble her gjort ved bruk av Planning Poker[16]. Konklusjonen er at en slik estimeringsprosess ikke gav like mye tilbake i forhold til tid som ble brukt på prosessen.

5.5 Arbeidsmetodikk

Gjennom prosjektperioden har det vært stor fokus på tett samarbeid. En normal arbeidsuke har bestått av arbeid på skolen fra 08.15 til 16.00 fra tirsdag til torsdag. Mandager var arbeidstiden 10:15 - 16:00. Fredager ble fra starten av holdt ledig til andre emner, men ble også brukt mot slutten av prosjektet. Dette gjelder kun felles arbeid og ikke individuelt.

Fordelen med å prioritere gruppearbeid fremfor individuell jobbing har vært stor med tanke på at mye av applikasjonens funksjonalitet henger sammen. Det har vært stor betydning å kunne ha løpende kommunikasjon når flere jobber mot samme funksjonalitet samtidig.

5.6 Taskboard

Alle oppgaver som ble definert i produktets backlog, samt potensielle forbedringer, problemer eller andre endringer ble satt opp i en egen tavle (taskboard) i Jira som ble brukt til prosjektstyring. Denne tavlen ble definert med et annet utsende i prosjektplanen (vedlegg B), men under utviklingsperioden så den slik ut:

Backlog	Milestones	Development	QA	Done
x WiP	x WiP	4 WiP	3 WiP	x WiP

Tabell 2: Taskboard

”WiP” står for "work in progress"og definerer antall oppgaver som kan ligge i hver kolonne.

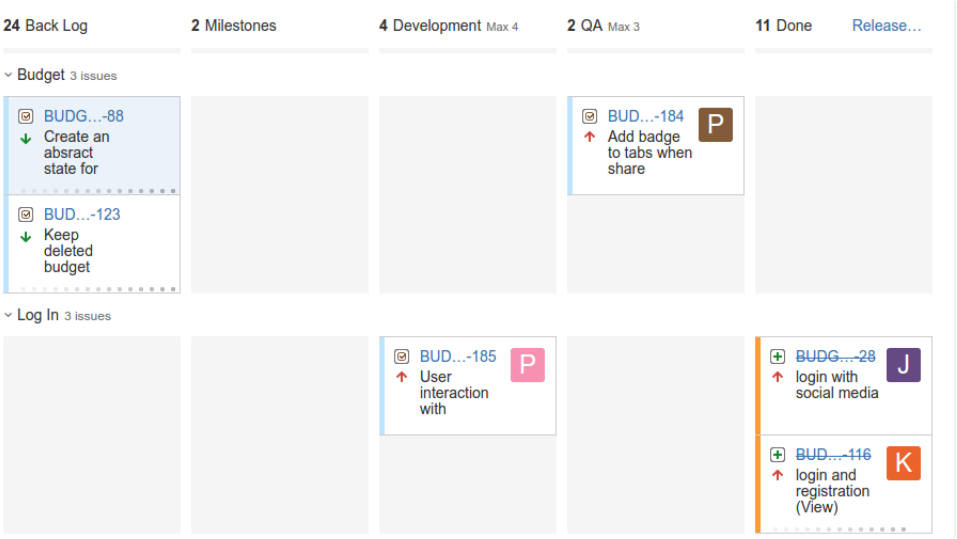
Til å begynne med ble alle definerte oppgaver lagt inn i “Backlog”. Antall oppgaver i Backlog"var dynamisk, slik at dette kunne endres på forløpende, og ikke hadde en grense. Antall oppgaver for en milepæl hadde heller ikke et fast antall og kunne også endres underveis, men det var da viktig å ha kontroll slik at det ikke ble for mange. Det lå hovedsaklig på et nivå mellom 40 og 50 oppgaver som var en blanding av nye oppgaver, forbedringer, og feiloppretting.

For kolonnen “Development” har det vært en maks grense på fire oppgaver, slik at alle har mulighet til å jobbe med en oppgave hver. Dette ble definert i prosjektplanen med en maks grense

på tre, men ble endret til fire på grunnlag av ønsket om mer fleksibilitet. Planen med tre var for at en person skulle ta seg av kvalitetssikring (QA), mens de tre andre jobbet med utvikling. Maks tre utviklingsoppgaver samtidig fungerte ikke i praksis og ble dermed endret til fire. Endringene ble gjort etter første milepæl.

I kolonnen kvalitetssikring (QA) ble det definert en øvre grense på 3. Dette for å prioritere å ferdigstille oppgaver fremfor å ha flere inne i dette stadiet slik det hoper seg opp. Det har vært perioder hvor nettopp denne kolonnen har hopet seg opp, men da har fokuset i gruppen endret seg mot å ferdigestille oppgaver her fremfor å starte på nye.

Kolonnen "Done" definerer alle oppgaver som er ferdig programmert og kvalitetssikret. Når alle oppgaver har vandret fra 'Milestones' til 'Done' er en milepæl definert som ferdig. Da blir alle oppgaver rullet ut til en ny versjon av applikasjonen.



Figur 4: Taskboard i Jira (Status 28. April 2016)

Figur 4 er et utsnitt av taskboard fra Jira som viser en liten del av hele tavlen. Hele taskboardet inneholder elementer som vist her. Alle elementer har et eget nummer, et navn, en prioritet og er tildelt en person så fort en oppgave vandrer ut ifra 'Backlog' eller 'Milestones'.

5.7 Møter

Her beskrives kort alle møter som har vært en del av utviklingsmodellen.

5.7.1 Daglig Møte

Det ble gjennomført et kort møte hvor hver person får tid til å presentere status for sine oppgaver fra det siste døgnet. Dette var for å holde hverandre oppdatert med hva hver enkelt arbeidet med, men samtidig drøfte eventuell problematikk som kunne dukke opp eller som allerede forel. Disse morgenmøtene ble holdt hver dag i startfasen. Utover i utviklingsfasen var det ikke nødvendig med et morgenmøte hver dag, men ble likevell holdt flere ganger hver uke.

Møtene ble holdt slik:

1. Oppsummering av hva vedkommende har jobbet med siste arbeidsdag.
2. Forklare eventuelle problemer eller feil som har dukket opp.
3. Legge frem sin plan over hva som skal jobbes med videre fremover.

Etter alle på gruppen har fått presentert sitt, ble det åpnet for felles diskusjon om det skulle være nødvendig. Her hadde vi ikke noe spesiell tidsramme å overholde, slik at møtet gjerne kunne ta lengre tid hvis det var noe viktig som måtte tas opp. Under utviklingsperioden ble det gjennomført 2-3 slike møter som varte lenger en normalt knyttet til tyngre funksjonalitet.

5.7.2 Møte med oppdragsgiver

Møte med oppdragsgiver var planlagt å gjennomføre i enden av hver milepæl for å kunne presentere det siste som var utviklet og få en tilbakemelding på utført arbeid. Det ble kun gjennomført ett slikt møte hvor alle var fysisk til stede, dette var etter første milepæl. Etter dette ble kommunikasjonen gjort via andre kanaler.

I tillegg ble det gjennomført et lengre oppstartsmøte før utviklingsperioden ble påbegynt. Ut over disse møtene har kommunikasjonen mot oppdragsgiver skjedd via e-post for klarering av mindre problematikk.

Følgende representerte Totens Sparebank:

- Geir Sindre von Schantz Nyborg (Banksjef forretningsstøtte)
- Dag Einar Steinsåker (Digital markedsfører og merkevarebygger)

Det ble også gjennomført et møte i forbindelse med utvikling av prototypen.

Oppstartsmøte

Møtet ble arrangert en uke etter prosjektstart i januar. Hovedpunkter for møtet var først og fremst å diskutere oppgaven, slik at begge parter hadde den samme forståelsen av produktet. Samtidig ble en prototype av den tidligere applikasjonen presentert for å kunne diskutere denne løsningen i forhold til denne oppgaven. Konklusjonen for oppstartsmøtet var at begge parter var enige, slik at produktet da var klart til å påbegynnes.

Statusoppdatering

Oppdateringene er brukt til å presentere alle nye endringer i applikasjonen for milepælen som har vært.

På møtet ble applikasjonen fremvist på stortskjerm slik at alle kunne se. Mesteparten av tiden ble brukt til å diskutere utseende av applikasjonen og hvilken informasjon som skulle vises til brukeren. Begge representanter fra Totens Sparebank er bruker av iPhone, slik at det var nødvendig å være kritisk til plattformspesifikke ønsker som oppdragsgiver da ønsket.

Videre statusoppdatering foregikk da på e-post. Det ble her presentert ny funksjonalitet i samme skala som på møtene. I tillegg kunne oppdragsgiver få muligheten til å prøve ut applikasjonen på deres egen telefon ved bruk av Ionic View (En egen applikasjon som gjør det mulig å teste budsjettapplikasjonen). Ved bruk av Ionic View er det enkelt å laste opp ferdig funksjonalitet og distribuere det videre til andre som ønsker å prøve applikasjonen.

Se vedlegg E for møtereferat.

5.7.3 Møte med veileder

Veiledermøte har stort sett blitt gjennomført hver torsdag til et fast tidspunkt. Det har også vært noen unntak hvor enkelte møter har blitt utelukket. Dette har vært under oppstart i utviklingsfasen, da et møte med veileder ikke var nødvendig.
Disse møtene har stort sett blitt brukt til å få tilbakemelding på skriftlige produkter som prosjektplan, forprosjekt og sluttrapport. I tillegg til å få tips og veiledning rundt prosesser i utviklingsfasen og på produktet i seg selv.

5.7.4 Retrospektive møter

Når en milepæl ble nådd ble det holdt et lengre møte i gruppen som hadde en lengde på én til to timer. Dette var et møte med tilbakemeldinger for perioden som har vært. Hva som har fungert bra og hva som har ikke fungert så bra har vært fokusområde. Tilbakemeldinger ble notert ned og tatt med videre til neste milepæl.

Møtene startet med en oppsummering av at gruppeleder har gått igjennom alle oppgaver og hva som ble gjort innenfor utført milepælen. Videre ble følgende hovedpunkter gått igjennom:

- Oppnådd arbeid
- Arbeidsmetode / metodikk
- Fremdrift
- Forbedringspotensiale

For hvert punkt fikk alle sammen 5 minutter til rådighet til å notere ned positive og konstruktive tilbakemeldinger. Tilbakemeldingene gikk hovedsaklig på prosessen, men personlige tilbakemeldinger var også tillatt. Alle fikk mulighet til å komme med sine tilbemeldinger uten at dette ble kommentert av andre (én hadde ordet om gangen). En felles diskusjon om eventuelle uenigheter eller uklarheter ble gjort til slutt. Eventuell diskusjon pågikk til alle var enige. Se vedlegg E for referat fra disse.

5.7.5 Oppsummering

Bruk av daglige møter og retrospektive møter var med å effektivisere utviklingsfasen i henhold til utviklingsmodell ved at vi gav tilbakemeldinger på hva som fungere bra og hva som fungerer dårlig. I tillegg holdt vi hverandre oppdatert, noe som økte effektiviteten.

5.8 Første milepæl

28. januar - 25. februar

Den første milepælen ble planlagt å inneholde helt generell funksjonalitet for budsjettering som vist under. Det viktigste var å få på plass sparing, budsjettkategorier, og mulighet for å legge til utgifter tilknyttet en budsjettkategori. Noen andre oppgaver som implementering av graf på forsiden, internasjonalisering og legge til en egen siden for innstillinger og informasjon. Brukergrensesnitt ble ikke prioritert. Alle oppgaver av ny funksjonalitet var som følger:

- Implementasjon av graf på forsiden
- Sparemål

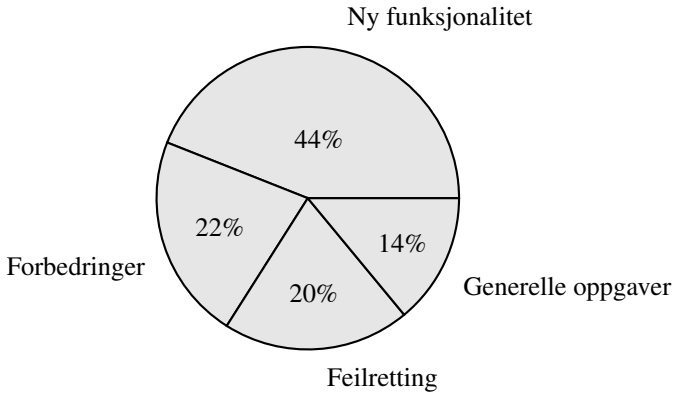
    - Opprette nye, endre, slette og kunne se dem
    - Lagre til database
- Budsjettkategorier
    - Standard- og egendefinerte
        - Opprette nye, endre, slette og kunne se dem
        - Lagre til database
- Internasjonalisering
- Klargjøre egen skjerm for informasjon og innstillinger
- Utgifter
    - Utseende
    - Integrering mot budsjettkategori
    - Lagre til database
    - Faste utgifter

Denne milepælen var preget av mye individuelt arbeid, selv om vi satt i fellesskap. Det ble jobbet mye individuelt fordi alle måtte lære seg nye rammeverk og metoder for utviklingen. Vi hadde ikke fastsatte standarder å utvikle etter, så koden fra forskjellige medlemmer var strukturert forskjellig. Funksjonaliteten som var definert for milepælen ble implementert på en bra måte, men stilen i de ulike delene var forskjellig. I tillegg til svak kommunikasjon innad i gruppen førte det til at en del kode måtte endres.

Mot slutten av milepælen ble kommunikasjonen forbedret og standarder fastsatt. Dette gjaldt mest strukturering, orden, rekkefølge og bruk av navn i koden. Dette gjorde at oppgaven hadde mer preg av samarbeid mot slutten.

Grafen nedenfor viser forskjellen mellom typer oppgaver som ble gjort under den første milepælen. Her ser vi blant annet at planlagt ny funksjonalitet kun tilsvarer 44%. I tillegg ble de første ukene brukt mye til prøving og feiling siden alle måtte sette seg inn i ny teknologi og nye rammeverk.

Totalt antall oppgaver: 50



Figur 5: Fordelinger oppgaver - Første milepæl

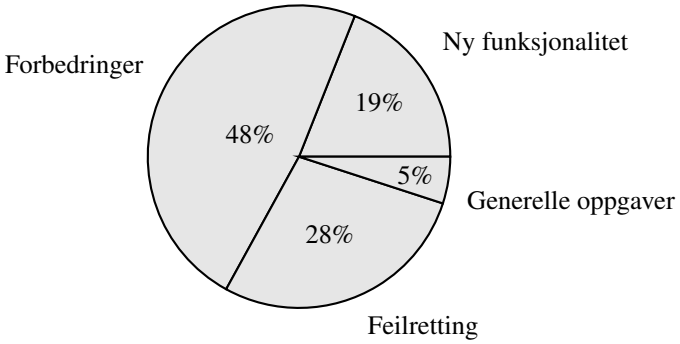
Grafen viser at det er en jevn fordeling mellom forbedringer, feilretting og generelle oppgaver som har blitt påført milepælen som oppgaver i tillegg til den nye funksjonaliteten.

5.9 Andre milepæl

26. februar - 12. april

Andre milepæl ble innledet av et større retrospektivt møte hvor det det var mye fokus på samarbeid og rammer for gruppen. Det ble satt standarder som hjalp på samarbeidet innad i gruppen. I stedet for å lage en funksjon som ble brukt et sted, ble det laget felles løsninger som alle kunne bruke. Dette grepet lot oss unngå mye dobbeltarbeid. Utviklingen heretter bærer preg av effektivitet og oppgaven bærer større preg av samarbeid.

Antall oppgaver: 43



Figur 6: Fordelinger oppgaver - Andre milepæl

Som vist i grafen ovenfor begynte den andre milepælen med en liten andel oppgaver som var av ny funksjonalitet. Mye av hovedfunksjonaliteten var på plass etter første milepæl. Hovedmengden ligger i andelen oppgaver av type forbedring og en del retting og klarering av eksisterende kode. Følgende ny funksjonalitet ble lagt til her:

- Innloggingsmekanisme (Backend)
- Inntekter
    - Faste inntekter
- Implementere SIFO-busdjett til standard kategorier
- Applikasjonens introduksjon

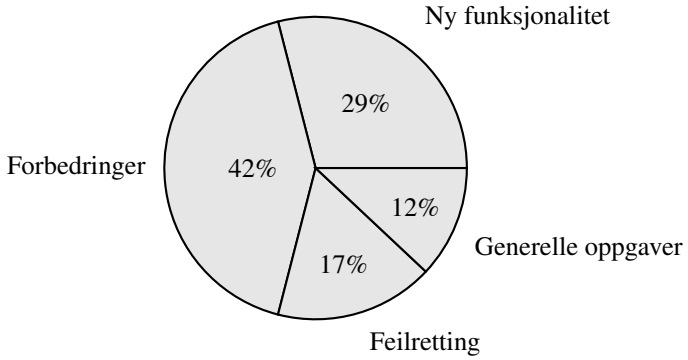
5.10 Tredje milepæl

13. april - 17. mai

Tredje milepæl ble innledet med fokus på delingsfunksjonalitet, løpende utgifter/inntekter og forbedringer av brukergrensesnitt. Det var under denne perioden brukertestingen ble foretatt. I dette stadiet i utviklingsfasen hadde alle vært igjennom to milepæler og kode skrevet på samme måte av alle. Kunnskapsnivået innen de forskjellige rammeverkene lå også likt innad i gruppen slik at prosessen for kvalitetssikring også endret retning. Kvalitetssikringsprosessene dreide seg

heretter mest om å se at koden fungerte og ikke var mangelfull. Tidligere var hovedfokuset med disse prosessene å komme med alternative forslag til hvordan kode kunne skrives for å få den mer effektiv og ryddig. Effektiv og ryddig kode var allerede en standard i gruppen som alle fulgte, så dette trengtes ikke å følges opp like nøye lenger.

Antall oppgaver: 24



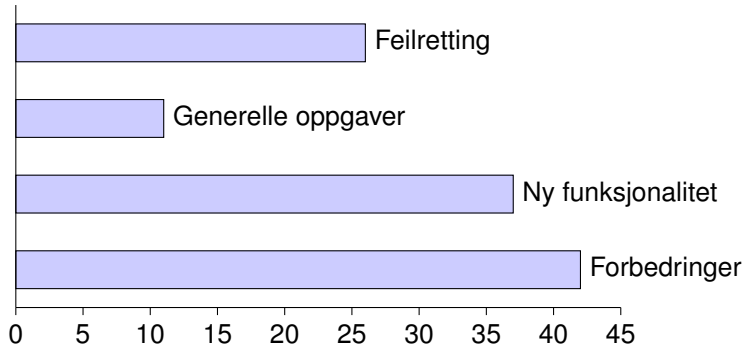
Figur 7: Fordelinger oppgaver - Tredje milepæl

Følgende ny funksjonalitet ble lagt til:

- Innlogging og registrering
- Kundeservice
- Grensesnitt for deling av budsjettposter

5.11 Oppsummering milepæler

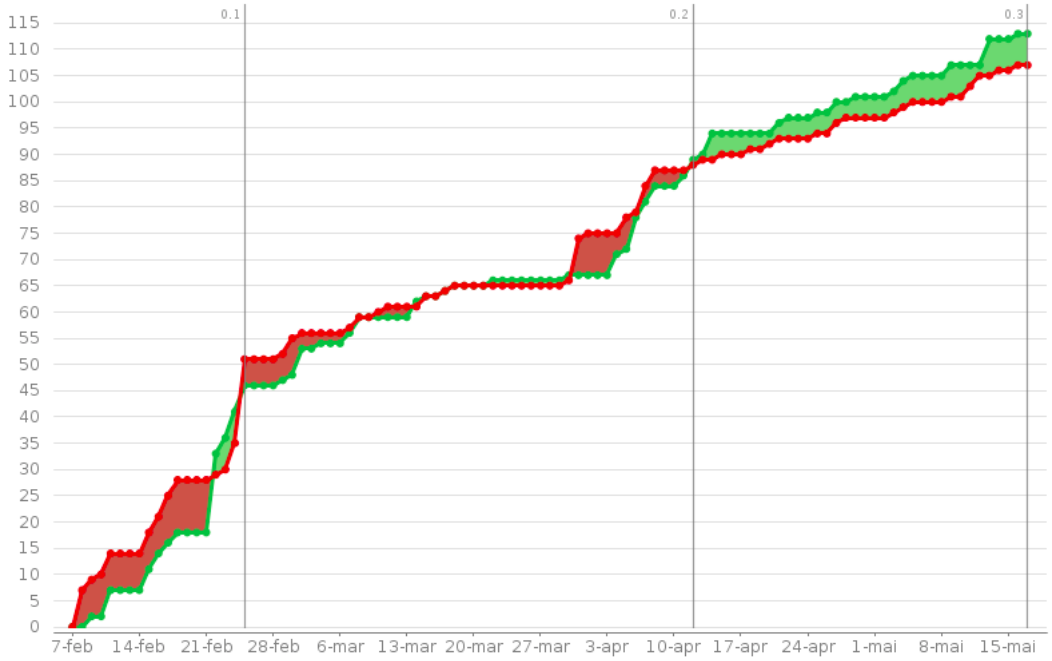
Grunnen til den store mengden av forbedringer er at første milepæl ble i tillegg til utvikling brukt som opplæring i verktøyene. Etter det ble gjennomført et retrospektivt møte mellom første og andre milepæl hvor det ble fastsatt standarder og metoder, ble eksisterende kode utbedret for å kunne tilfredstille gitt standard. I tillegg var det mye funksjonalitet som ikke fungerte helt som det skulle (ikke direkte en feil) som også trengte forbedringer. Mange av forbedringsoppgave har vært å presentere informasjon for brukeren på en bedre måte enn hva det resulterte i etter første milepæl hvor dette ikke var prioritert.



Figur 8: Statisktisk oversikt over alle oppgaver

Grafen viser at denne milepælen har hatt fokus på oppgaver som gjelder forbedringer og feilrettinger. Det er lagt til annen ny funksjonalitet. Mye av dette er basert på nye rammeverk og teknologi.

5.12 Progresjon

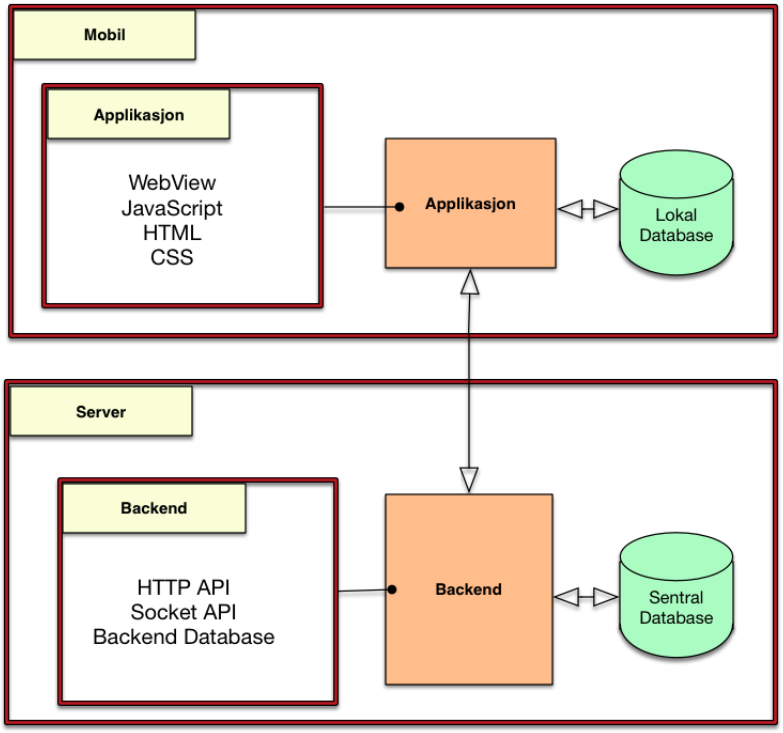


Figur 9: Nye og ferdige oppgaver

Figur 9 viser differansen mellom nye og ferdig oppgaver gjennom hele prosjektperioden. Den røde grafen viser antall uferdige oppgaver, mens den grønne viser antallet ferdige oppgaver. X-aksen viser utviklingsperioden i tid, Y-aksen viser antallet oppgaver. De to vertikale strekene midt i diagrammet viser skillet på milepæler. Den første streken viser enden på første milepæl og en andre viser enden på andre milepæl.

Grafen ovenfor viser at forholdet mellom ferdige og uferdige oppgaver har vært tilnærmet null ved slutten av milepæler. Antallet oppgaver som har vært uferdige og ferdige har vært parallelt gjennom store deler av utviklingsfasen. Dette er på grunnlag av at det stadig ble definert oppgaver for for retting, endring eller forbedring av eksisterende kode. Disse tilkom ettersom ny funksjonalitet ble ferdig. Hvis det hadde blitt satt krav om at det ikke hadde vært mulig å definere oppgaver underveis, ville dette sett helt annerledes ut. Grafen ville da vist kortere milepæler og tilsvarende endring mellom uferdige og ferdige oppgaver.

6 Systemariktektur



Figur 10: Systemarkitektur

Applikasjon er delt inn i to hoveddeler.

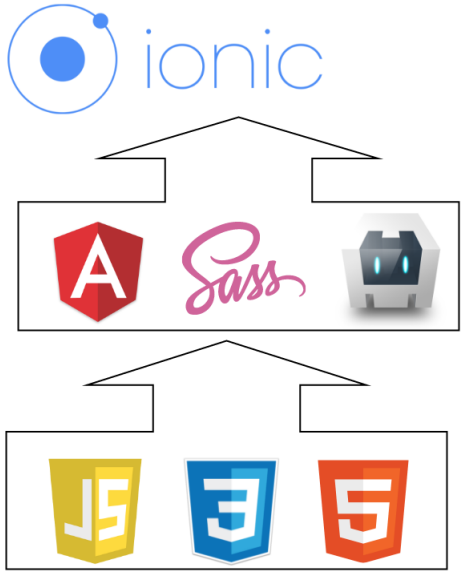
- **Mobil Plattform** som kjører applikasjonslogikk og tilbyr lokal lagring.
- **Server** som kjører backend-logikk og tilbyr sentral lagring.

Vi har valgt å gå for en tykklient-applikasjon. Dette fordi vi vil tilby ikke-innloggede brukere så mye funksjonalitet som mulig og for å ikke belaste serveren med for mange forespørsler som klienten kunne gjort selv. Serverens oppgaver går i hovedsak ut på å fungere som en proxy for den sentrale databasen og gjøre kritiske forespørsler mot tredjeparts brukertjenester. Serverens oppgaver beskrives i detalj i kapittel 6.3 Backend

Applikasjonen er ikke avhengig av forbindelse mellom de to hoveddelene. Applikasjonen kan kjøres uten nettilgang hvor den kun benytter seg av den lokale lagringen for kontinuitet, men vil da ikke utnytte funksjonaliteter som deling av budsjettposter, og mulighet til å ha samme budsjett på flere enheter.

6.1 Applikasjon

Prosjektet er utviklet med flere forskjellige lag av teknologier som gjør det mulig å bygge en applikasjon som kjøres på flere plattformer.



Figur 11: Teknologistack

I bunnen finner vi de forskjellige grunnleggende webteknologiene som gir et grunnlag for de andre teknologiene. Disse danner en treenighet for web-utvikling hvor HTML5 brukes for å definere layout, CSS definerer utseende og JavaScript definerer oppførsel.

AngularJS [17] er et rammeverk for å skape dynamiske flersiders web-applikasjoner. Det er utviklet for å tilrettelegge bruk av MVC-patternet gjennom sine Controllere, og Directives(view). AngularJS hjelper til med skape uttrykksfulle HTML-elementer som er gjenbrukbare. I tillegg bidrar AngularJS med å gjøre prosjektet mer skalerbart ved å lagre unna avgrenset og beslektet kode i moduler. Disse kan inkluderes i andre moduler. Det benytter seg av en 2-veis binding mellom view og kontrolleren som gjør at dataene blir direkte oppdatert på view-et når endringer i modellen oppdages av kontrolleren. Bruken av et slikt designmønster hjalp oss å holde kodebasen ryddig både for vår egen del, og for eventuelle videreutviklere.

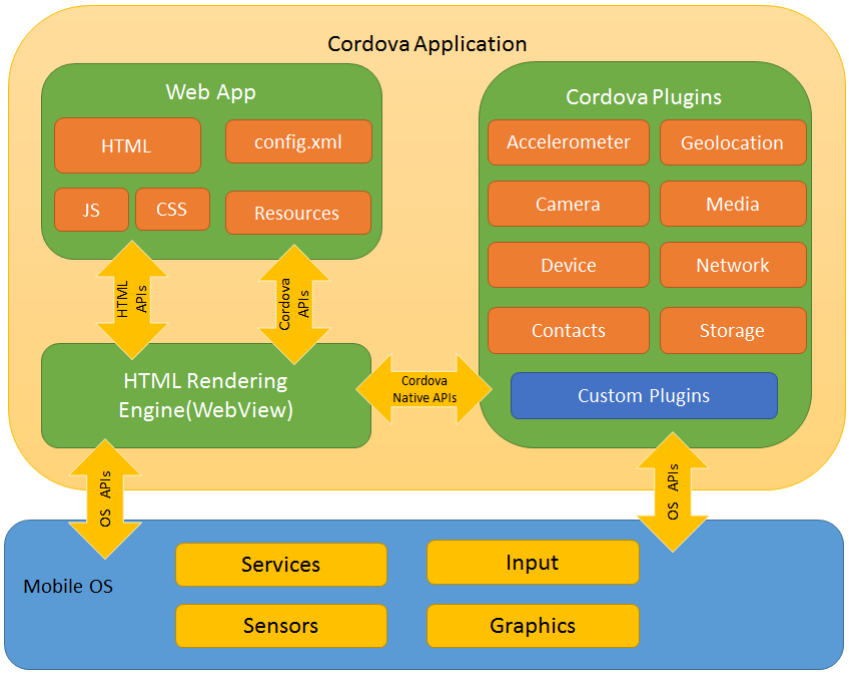
AngularJS i seg selv inneholder mye av funksjonaliteten vi hadde bruk for under utvikling av applikasjonen, men for enkelte problemer tok vi i bruk tredjeparts Angularutvidelser. ”Angular-translate” er en slik utvidelse, som lot oss internasjonalisere applikasjonen. Dette var ikke en høy prioritet for oppdragsgiveren, men prosjektgruppen ønsket å få dette til for å potensielt øke brukerantallet. Dette var spesielt attraktivt ettersom angular-translate gjorde jobben relativt enkel. Utvidelsen ble brukt til å lokalisere applikasjonen for norsk- og engelskspråklige brukere, men dersom flere språk ønskes kan dette legges til på samme måte som de eksisterende språkene.

En annen utvidelse som er tatt i bruk i applikasjonen er Angular Chart. Vi har brukt den til å vise frem dynamiske grafer om brukerens pengebruk. En slik graf er synlig på hjem-skjermen, hvor formålet er å vise brukerens totale utgifter sammenlignet med hverandre, i form av et sek-

tordiagram. Utvidelsen er også brukt til å vise historiske data om pengebruk innen spesifikke budsjettposter. Dette gjøres ved hjelp av et linjediagram.

Sass[18] er en utvidelse til CSS som gjør CSS mer dynamisk ved at den tillater å tilegne variabler i stilsett slik at det lettere kan gjøres forandringer et sted og det vil ha effekt flere steder. Det er også mulighet for å inkludere spesfikke regler fra andre sass filer.

Cordova[19] er rammeverket som bygger koden til ulike plattformer og setter inn web-kildekode til et web-view i den native applikasjonskoden (se figur 12). Den lager også et JavaScript-grensesnitt mot native API-er gjennom plug-ins.



Figur 12: Cordova

Disse oversetter Javascript-kode til plattformspesifikk kildekode og setter det inn i kildefilene når Cordova-prosjektet bygges. Dette lot oss utvikle applikasjonen uten å forholde oss til plattformspesifikke løsninger, som for eksempel låsing av bildet i portrettmodus.

Det siste laget i teknologistacken er:

Ionic[20] er et rammeverk for hybridapplikasjoner på mobile enheter. Det bygger på en kombinasjon av Angular og Apache Cordova. I hovedsak er det ferdiglagde HTML-, CSS- og JavaScript-elementer som er laget spesifikt for mobile enheter. Ionic styrer store deler av brukergrensesnittet, i tillegg til at det tilpasser utseendet til å følge plattformspesifikke standarder i forhold til mobilens operativsystem.

6.2 Database

Valg av database ble gjort med tanke på funksjonaliteten systemet vårt skal tilby. Brukere skal ha tilgang til sine data på enheten dersom de ikke har nettverkstilgang. De skal også ha tilgang til disse dataene på flere enheter. Det skal skal også kunne deles mellom brukere. I følge

CAP-teoremet[21] sies det at man kun kan ha to av de tre egenskapene for et nettverksystem: Consistency(C), Availability(A) og Partition tolerance(P). Når man ser funksjonaliteten systemet skal tilby og egenskapene i CAP teoremet er det to som er sentrale: Availability og Partition tolerance er de viktigeste for vår applikasjon. Partition tolerance er sentralt grunnet at dette er en mobil applikasjon, avbrudd og varierende nettverksmuligheter er en garanti. For at systemet skal kunne fungere normalt under slike omstendigheter er dette en sental egenskap datalagringen må ha. Da kan det velges mellom Consitency og Availabilty. Når et slikt avbrudd skjer er det to muligheter:

- Ofre availability: Brukeren kan ikke skrive til sin lokale database. Eventuelt kan ikke andre brukere skrive til den sentrale databasen. Dette opprettholder konsistensen siden man begrenser tilgjengeligheten på denne måten.
- Ofre Consistency: Brukeren kan skrive til sin lokale database og andre brukere kan skrive data til den sentrale databasen. Dette vil gjøre at det blir inkonsistens i dataene men opprettholder tilgjengeligheten.

Tradisjonelle relasjonsdatabaser som MySQL har egenskapene C og A [22]. CouchDB er en database som har egenskapene vi ser etter, nemlig A og P. Dette er en noSQL database der det ikke er definert en struktur på forhånd son må følges, men heller benytter seg av mer dynamisk dokumentstruktur. En annen grunn til vi valgte CouchDB var en versjon av dette databasesystemet som var laget for å kjøre i nettleseren, kalt PouchDB [23]. I tillegg til å være laget for nettlesere har PouchDB gode replikeringsmuligheter med CouchDB. Dette passet utmerket for systemet vårt med en database på klient og en sentral database for brukeren. Brukere lagrer da alltid data på sin enhet og deretter blir dataene replikert til den sentrale databasen når det er mulig og hvis brukeren ønsker dette.

6.3 Backend

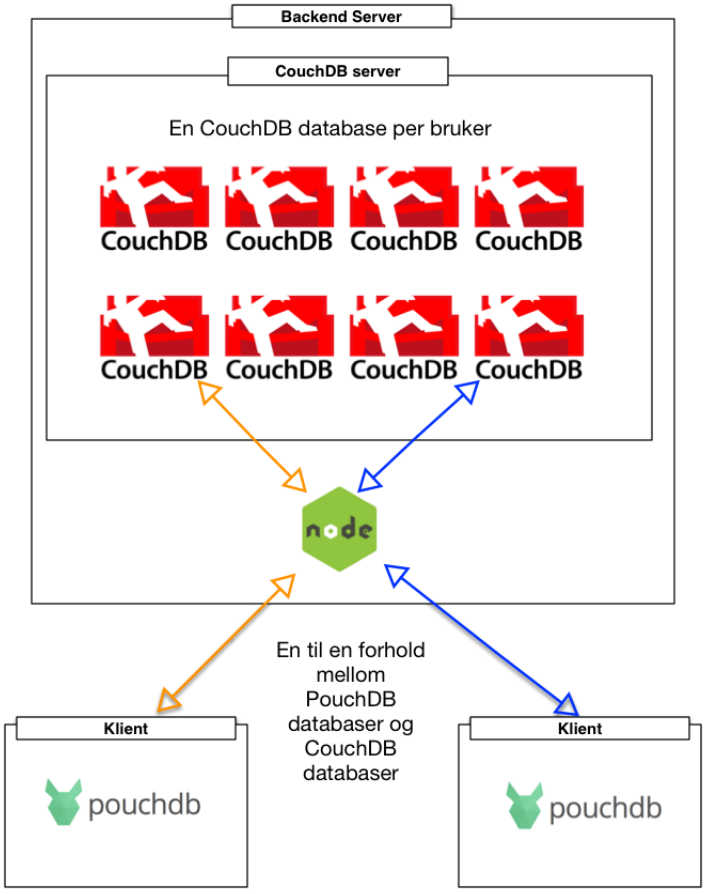
For applikasjonens backend ble valge å bruke NodeJS. NodeJS er et ”event driven” JavaScript-runtime som kjører i én tråd hvor operasjoner skjer asynkront [24]. Dette betyr at NodeJS fungerer bra for kommunikasjon i sanntid mellom server og klient og at den har bra kapasitet på gjennomstrømming. Ulempene er at hvis den skal gjøre for mye kalkuleringer som tar lang tid vil dette være ugunstig siden det kun er én tråd. Ved å bruke Node.js kunne vi skrive i samme språk og mye av den samme stilen for både frontend og backend. Koden kunne dermed også kvalitetssikres og forstås av alle i gruppen. Dette gjorde også at vi kunne bruke npm som er pakkehånderingssystemet for Node.js for å legge til moduler. På frontend ble også npm benyttet slik vi allerede var kjent med verktøyet.

6.3.1 Backend eller kun database

CouchDB er backend databasesystemet til applikasjonen. Denne har et HTTP API hvor det sendes forspørsler med HTTP-forespørsel som eksempelvis GET og PUT for å lagre og hente informasjon [25]. Dette kombinert med synkroniseringen til en PouchDB-database gjorde at det var en mulighet for å kun ha en CouchDB-databaseserver kjørende uten noen egen logikk i backenden. Dette ville krevd mer av hver klient siden all logikk måtte ligget der. Databasen ville da kun ha fungert som en sentral lagringsplass for informasjon. Fordi replikeringen i hovedsak er for én database til en annen bestemte vi oss for å ha én database per bruker [26]. Alle disse databasene vil da bli håndtert av CouchDB-serveren og alle er tilgjengelige med det samme HTTP API. Problemet med dette var sikkerheten. Det er mulig å begrense hvilke brukerne som kan skrive eller

lese fra en database slik at kun eieren av databasen kan bruke den. For å sette disse grensene og opprette databasen i utgangspunktet kreves en administratorbruker. Derfor måtte løsningen bli å ha en felles administratorbruker som gjorde dette på vegne av brukerne i applikasjonen. Dette så vi ikke som noen god løsning da en administratorbrukers innloggingsinformasjon ville ligget på klientsiden. Dette kunne skapt en sikkerhetsrisiko fordi man kunne potensielt tilegnet seg denne informasjonen og dermed fått administrator-tilgang på den sentrale databaseserveren.

I stedet for dette kunne vi ha en backend som gjorde database-operasjoner på vegne av applikasjonen og ga den et minimalt grensesnitt for å kommunisere med databasen igjennom backend istedenfor hele HTTP API-et til CouchDB. CouchDB ville ikke da bli eksponert til nettverket utenfor serveren og alle HTTP forespørslene for API-et til CouchDB blir kjørt lokalt på serveren fra NodeJS. En av grunnene til at dette lot seg lett gjøre er at applikasjonen vår benyttet en svert liten del av hele API-et til CouchDB. Ingen enkeltdata skal hentes eller skrives mellom databasen på enheter og databasen sentralt. Disse skal kun holde seg oppdatert via synkronisering. Dette betyr at backenden vår ikke måtte implementere funksjonalitet for mange av operasjonene på databasen.



Figur 13: Databaser sentralt og lokalt med NodeJS som proxy i mellom

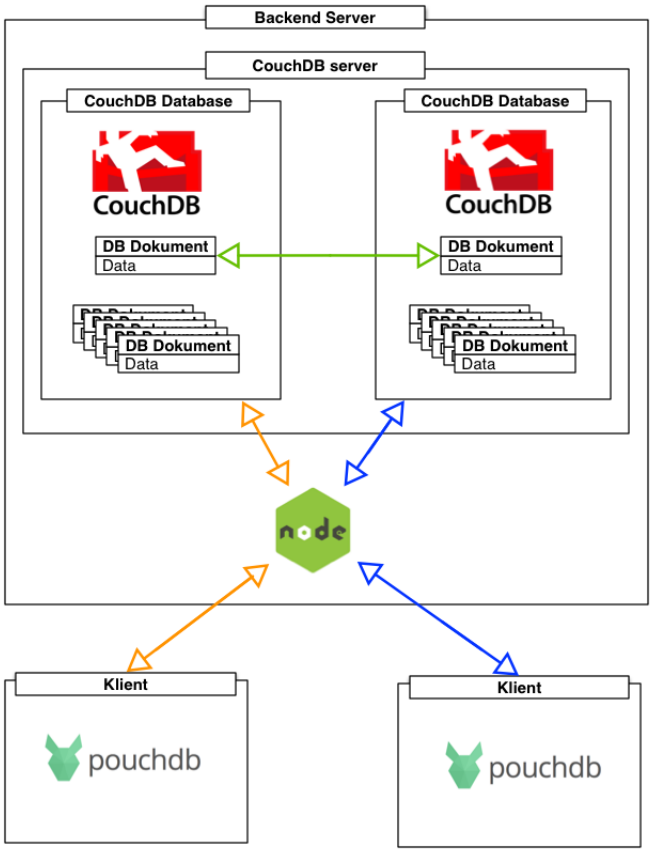
Å skjerme CouchDB og kun la NodeJS-serveren gjøre API-kall til den på vegne av klientene mener vi vil øke sikkerheten til systemet vårt. Grunnen til dette er at vi kan ha kritisk funskjona-

litet kjørende sentralt og ha kontroll over hvilke operasjoner klienter kan gjøre mot den sentrale databasen.

6.3.2 Replikering

Når NodeJS brukes som proxy mellom klient og server krever dette mer arbeid enn å replikere direkte. CouchDB og PouchDB er så like at PouchDB kan brukes som et grensesnitt til en CouchDB-database. Dette betyr i praksis at i stedet for å benytte CouchDB sitt HTTP-API for å gjøre operasjoner kan heller PouchDB tas i bruk som et JavaScript grensesnitt for CouchDB. Dette lot oss benytte en plugin til pouchDB som skriver innholdet i en database til en strøm. Denne benyttes på server-siden der PouchDB er et grensenitt til CouchDB samt på klientsiden for PouchDB direkte. Etter å ha prøvd flere metoder for å sende denne strømmen fra klient til server og motsatt, endte vi opp med å bruke sockets for å løse dette. Ved å bruke sockets kunne vi ha toveis kommunikasjon mellom klient og server. Dette gjorde at serveren kunne sende informasjon til klienter uten at de har hadde sendt en forespørsel. Dermed kunne serveren selv kan si ifra om viktige hendelser til klienten fortløpende.

Sockets ble implementert i applikasjonen via socket.io som er et bibliotek for JavaScript for kommunikasjon igjennom sockets. I tillegg til at det gjorde databasereplikeringen lettere å gjennomføre var det også en annen viktig faktor som gjorde at vi valgte å implementere sockets for hovedkommunikasjonen mellom klient og server. Dette var at brukerne skulle ha mulighet for å dele budsjettposter med hverandre. Funksjonalitet som dette krever at brukere har mulighet til å sende forespørsler til hverandre om at det er greit å dele denne informasjonen. Hvis vi hadde brukt HTTP-forespørsler for denne kommunikasjonen hadde klientene selv måttet spurt serveren om de hadde mottatt en forespørsel. Dette hadde blitt veldig unøyaktig og unødvendig siden klienten ikke vet om dette er tilfellet samt at dette ikke er en hendelse som skjer så ofte for en gjennomsnittlig bruker. Derfor hadde det vært bedre om serveren kunne si ifra til klienten dersom om han eller hun hadde mottatt en slik forespørsel.



Figur 14: Deling av et dokument mellom to brukere

Disse delingsforespørslene er for budsjettposter. En budsjettpost kan være felles for to eller flere brukere hvis en slik forespørsel godtas av den andre brukeren. Delingen foregår ved at de sentrale databasene til brukerene replikerer dataene for denne budsjettposten imellom seg som vist på figur 14. Endringer fra andre brukere blir så replikert videre til den lokale databasen på enheten til brukerene.

6.3.3 Oppbygning

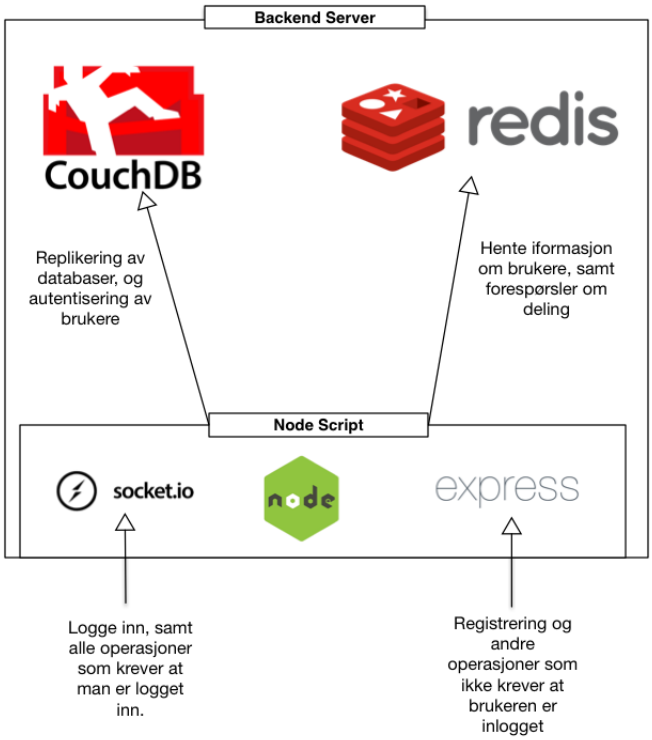
Vi bruker to utvidelser til socket.io. Den ene er socket.io-stream som gjør det mulig å bruke NodeJS sitt stream-API gjennom sockets. Grunnen til dette var å kunne sende replikeringsstrømmen fra databasene mellom klient og server. Den andre er en autentiseringsutvidelse som gjør at brukere som kobler til en socket må autentisere seg hvis annen informasjon enn den som kreves for innlogging skal kunne sendes gjennom socketen. Vi kunne implementere våre egne autentiseringsmetoder som samspilte bra med denne utvidelsen.

I tillegg til socket-kommunikasjon har serveren har også flere HTTP-ruter det kan sendes forespørsler til. Disse er for alle operasjoner som ikke krever innlogging som registrering, henting av tokens og lignende. Grunnen til at strukturen er slik er å ha et klart skille mellom operasjoner som krever at brukeren er innlogget og de som ikke krever dette. HTTP-funksjonaliteten ble implementert ved hjelp Express.js som er et web-rammeverk for NodeJS. Disse HTTP-rutene ble i hovedsak brukt for innlogging og registrering og andre operasjoner som kan gjøres i sammen-

heng med dette.

For innlogging og registrering tilbyr applikasjonen tre forskjellige metoder brukerne kan velge imellom. En kan bruke eksterne som Facebook- og Googlekonto eller en konto som kun er for vårt system. Serveren gjør HTTP forespørsler til CouchDB, Facebook, Google API-et bak kulissene når brukeren skal registrere seg eller logge inn. Svarene på disse forespørsler avgjør om brukeren får logge inn eller registrere seg. For Facebook og Google er prinsippet at påloggingen skjer på klienten og mottar en access token som svar fra disse tjenestene. Access token blir sendt videre til serveren som sjekker om denne er gyldig via tjenesten brukeren logget seg på med. For innlogging via databasen benyttes CouchDB's autentiseringssystem. Et HTTP-kall til databasen med brukernavn og passord er alt som trengs for å verifisere innloggingsdata.

For å holde på brukerinformasjon som serveren trenger for å håndtere innlogging og deling mellom brukere ville vi ha et mer robust system enn å lagre denne informasjonen i selve backendscriptet. Vi ville heller ikke benytte oss av CouchDB fordi dette ville være tregere på grunn av lesing fra disk og overføring via HTTP, og ville det hatt data som serveren brukte for å operere i samme databasesystem som brukerenes data. Derfor valgte vi å ha denne informasjonen i en Redis-database. Denne lagrer data i minnet på serveren og passer bra til enkel informasjon som skal være lett tilgjengelig. Den benytter key-value for lagring av data, men har støtte for mer avanserte strukturer som lister og sett[27]. Socket-id, brukernavn, databasenavn, og navnet til brukeren lagres her slik at det skal kunne knyttes til en bruker. Redis støtter også enkel mønstergjennkjenning som wildcards (? *) for å søke etter nøkler. Dette er blant annet nyttig når en klient skal søke etter en bruker han eller hun vil dele budsjettposter med.



Figur 15: Overblikk over backend

I figur 15 vises grunnleggende struktur på serveren for systemet. Node.JS kjører backendscriptet, socket.io tilbyr socket kommunikasjon og Express.js hånterer HTTP-kall. De to databasesystemene blir brukt til ulike formål.

7 Frontendarkitektur

7.1 Hovedkomponenter

Applikasjonens frontend er i all hovedsak bygget opp av fire hovedmoduler;

1. Home
2. Budget
3. Savings
4. Settings

Disse fire modulene har egne tabs i applikasjonen, og har en eller flere undermoduler. Disse undermodulene er ikke nødvendigvis direkte knyttet til de større modulene, men det er naturlig å kategorisere dem på denne måten. Et eksempel på dette vil være måten Transaction-modulen er knyttet til Home. Tilknytningen mellom disse er at Home inneholder to knapper som fører brukeren til et vindu for å legge inn inntekter eller utgifter. Dette foregår da i Transaction, som ikke har noen direkte tilknytning til Home, bortsett fra at de deler history stack fordi de ligger i samme tab.

7.1.1 Home

Home inneholder ett view, noe som også gjelder Transaction. Disse er henholdsvis hjemskjermen og skjermen for å legge til utgifter eller inntekter. Hjemskjermen inneholder et diagram som brukes som en visuell indikator på brukerens forskjellige utgifter sammenlignet med hverandre. Home-modulens kontroller tar seg av nødvendige utregninger for å fremstille diagrammets informasjon, ved at den ser igjennom alle budsjettpostenes utgifter og trekker ut de som er lagt til i inneværende måned og summerer disse. Da legges summen og budsjettposten til i datasettet for diagrammet hvis summen er ikke er 0. I tillegg vil det bli lagt til en tilfeldig utregnet farge som reflekteres i diagrammet og i diagramforklaringen. Det kontrolleres at fargene ikke blir for like ved en enkel fargesammenligningsfunksjon. Deretter får teksten til diagramforklaringen en kontrastfarge ut ifra hvor lys eller mørk fargen er.

Transaction-modulen kan som tidligere nevnt tenkes på som en undermodul av Home. Denne lar brukeren legge inn utgifter eller inntekter avhengig av hvilken knapp som ble trykket på hjemskjermen. Brukeren kan her legge inn enkelte eller gjentagende inntekter. Kontrolleren samler informasjonen brukeren skriver inn, og sender det videre til userModel som holder på dataene for applikasjonen, som også sender informasjonen videre til dbPutter som lagrer dette i databasen.

Ordet ”transaction” blir brukt som en fellesbetegnelse på inntekt og utgift både i modulnavn og i denne rapporten. Dette ordet blir ikke brukt i brukergrensesnittet, for å sikre at brukeren ikke mistolker det som faktiske banktransaksjoner.

I tilfeller hvor brukeren legger inn gjentagende utgifter eller inntekter, vil kontrolleren oppføre seg relativt likt som når enkelttilfeller legges inn. Forskjellen er at det lages et repeater-objekt som representerer den gjentakende utgiften. Repeater-objekter er en sentral ressurs når det gjelder gjentagende utgifter og inntekter. Disse brukes av en tjeneste kalt transactionRepeater, som

kan leses mer om kapittel 7.4 Kart Over Tjenester.

7.1.2 Budget

Budget er hovedmodulen som styrer brukerens budsjett. Dette inkluderer de budsjettposter brukeren har satt opp og tilhørende utgifter og inntekter. Modulen består av fire undermoduler, som hver har en spesiell oppgave knyttet til budsjettet. Disse er budget-new, budget-display, budget-list og budget-share.

Budget-new lar brukeren legge til en ny budsjettpost for enten inntekter eller utgifter. Budsjettposten blir gitt et ikon, et månedsbeløp, et navn og en beskrivelse. Når dette er lagret, kan budsjettposten tas i bruk. Ikonene som brukeren kan velge mellom er et sett med ikoner som ble gitt til oss som en del av design-kravene fra oppdragsgiveren. (Se kapittel 7.2.3 Uttrekk av budsjettpostikoner)

Dersom brukeren ønsker å se informasjon om en spesifikk budsjettpost, brukes budget-display. Denne undermodulen presenterer historiske data, deriblant summene fra tidligere måneder og alle eksisterende utgifter eller inntekter.

Budget-list fremviser en liste over alle budsjettposter. Fra denne listen har brukeren tilgang til å velge en post å få mer informasjon om. Dette vil da føre brukeren til budget-display. Brukeren har også tilgang til å legge inn nye budsjettposter, og endre eller slette eksisterende poster. Dette vil da føre brukeren til budget-new.

Budget-share er undermodulen som lar brukeren dele budsjettposter med andre. Denne kan nås ved å trykke på knappen ”Del budsjettpost” i budget-display. Mesteparten av arbeidet bak denne delingen foregår på backend, mens denne undermodulens funksjon er å la brukeren søke etter andre brukere, og sende dem forespørsler om deling av budsjettposter.

7.1.3 Savings

Savings er hovedmodulen for sparing og tar seg av brukerens sparemål og kan sammenlignes med budget-list, da den på samme måte viser en liste over brukerens sparemål. Her er det mulig å se fremdriften på de forskjellige sparemålene ved hjelp av en progressbar og konkrete verdier for status per dags dato sammenlignet med målet.

Hvert sparemål er et objekt som inneholder dataene

- title - Sparemålets navn.
- id - En unik identitet.
- dateEnd - Sluttdato
- dateStart - Startdato eller siste dato sparemålet ble endret.
- saved - Totalsummen brukeren har spart inntil tidspunktet sparemålet blir endret.
- sum - Sparemålets totalsum.

For å kunne vise brukeren hvor mye som er blitt spart i forhold til målet må dette beregnes på følgende måte:

```
1  function display(item) {
2    var totalDays = dayDiff(item.dateStart, item.dateEnd);
3    var totalDaysLeft = dayDiff(new Date(), item.dateEnd);
```

```
4       var display = ((totalDays - totalDaysLeft) * ((item.sum-item.saved) /
            totalDays));
5       return (display > item.sum) ? item.sum : display;
6     }
```

Kodeutsnitt 7.1: Status på sparemål

Variabelen ”display” finner først differansen mellom totalt antall dager for sparemålet og hvor mange dager som er igjen. Dette ganges videre med den gjennomsnittlige sparesummen som brukeren må spare hver dag.
Variabelen ”item.saved” vil kun være i bruk hvis sparemålet er endret. Da vil variabelen inneholde verdien brukeren har spart inntil sparemålet ble endret. Ellers vil denne verdien være 0. Dette blir forklart under 7.2.2 Endring av sparemål.

Brukeren får også tilgang til å endre eksisterende sparemål eller å legge inn nye. Dette vil føre brukeren til en undermodul kalt ”savings-new”. Her skriver brukeren inn data som navn, sum og dato. Kontrolleren vil da regne ut hvor mye som må spares hver dag, frem til sparemålets sluttdato. Brukeren vil hele tiden få oppdatert informasjon om hvor mye som må spares hver dag og over hvor mange dager. Denne informasjonen oppdateres synkront ved at brukeren enten endrer dato eller ønsket sparesum.

7.1.4 Settings

Den siste hovedmodulen kalles ”Settings”, og inneholder en samlet liste over funksjonalitet som brukes sjeldnere enn applikasjonens hovedfunksjonalitet. Dette innebærer innlogging og registrering av brukerkonto, kjøpshistorikk, kundeservice og informasjon om applikasjonen. Enkelte av disse undermodulene ble nedprioritert, og arbeid på dem startet sent i prosjektperioden. Derfor var disse fremdeles under utvikling ved prosjektslutt.

Settings-login er en av de ferdig implementerte undermodulene. Denne lar brukeren logge inn, eller registrere en ny konto. Registreringen skjer da i en egen undermodul kalt settings-register.

Settings-contact er også implementert, og er en undermodul med bankens kontaktinformasjon. Her får brukeren også se kart over Totens Sparebanks kontorer.

Undermodulen for kjøpshistorikk er en av de uferdige modulene. Denne skal samle historikken fra alle budsjettposter, og vise dem i en kronologisk rekkefølge, uavhengig av hvilken budsjettpost de tilhører. Denne undermodulen kalles ’settings-history’, og har som hensikt å gi en bredere oversikt over brukerens kjøp.

Informasjonsmodulen heter ’settings-about’. Dette er den siste undermodulen i Settings, og er også uferdig. Planen for denne modulen er å la den forklarer at applikasjonen er utviklet som et bachelorprosjekt på oppdrag fra Totens Sparebank. Den skal også inneholde en Personvernerklæring og eventuelt lisens.

7.2 Spesielle tilpasninger

Her presenteres løsningsforslag av utvalgt funksjonalitet som måtte løses på en spesiell eller uvanlig måte for oppnå ønsket virkemåte.

7.2.1 Gjentakende inntekt / utgift

Registrering av gjentagende transaksjoner var en funksjonalitet som bydde på en del problemer. Det første problemet dreide seg om måten brukeren skulle få velge hvor ofte transaksjonen skulle gjentas. Det ble vurdert å kun la brukeren velge hvor mange dager det skulle gå mellom hver transaksjon. Det ble imidlertid rask bestemt at brukeren måtte ha flere muligheter. Vi endte opp med en løsning hvor brukeren velger et intervall bestående av et tall og enten dag, uke, måned eller år. På denne måten kan for eksempel brukeren velge at transaksjonen skal forekomme på en gitt dato hver 4. måned.

Da brukergrensesnittet og mekanismen bak registreringen av transaksjonen var implementert var det behov for en tjeneste som kunne kjøre i bakgrunnen. Denne tjenesten måtte automatisk legge inn transaksjonene, etter hvert som dager passerte. For å spare ressurser måtte tjenesten kun kjøre så ofte som det var behov for. Dette ble løst ved at den selv sjekker når den sist kjørte sin hovedfunksjonalitet når brukeren åpner applikasjonen. Dersom den ikke har kjørt hovedfunksjonaliteten den gjeldende dagen, vil den dermed gjøre det.

Tjenestens hovedfunksjonalitet består av å gå igjennom brukerens gjentagende transaksjoner og legge til transaksjoner dersom dette er nødvendig. Tjenesten kan kun kjøre dersom applikasjonen er kjørende, så dersom brukeren ikke har brukt applikasjonen over lengre tid, kan det mangle flere transaksjoner av same type. Under vises et utdrag av tjenesten.

```
1     function addMissingTrans() {
2     angular.forEach(userModel.categories, function(category) {
3       angular.forEach(category.repeaters, function(repeater) {
4         var newDateString;
5         var newTransaction;
6         var nowTime = moment(moment().format("YYYY-MM-DD"));
7         var preferredDate = repeater.firstDate.split('-')[2];
8         var unitSplit = repeater.repeatEvery.split('s')[0];
9
10        // As long as the first date + the repeat interval is in the past, add a
              transaction.
11        var lastTransMoment = moment(repeater.lastTransaction);
12        var newDateMoment = moment(lastTransMoment.add(repeater.repeatNumber,
              repeater.repeatEvery));
13        while(moment(nowTime).diff(newDateMoment, repeater.repeatEvery, true) >=
              0) {
14
15          // If the date is something like 31, this if-sentence makes sure that
                there's no problem
16          //    for months with less days than 31.
17          if(newDateMoment.date() < preferredDate && unitSplit === "month") {
18            var endOfMonth = moment(newDateMoment).endOf(unitSplit);
19            newDateMoment.date(endOfMonth.date());
20          }
21          newDateString = newDateMoment.format("YYYY-MM-DD");
22          newTransaction = {
23            description: repeater.description,
24            sum: repeater.sum,
25            date: newDateString,
26            repeaterId: repeater.id
27          };
28          repeater.lastTransaction = newDateString;
29          userModel.addExpense(category, newTransaction, -1);
30
31          // Prepare a new date for further looping
32          lastTransMoment = moment(newDateString);
33          newDateMoment = moment(lastTransMoment).add(repeater.repeatNumber,
                repeater.repeatEvery);
```

```
34          }
35        });
36      });
37    }
```

Kodeutsnitt 7.2: Utdrag av transactionRepeater

Funksjonen addMissingTrans inneholder hovedfunksjonaliteten til tjenesten. Den går gjennom brukerens budsjettposter, og sjekker om de inneholder repeaterobjekter. Disse objektene inneholder informasjon om gjentagende transaksjoner, som blir brukt til å automatisk konstruere nye transaksjoner. Tjenesten må regne ut flere tidspunkter og -perioder, og for disse utregningene blir biblioteket MomentJS [28] tatt i bruk. Dette biblioteket inneholder blant annet funksjonalitet for sammenligning av tidspunkter og lar oss lettere manipulere dato-objekter.

I while-løkken som starter på linje 13 foregår registreringen av transaksjoner for en spesifikk repeater. På linjen over regnes det ut hvilken dato det skal registreres en ny transaksjon på. Dette gjøres ved å se på den siste transaksjonen, og legge til intervallet brukeren har valgt. While-løkken starter dersom det har gått lengre tid enn brukerens tidsintervall siden siste transaksjon.

If-setningen på linje 17 tar seg av problemet som oppstår når brukeren legger inn en gjentagende transaksjon som starter på slutten av en måned. Et eksempel på dette kan være at brukeren starter en gjentagende transaksjon 29. februar. I slike tilfeller ville resultatet blitt uforutsigbart. Tjenesten ville lagt inn transaksjoner 1. mars eller kun hvert fjerde år (skuddår), alt ettersom hvilken løsning vi hadde utviklet. Vi skrev denne if-setningen for å sikre at transaksjoner automatisk blir registrert den siste tilgjengelige datoen i måneden, i stedet for å flyte over i neste måned.

Etter datoen er satt, blir transaksjonen lagt til i model-en. Deretter blir en ny dato forberedt, og while-løkken starter eventuelt på nytt.

7.2.2 Endring av sparemål

I det en bruker endrer et allerede eksisterende sparemål vil det være ønskelig å kunne bevare data som allerede ligger i sparemålet.

Scenario: En bruker har lagt inn et sparemål for en sydentur med avreise 16.august 2016 hvor det trenger å spares 6 000 kr. Dette sparemålet ble registrert 4. mai 2016. Brukeren får videre beskjed om at det må spares 57,69 kr over 104 dager. Etter at brukeren har spart i 50 dager vil sparesummen være totalt 2884,50 kr. Brukeren finner så ut at denne ferieturen i realiteten kun vil koste 5 000 kr og går inn på applikasjonen for å endre totalsummen for dette sparemålet. Dette betyr at brukeren ikke trenger å spare like mye hver dag for å nå sitt mål.

For å løse dette problemet er det i tillegg til dato og sum, også blitt lagret en egen variabel kalt ”saved” tilknyttet hvert sparemål. Dette vil bli lagt inn i alle fremtidige kalkuleringer av sparemålet. Ved endring av sparemål vil altså summen som allerede er spart opp bli videreført så dette fungere som en startsum fra dagen brukeren endrer sparemålet. Startdato blir også endret til dagens dato. Hvis brukeren endrer sparemålet flere ganger etter hverandre vil ”saved” alltid være det totale oppsparte per dags dato. Brukeren kan ikke endre et sparemål til å være mindre enn den oppsparte summen, da dette ikke ville fungert i praksis.

7.2.3 Uttrekk av budsjettpostikoner

Modulen ”budget-new” viser frem alle ikoner som ikke er i bruk i brukerens budsjett. Oppdragsgiver hadde som designkrav at disse skulle implementeres. Problemet var at de kom i PNG format og gjorde dem vanskelig å bruke. De skalerte dårlig og passet ikke fargeprofilen vi hadde laget.

Vi ble nødt til å konvertere disse ikonene til SVG-format for å optimalisere bruken av dem. Å konvertere ikonene til SVG-format gav muligheten til å gjøre ikonene til sin egen CSS-klasse for å bruke de. Da kunne vi skalere ikonene slik vi ville uten å forringe utseendet og gi dem den fargen vi ønsket å bruke.

For å få til det hentes alle SVG-elementer fra prosjektets CSS-mappe hvor SVG-filene ligger. Siden SVG-filer har et XML-format kan det bruke JavaScripts innebygde DOM-parser og plukke ut alle elementene som trenges og for å tilordne riktig verdier til ikon-velgeren. Uten å måtte hardkode disse ikonene i kildekoden.

```
1     if (vm.newCats.length === 0) {  // Vi vil bare gjore dette kallet hvis det
          ikke eksisterer
2     // Hent undersett av alle ikonene
3    $http.get('lib/ionic/fonts/eikaChoice.svg').then(function (data) {
4     // Lag ny DOM-parser
5      var parser = new DOMParser();
6       // Parse dataen fra HTTP foresporselen
7      var doc = parser.parseFromString(data.data, "image/svg+xml");
8      // Lag ny array
9      var fontz = [];
10     for (var i = 0; i < doc.getElementsByTagName('glyph').length; i++) {
11        // For hvert element i arrayet, lag nytt objekt
12         fontz[i] = {};
13          // La hvert object ha en icon egenskap som begynner med icon-
14          //    og hent navnet til ikonet
15         fontz[i].icon = "icon-" + doc.getElementsByTagName('glyph')[i].
               getAttribute('glyph-name');
16     }
17     // Na har vi et array med objekter som inneholder navnet pa ikonet
18     vm.newCats = fontz;
19     // Na kan vi sette opp resten av innholdet i kategoriene
20     setUpNewCats();
21   });
22 }
```

Kodeutsnitt 7.3: Trekk ut navnet til alle ikonene

Pseudokoden til kodeutsnitt 7.3 kan forklares slik

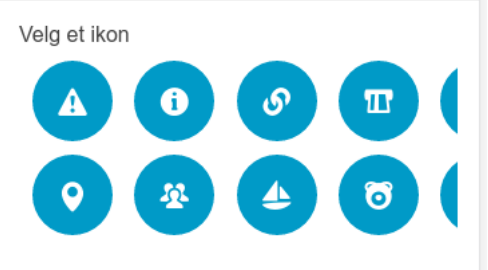
- Hvis newCats er tomt:
- Lag et tomt array
- Hent fila som har ikonene
- Parse fila med en DOMParser
- Gå igjennom alle elementene og gjør:
    - Legg til et nytt objekt i arrayet med egenskapen ”icon” med verdien til attributtet ”glyph-name” i det aktuelle elementet
- Sett newCats til å være det oppfylte arrayet
- Grupper kategoriene til groupFonts

```
1 <div class="row" ng-repeat="fontGroup in bdgnewCtrl.groupfonts">
2     <div class="col col-25" ng-repeat="font in fontGroup">
3         <button
4             class="button button-circle {{font.icon}}"
5             ng-class=" {'ready': bdgnewCtrl.ready(font.icon)}"
```

```
6               ng-click="bdgnewCtrl.setIcon(font.icon)">
7           </button>
8       </div>
9   </div>
```

Kodeutsnitt 7.4: Bruk av ikoner

Da kan vi benytte objektetene på view-et vårt slik som i kodeutsnitt 7.4. Legg merke til linje 5 som sier at <button> skal ha klasse-attributtet som er verdien til egenskapen ”icon” for å få frem ikonet.



Figur 16: Resultat av ikonene

Figur 16 viser resultatet, legg merke til her at det er bare deler av femte kolonne med ikoner som vises, dette for å tilby (Affordance) brukeren å scrolle for å se flere ikoner.

7.2.4 Sortering av budsjettposter

Sortering av budsjettposter er gjort på en spesiell måte på grunn av oppsettet av kategorier i databasen. Hver kategori er et eget dokument som får sin faste posisjon etter opprettelse. Når det gjelder sparemål er dette gjort annerledes, ved at det finnes en array av spare-objekter i databasen som kan reorganiseres.

```
1   function moveItem(item, fromIndex, toIndex) {
2     vm.cards.splice(fromIndex, 1);
3     vm.cards.splice(toIndex, 0, item);
4     userModel.changeOrder(item, fromIndex, toIndex, 'saving');
5   }
6   vm.moveItem = moveItem;
```

Kodeutsnitt 7.5: Funksjon for å reorganisere sparemål

Reorganiseringen gjøres først i brukerens view ved å ordne rekkefølgen i ”vm.cards” som inneholder alle sparemål. Videre oppdateres det i databasen på samme måte.

Reorganisering av budsjettposter kan ikke gjøres på samme måte. Siden alle kategorier er lagret som egne dokumenter i database er det ikke ønskelig å endre posisjon på disse da det lettere kan forårsake korrupt data og at applikasjonen kan slutte å fungere som normalt. Det er derfor innført en egen hjelpe-array ved navn ”order” for å løse dette problemet. Denne arrayen ligger også lagret i databasen og inneholder id til alle dokumenter som er lagret etter brukerens angitte rekkefølge. Alle kategoriers id er statisk og kan ikke endres på. Hvis brukeren ikke har angitt en egen rekkefølge vil denne arrayen være tom og ikke bli brukt. Når brukeren endrer rekkefølgen på sine budsjettposter, vil det ikke bli gjort noen endringer på selve budsjettpostene, men da i denne hjelpe-arrayen istedenfor.

Hjelpe-arrayen blir brukt når brukeren går inn på oversikten over sine budsjettposer eller når det blir gjort andre endringer som å slette eller endre kategorier. Da blir funksjonen nedenfor kjørt.

```
1  function sortCategories(objects, order) {
2    var newOrder = [];
3      for (var i = 0; i <= order.length-1; i++) {
4        for (var j = 0; j <= order.length-1; j++) {
5          if (order[i] === objects[j]._id) {
6            newOrder.push(objects[j]);
7          }
8        }
9      }
10     // If there is categories in 'object' that is not listed in 'order'
11     if (objects.length > order.length) {
12       for (var k = order.length - 1; k <= objects.length - 1; k++) {
13         newOrder.push(objects[k]);
14       }
15     }
16   return newOrder;
17 }
```

Kodeutsnitt 7.6: Funksjon for å reorganisere budsjettposter

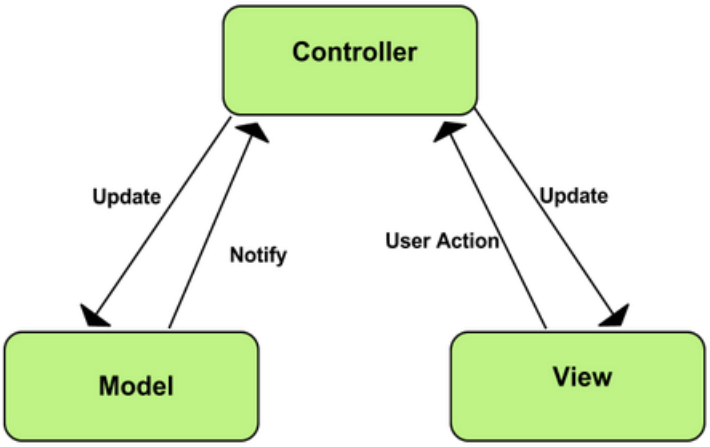
Funksjonen tar inn to parametre:

1. Array med kategori-objekter (objects)
2. Array med organiseringen av kategoriene listet etter id (order)

Funksjonen starter med to for-løkker som går igjennom begge arrayer og legger opp alle kategoriene(inkludert data) etter rekkefølgen til 'order'. Dette blir returnert i slutten av funksjonen. Hvis det skulle være objekter som mangler i "order", så vil koden på linjen 18-22 kunne løse det ved å legge det bakerst i arrayen. Dette for å inkludere eventuelle nye budsjettposter som blir lagt inn i ettertid.

7.3 Model, View, Controller

Vi benytter oss av et MVC pattern. Det er mange varianter av implementasjon av MVC patternet og vi har benyttet oss Google sin beskrivelse av MVC [29].



Figur 17: Google MVC

Figur 17 er hentet fra sidene til Google. Det går i hovedtrekk ut på å la kontrolleren være et bindeledd mellom modellen og view-et. Noe som var viktig for vår del da kontrolleren ofte måtte behandle rådata fra modellen før det ble oppdatert på view-et

7.3.1 View

Vi holder orden på view-ene våre ved hjelp av noe som heter history-stack. Denne er implementert i AngularJS som modulen ”ui.router”.

Den lager en beholder som holder på sider(views). Hvor mange sider en history-stack har til enhver tid kalles en tilstand. Etterhvert som brukeren navigerer seg fremover, ved å trykke en knapp som tar brukeren til en annen side, vil de sidene legge seg på toppen av history-stacken. Kontrolleren til den siden vil lastes inn og benyttes. Når brukeren trykker på tilbakeknappen eller når aktiviteten til brukeren er ferdig, vil siden fjernes fra history-stacken.

Siden både view-et og kontrolleren for den siden brukeren kommer tilbake til, allerede finnes på history-stacken vil de ikke lastes inn på nytt, men bare komme i fokus igjen.

Det er mulig å legge til flere history-stacker. Det gjør det mulig å hoppe fra en history-stack til en annen. Fordelen med det er at alle stackene bevarer sin tilstand uavhengig av hvilken history-stack som er den aktive. History-stackene konfigureres ved å angi navn og angi hvilke sider den kan inneholde. Disse konfigurasjonsfilene ender med ”.states” i prosjektet

```
1  angular.module('budget.home')
2    .config(function($stateProvider) {
3      $stateProvider
4        .state('tabs.newExpense', {
5          url: '/newExpense',
6          params: {
7            isIncome: false,
8            data: null
9          },
10         views: {
11           "home-full": {
12             templateUrl: 'transaction/transaction.html',
13             controller: 'NewTransactionController',
14             controllerAs: 'vm'
15           }
16         }
17       })
18       .state('tabs.newIncome', {
19         url: '/newIncome',
20         params: {
21           isIncome: true,
22           data: null
23         },
24         views: {
25           "home-full": {
26             templateUrl: 'transaction/transaction.html',
27             controller: 'NewTransactionController',
28             controllerAs: 'vm'
29           }
30         }
31       });
32   });
```

Kodeutsnitt 7.7: Konfigurering av history-stacker

Kodeutsnitt 7.7 viser hvordan man konfigurerer en history-stack. I dette tilfellet er history-stacken med navn ”home-full” konfigurert. Den er konfigurert på følgende måte: Funksjonen som $stateProvider kaller er .state, den beskriver et tilstandsobjekt som tar to parametere. Navnet til tilstandsobjektet og et objekt som har egenskapene til tilstandsobjektet. Parameterobjektet

inneholder følgende egenskaper

- **url:** url'en til siden, som vises i browseren.
- **params:** Parametere som siden skal motta. Disse er standardverdier og kan endres
- **views:** Egenskapene til history-stacken siden skal legge sed på.
    - **Navnet til history-stacken:** Objekt med egenskaper
        - **templateUrl:** Hvor koden til siden befinner seg
        - **controller:** Hva slags kontroller skal den benytte seg av
        - **controllerAs:** Hvilket navn siden skal assosiere controlleren med

Applikasjonen benytter fire forskjellige history-stacker. Disse har vi kalt

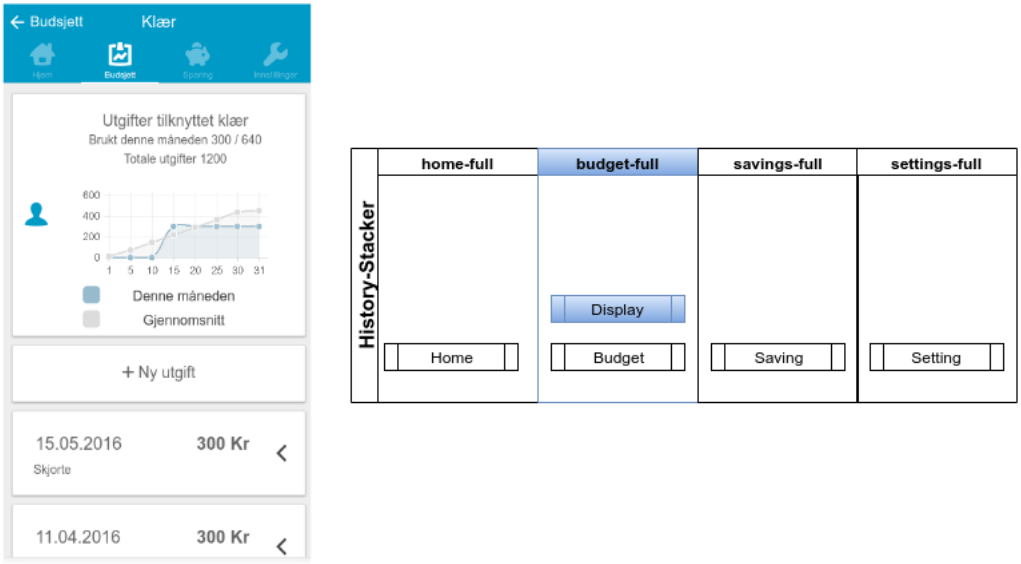
- **Home** Stack-navn: "home-full"
- **Budget** Stack-navn: "budget-full"
- **Saving** Stack-navn: "savings-full"
- **Setting** Stack-navn: "settings-full"

I view-et velger man hvilken history-stack som skal være aktiv ved å klikke på "tab-en" som representerer den aktuelle history-stacken.

```
<ion-tabs class="tabs-positive">
  <ion-tab title="Hjem" icon-on="icon-bolig" icon-off="icon-bolig" href="#/tabs/
      home">
    <ion-nav-view name="home-full"></ion-nav-view>
  </ion-tab>
  <ion-tab title="Budsjett"
           icon-on="icon-budsjett"
           icon-off="icon-budsjett"
           href="#/tabs/budget"
           ng-controller="BudgetController as bdgCtrl"
           badge="bdgCtrl.getBadgeCount()"
           badge-style="badge-assertive">
    <ion-nav-view name="budget-full"></ion-nav-view>
  </ion-tab>
  <ion-tab title="Sparing"
           icon-on="icon-sparing"
           icon-off="icon-sparing"
           href="#/tabs/savings">
    <ion-nav-view name="savings-full"></ion-nav-view>
  </ion-tab>
  <ion-tab title="Innstillinger"
          icon-on="icon-verktoy"
          icon-off="icon-verktoy"
          href="#/tabs/settings">
    <ion-nav-view name="settings-full"></ion-nav-view>
  </ion-tab>
</ion-tabs>
```

Kodeutsnitt 7.8: Implemetering av history-stacker

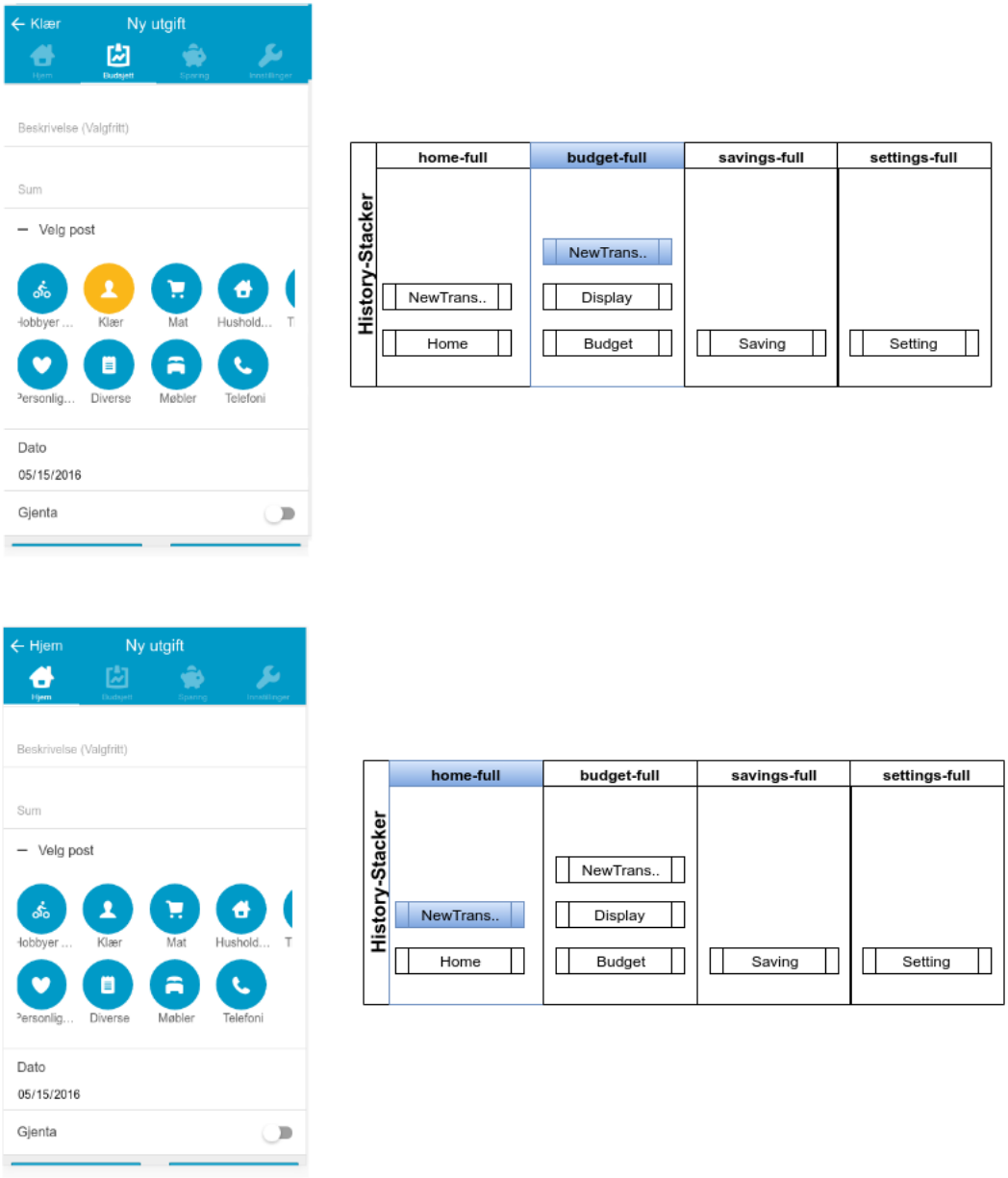
View-et fordeler de forskjellige stackene inn i forskjellige "ion-nav-view" elementer hvor attributten "name" er lik stack-navnet. Det forteller også at den siden som er øverst i history-stacken skal fylle elementet "ion-nav-view" med de elementene som finnes i "templateUrl" Vi ser også ut i fra koden at alle "ion-nav-view" elementene ligger i "ion-tab" elementer. Det gjør det mulig å navigere mellom history-stacken ved hjelp av "tabs"



Figur 18: History-stack

Figur 18 viser hvordan brukeren kan se hvilken history-stack som er aktiv ved at ”Tab-en” Budsjett er fremhevet og den øverste siden på history-stacken er Display-siden. Brukeren kan fortsette navigeringen herfra ved å trykke på ”Ny Utgift”. Det vil legge siden NewTransaction ovenfor siden Display og bli den aktive.

Fra kodeutsnitt 7.7 kan vi se at siden NewTransaction også kan inngå i history-stacken til ”home-full”. Hva vil skje dersom begge history-stackene holder på samme modul?



Figur 19: Samme side i forskjellige history-stacker

Figur 19 viser hvordan to like views kan legges i to forskjellige history-stacker. Dette er uproblematisk da tilstanden til history-stackene opprettholdes. Brukeren kan se hvor han har navigert ved å se hvilken av history-stackene som er den aktive, og hvor han kommer når han trykker på tilbakeknappen. Dette gjør at hele modulen hvor siden og kontrolleren ligger i kan gjenbrukes.

7.3.2 Controller

En kontroller er tilknyttet et view i applikasjonen.

Ansvaret til kontrolleren er:

- Behandle rådata fra modellen slik at view-et kan presentere fornuftig data til brukeren

- Validering av data fra brukeren
- Motta oppdateringer fra modellen og behandle oppdaterte data
- Tilby view-et funksjonalitet

Scope

Kontrollerene opererer på et klart definert og avgrenset område. Det er innholdet i templateUrl egenskapen i tilstandsobjektet (se figur 7.7) som definerer dette området. Dette området kalles for scopet til kontrolleren. Når kontrolleren instansieres bindes view-et sine variabler med kontrolleren sine variabler.

State Parametere

I samme modul som tilbyr history-stack (se kapittel 7.3.1 View) får applikasjonen også tjenestene "$stateParams"og "$state".

Den gjør det mulig for applikasjonen og gå fra en tilstand til en annen med metoden

```
$state.go('state',stateParams)
```

Hvor det første parameteret er navnet til tilstandsobjektet som beskrevet i kodeutsnitt 7.7 og det andre er parameterobjektet som skal sendes til tilstandsobjektet. Det er valgfritt å kalle på funksjonen med parameterobjek, men hvis $state.go kalles med parameterobjekt, vil dette overskrive eventuelle standardverdier som blir definert i params egenskapen til tilstandsobjektet (se kodeutsnitt 7.7).

```
/**
 * Navigate the user to the edit expense page
 * @param {object} expense The expense to edit
 * @returns {undefined}
   */
function editExpense(expense) {
  var stateParams = {
    data: {
      expense: expense,
      category: vm.item
    },
    isIncome: vm.item.isIncome
  };
  $state.go('tabs.edit-expense',stateParams);
}
```

Kodeutsnitt 7.9: Kall til en annen side med parametere

Kodeutsnitt 7.9 er fra siden Display. Denne funksjonen blir brukt av både ”Ny utgift” knappen og knappen ”Endre utgift” som vises ved å trykke på pila i et utgiftskort (se figur 18). Forskjellen er at ”Ny utgift” sender med null som parameter, mens ”Endre utgift” sender med et eksisterende utgiftsobjektet som parameter. Begge kallene sender med budsjettposten som brukeren ser detaljer fra. Når funksjonen kalles havner det nye tilstandsobjektet på toppen av history-stacken, dermed blir dette den aktive siden, og parameterobjektet sendes til det nye tilstandsobjektet som mottar det når kontrolleren blir instansiert.

```
function NewTransactionController(...) {
    vm = this;
    ...
    if ($stateParams.data != null) {
        if ($stateParams.data.expense !== null) {
          vm.transaction = $stateParams.data.expense;
          vm.transaction.date = new Date(vm.transaction.date);
        }
        vm.budgetPost = $stateParams.data.category;
```

```
10          oldPost = $stateParams.data.category;
11          vm.isIncome = $stateParams.data.category.income;
12          index = vm.budgetPost.expenses.indexOf(vm.transaction);
13        }
14      }
```

Kodeutsnitt 7.10: Mottagende side tar imot parametere

Kodeutsnittt 7.10 mottar parameterobjektet og binder disse til scopet som blir definert i linje 2. Det sørger for at variablene også blir bundet til view-et og presentert for brukeren.

```
1  <ion-item class="item item-input" ng-class="{'ooops': vm.error === 'descr'}">
2      <label class="item-floating-label" style="width:100%">
3        <span class="input-label">{{ 'description_message' | translate }}</span>
4        <input type="text"
5               placeholder="{{ 'description_message' | translate }} ({{ '
                    optional_message' | translate }})"
6               ng-model="vm.transaction.description">
7      </label>
8  </ion-item>
9    <!-- The sum input -->
10   <ion-item class="item item-input" ng-class="{'ooops': vm.error === 'sum'}">
11     <label class="item-floating-label" style="width:100%">
12       <span class="input-label">{{ 'amount_message' | translate }}</span>
13       <input type="tel"
14              placeholder="{{ 'sum_message' | translate }}"
15              ng-model="vm.transaction.sum">
16     </label>
17   </ion-item>
18   ...
19   <div class="col col-25" ng-repeat="post in postGroup" style="margin: 0 auto;
         text-align:center;">
20       <button
21         class="button button-circle {{ post.icon }}"
22         ng-click="vm.setPost(post)"
23         ng-class="{'ready': vm.selected(post._id)}">
24       </button>
25       <br> <p> {{ post.name | translate }} </p>
26     </div>
```

Kodeutsnitt 7.11: Scope variable bundet til view-et

View-et bruker i linje 6 og 15 i kodeutsnitt 7.11 scope-variable som blir presentert til brukrere. Linje 23 forteller at hvis scope-funksjonen selected returnerer true, altså at ikonet er valgt, skal det få en annen klasse som forandrer fargen på ikonet som vises. Det gjenspeiles i figur 19 der vi ser at et ikon har fått en annen farge. Applikasjonen vil virke mer intelligent ved at den gjetter at brukeren vil komme til å legge til et nytt billag i den budsjettposten brukeren kom fra.

7.3.3 Model

View og kontroller har sterk kobling til hverandre. De er bundet av scopet og har en to-veis innflytelse på hverandre. Når det er sagt er kontrolleren avhengig av å få data den kan kontrollere. Det blir gitt av modellen. Siden modellen vår skal være lik for hele applikasjonen trenger vi å sentralisere den. Det har vi gjort ved å tilby en tjeneste som gir tilgang til modellen og operasjoner som har innvirkning på den. Denne tjenesten har vi kalt ”userModel” (se kapittel 7.4 Kart over tjenester). Det er veldig viktig at endringer som skjer på modellen også skjer på databasen og hvis brukeren er logget på backend-serveren skal endringene også speiles på den.

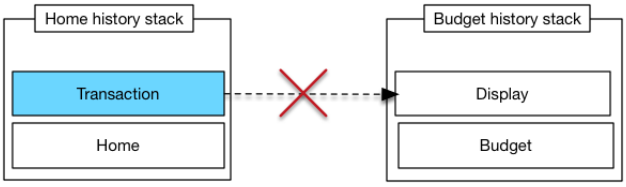
Hvordan holde model-en oppdatert

I applikasjonen er noe av det viktigste å holde modellen vår oppdatert på alle view-ene. Fra kapittelet 7.3.1 View står det beskrevet at objekter bare instansieres når de blir lagt på history-

stacken. Fra kapittel 7.3.2 Controller står det at mange kontrollere behandler data fra modellen når de instansieres.

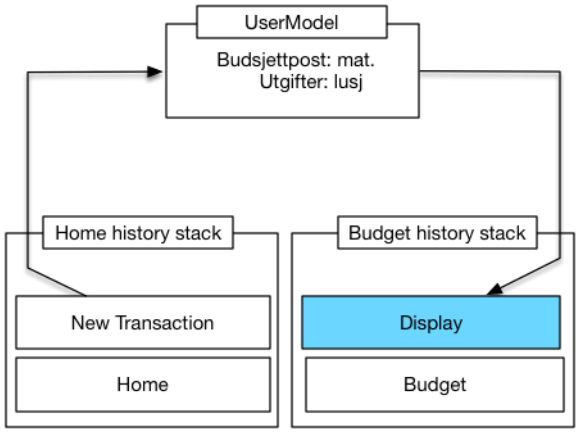
Scenario - Hva skjer i et view hvis endringer i modellen skjer på en annen history-stack

La oss si en bruker ser på oversikten over alle utgiftene sine i budsjettposten Mat gjennom siden Display. Brukeren har da trykket på ”tab-en” Budsjett. Han går så til ”tab-en” Hjem og trykker på knappen ”Ny Utgift”. Der legger han til billaget ”Lunsj” for i dag i budsjettposten Mat. Han trykker deretter på ”tab-en” Budsjett. Brukeren forventer da å se at det nye billaget han nettopp har lagt inn skal vises på siden Display som er den aktive siden i history-stacken til Budsjett.



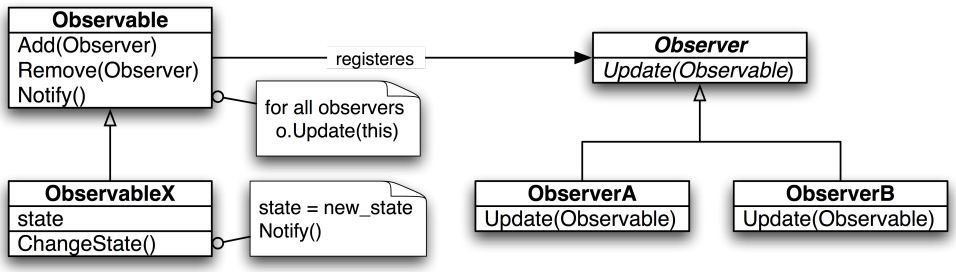
Figur 20: History-stack uten koblinger

Problemet er at det er ingen koblinger mellom history-stackene, de vet ikke at noen andre eksisterer. Så hvordan skal et view oppdatere som er på en annen history-stack hvis de ikke vet om hverandre.



Figur 21: Modellen har ansvaret for oppdatering

Løsningen er å la history-stackene forstsatt være uvitende om hverandre og la modelltjenesten fortelle til alle objektene på alle history-stakene som er avhengig av å vite om forandringer i modellen, at en forandring har skjedd. Applikasjonen implementerer et *”Observer Pattern”* som sørger for dette. Et vanlig diagram av dette patternet kan se noe slik ut



Figur 22: Observer Pattern
[30]

Hver modul(ObserverA,B..) som skal lytte etter forandringer på modellen legger sin observer(heretter listener) til i userModel(Observable) sin addL metode(Add(Observer)) som registrerer listeners. Hver gang en funksjon som endrer modellen kalles(ChangeState()) så kjøres notifyListeners(Notify()). Da blir alle listeners kalt(Update(newModel)) med den nye modellen.

Siden userModel-tjenesten bare blir instansiert en gang, mens kontrollere blir instansiert hver gang det legges på history-stacken må applikasjonen sjekke om navnet til funksjonen allerede finnes i listen over allerede registrerte listeners. Vi oppdaget at ved å bare dytte funksjonen bakerst i array-et, ville dette potensielt skape en minnelekasje og/eller skape unødvendig henging av systemet siden kontrollere blir instansiert hver gang et view opprettes. Da ville man få flere like funksjoner i array-et, og det vil man unngå. Derfor er det et krav at alle listeners har unike navn.

```
1
2   function addL(callBack) {
3     // Hent indeksen til funksjonsnavnet i arrayet
4     var index = listenersName.indexOf(callBack.name)
5     if (angular.isFunction(callBack)){
6     // Hvis den ikke finner navnet skal det legges bakerst i arrayet
7       if (index === -1) {
8         listeners.push(callBack);
9         listenersName.push(callBack.name);
10      }
11      // Hvis den finnes skal den overskrive gammel listener
12      else {
13        listeners[index] = callBack;
14      }
15    }
16  }
```

Kodeutsnitt 7.12: Idempotent funksjon for registrering av listeners

Kodeutsnitt 7.12 sjekker navnet til funksjonen som skal registreres, hvis den finnes i arrayet som holder på navnene til funksjonene lagres indeksen. Deretter overskriver den funksjonen som ligger i arrayet med listeners som er på indeksnummeret som er lagret. Hvis den ikke finner navnet dyttes funksjonen på arrayet. Dette gjør registrering av listeners til en idempotent funksjon ved at den kan registrere en listener så mange ganger den vil, men listeneren vil bare bli lagt til hvis den ikke er registrert fra før.

```
1   function notifyListeners() {
2     angular.forEach(listeners, function(listener) {
3       listener(model);
4     })
5   }
```

Kodeutsnitt 7.13: Oppdatering av model

Hver hver gang det skjer en forandring på modellen blir notifyListener kalt. NotifyListener går igjennom alle funksjonene i modelChangeListeners array-et og kaller hver funksjon med den oppdaterte modellen som parameter.

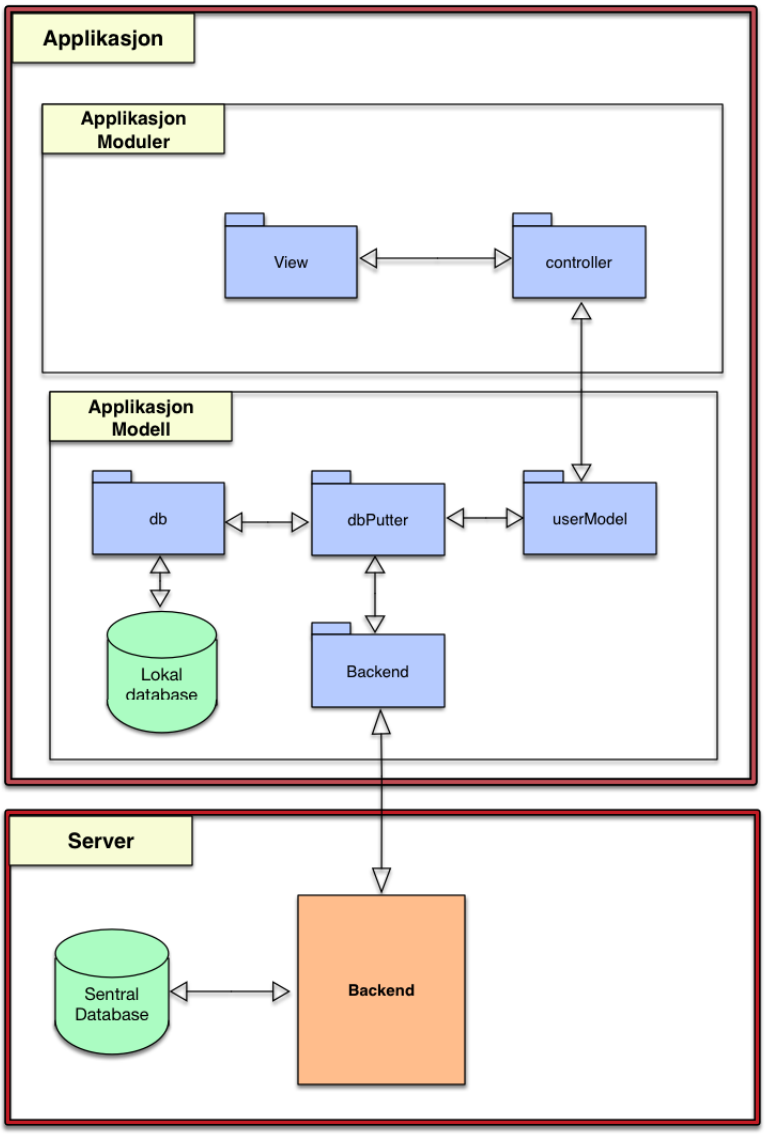
Nå blir det opp til hver enkel side på history-stacken å gjøre hva den vil med den oppdaterte modellen.

```
1  function bdgDsplistener(newModel) {
2    var id = vm.item._id;
3    vm.item = null;
4    angular.forEach(newModel.categories, function(category) {
5      if (id === category._id) {
6        vm.item = category;
7      }
8    });
9    // I tilfelle forandringen var a fjerne budsjettposten
10   // blir brukeren navigert tilbake til forrige side
11   if (vm.item === null) {
12     $ionicHistory.goBack();
13   }
14   else {
15     vm.data = getChartData(getChartSetupData());
16   }
17 }
18 userModel.addL(bdgDsplistener);   // Registrer listener funksjonen til
       userModel
```

Kodeutsnitt 7.14: Callback funksjon

Kodeutsnitt 7.14 viser at den finner tilbake til den budsjettposten den viste detaljer fra og oppdaterer den. Hvis endringen var å slette hele budsjettposten vil brukeren bli navigert tilbake til forrige side

Ansvaret til userModel



Figur 23: MVC lagdeling

I tillegg til å holde model-en vår konsistent, trenger vi også å forsikre oss at dataen som finnes i modellen er lik dataen som finnes på den lokale databasen. Vi vil ikke la modulene gjøre endringer på databasen som ikke er i tråd med de endringene som er gjort på modellen. Derfor har vi lagt oss på en lagdeling hvor view-et ligger på toppen. Controller-en ligger under og modellen ligger i bunnen. Under modellen finner vi mer lagdeling der vi har den lokale databasetjenesten. Siden det er userModel som har ansvaret for å endre dataene på model-en er det da også naturlig å la den være den eneste som har et interface mot den lokale databasetjenesten. På den måten sikrer vi at data fra databasen er det samme som data fra userModel.

```
1 function alterCategories(item) {
2     var found = false;
3     if (angular.isDefined(model.order[0])) {
4       alterOrders(item);
5     }
6     for (var i = 0; i < model.categories.length; i++) {
```

```
7       if (model.categories[i]._id === item._id) {
8         model.categories[i] = item;
9         found = true;
10        dbPutter.addCat(item,changeRevision,i);
11        break;
12      }
13    }
14    if (!found) {
15      model.categories.push(item);
16      dbPutter.addCat(item,changeRevision,model.categories.length -1);
17    }
18  }
```

Kodeutsnitt 7.15: Oppdater budsjettposter

Kodeutsnitt 7.15 er userModel sin funksjon for å endre eller legge til en budsjettpost. Den prøver å finne budsjettposten for å overskrive den som allerede ligger der eller legge til ny hvis den ikke finner den. Uansett så kaller den på dbPutter sin addCat funksjon.

```
1   function addCat(newCat,callback,index) {
2     db.get(newCat._id, function (err, data) {
3       if (!err) {
4         db.put(newCat, data._rev, function(err, resp){
5           if (err) {
6             $log.error(err);
7           }
8           else {
9             push();
10            callback(resp.rev,index);
11          }
12        });
13      }
14      else {
15        db.put(newCat,function(err, res) {
16          if (err) {
17            $log.error(err);
18          }
19          else {
20            push();
21            callback(res.rev, index);
22          }
23        });
24      }
25    });
26  }
```

Kodeutsnitt 7.16: Legg til ny budsjettpost i lokal database

Kodeutsnitt 7.16 prøver å hente ut dokumentet med id som budsjettposten har. Hvis det finnes vil den skrive over dokumentet i databasen og kalle callback funksjonen med nytt revisjonsnummer. Hvis det ikke finner dokumentet vi det bli lagret som nytt dokument, og fortsatt kalle på callback funksjonen med det første revisjonsnummeret til dokumentet. Funksjonen push er en funksjon som sjekker om brukeren er pålogget og dytter endringene til backend. Hvis brukeren ikke er online blir det satt et dirty bit. Hvis det er satt vil dbPutter prøve å pushe databasen til backend så snart brukeren er på nett igjen.

```
1     function changeRevision(rev,index) {
2     model.categories[index]._rev = rev;
3     notifyListeners();
4   }
```

Kodeutsnitt 7.17: Callback funksjonen

Kodeutsnitt 7.17 mottar revisjonsnummeret og indeksen til det oppdaterte dokumentet og endrer det i sin egen modell. Deretter sender det ut et kall til alle listeners som lytter på forandringer på modellen slik at riktig revisjonsnummer blir benyttet.

Ansvaret til dbPutter

Når vi har gjort oppdateringer på den lokale databasen må vi si ifra til backend-tjenesten at endringer er gjort lokalt og vi trenger å oppdatere den sentrale databasen. Backend-tjenesten sender en forespørsel over nettet til den sentrale backend-tjenesten (se kapittel 6.3 Backend) Den forsikrer seg at forspørselen kommer fra en bruker som er pålogget og sender forespørselen til riktig sentral database. Vi må også fortelle userModel at lagring til den lokale databasen var vellykket. Dette gjøres med en callback funksjon.

7.4 Kart over tjenester

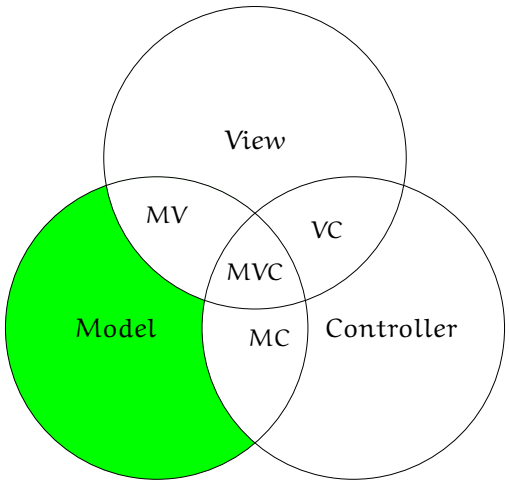
Vi definerer en tjeneste i vår applikasjon etter følgende kriterier:

- Uten view
- Brukes av andre moduler og/eller tjenester
- Har et definert ansvarsområde
- Er gjenbrukbar

Vi benytter tjenester i applikasjonen på de stedene hvor det er naturlig å dele opp ansvarsområder, og de stedene i applikasjonen hvor det utføres gjør de samme operasjonene på forskjellige steder. Eksempel på det siste kan være validering av utfylte skjema. Tjenestene kan deles inn i hvor i MVC-patternet de yter sin tjeneste. Alle tjenestene er avhengige av modulen 'angular' og/eller modulen 'ionic'. Det er ikke alle felter i MVC-diagrammet som trenger å ha en egen tjeneste.

Model

Tjenester herunder yter bare tjenester som går på endringer på modellen. Disse er:



Figur 24: Model

db		
Hovedansvar	Tilby et grensesnitt for lokal database	
Avhengigheter	PouchDB	
Tilbyr	db	Grensesnitt for lokal database

Tabell 3: Db.

backend		
Hovedansvar	Grensesnitt mot sentral server	
Avhengigheter	Ingen	
Tilbyr	register	Registrer en bruker mot sentral server
	login	Sender en logg inn forespørsel til sentral server
	connect	Kobler brukeren opp til en socket til den sentrale serveren
	disconnect	Kobler brukeren fra socketen mot sentral server
	shareReq	Sender en dele-forespørsel til sentral server
	getFbLongToken	Sender en forespørsel til serveren om å be om en token med lang holdbarhet
	shareResponse	Sender et svar til en dele-forespørsel til sentral server
	pullDb	Henter databaseinnhold fra sentral server
	pushDb	Sender databaseinnhold til sentral server

Tabell 4: Backend

dbPutter		
Hovedansvar	Ta seg av databaseoperasjoner	
Avhengigheter	db og backend	
Tilbyr	addSaving	Legge til et sparemål i lokal database
	deleteSaving	Fjerne et sparemål i lokal database
	changeSaving	Endre et sparemål i lokal database
	addCat	Legge til en kategori i lokal database
	addExpense	Legge til en transaksjon i lokal database
	init	Henter ut alle brukerdata og kategorier fra lokal database
	changeOrder	Lagrer endringer i kategorirekkefølgen i lokal database
	setLastCheck	Lagrer dagens dato i den lokale databasen
	addShareRequest	Lagrer en deleforespørslen i den lokale databasen
	userLogin	Logger en bruker på med hjelp av serveren
	registerUser	Registrerer en bruker mot serveren
	removeShareRequest	Fjerner en deleforespørsel fra lokal database
	registerPromiseCallback	Registrerer en callback som lytter på forandringer i brukerens innloggingsstatus
	addSifoBudget	Legger til SIFO-kategorier til lokal database
	deletePost	Sletter en kategori fra lokal database
	logOut	Logger en bruker av serveren
	getBudgetItems	Henter ut alle budsjettpostene
	addSRListener	Registrerer callback funksjon til backend

Tabell 5: DbPutter

sifo		
Hovedansvar	Legge til budsjettposter fra standardbudsjettet til SIFO	
Avhengigheter	Ingen	
Tilbyr	getBudget	Henter budsjettposter ut i fra brukerens kriterier

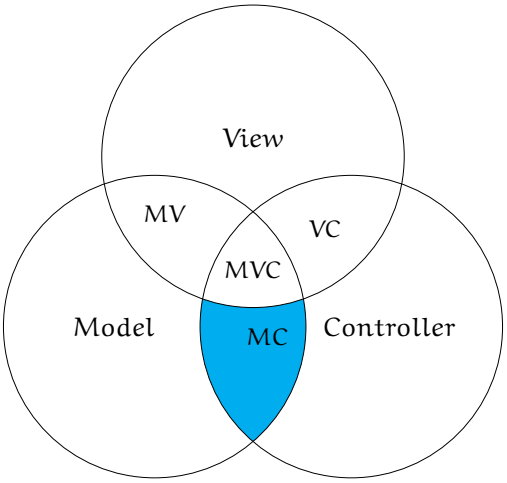
Tabell 6: Sifo

transactionRepeater		
Hovedansvar	Oppdatere gjentagende utgifter	
Avhengigheter	userModel	
Tilbyr	onResume	Sjekker om repeterende utgifter mangler ved gjenoppstart av applikasjonen
	addMissingTrans	Legger til manglende utgifter i modellen

Tabell 7: TransactionRepeater

Model/Controller

Disse tjenester går både på endringer i model-en og gir kontrolleren funksjonaliteter



Figur 25: Model og Controller

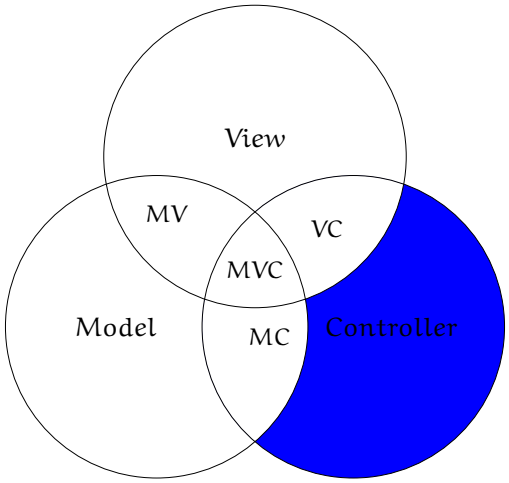
Tabell 8: UserModel.

userModel	
Hovedansvar	Holde på data om brukeren, herunder sparemål, budsjett, og deleforespørsler.
Avhengigheter	dbPutter
Tilbyr	
alterCategories	Legg til eller endre en budsjettpost
init	Initierer brukeren og henter brukerinformasjon fra databasen
addL	Legger til en funksjon som skal lytte på endringer i modellen
getCats	Returnerer et Promise som blir resolvet når modellen er lastet
addExpense	Legger til eller endrer en transaksjon i modellen
deleteCategory	Fjerner en budsjettpost fra model-en
introCategories	Legger til budsjettposter som er lagt til fra oppstart
getCurrent	Returnerer den totale summen fra en budsjettpost for inneværende måned
removeExpense	Fjerner en transaksjon fra en budsjettpost
changeOrder	Endre rekkefølge på budsjettposter eller sparemål
addRepeatingTrans	Legger til en gjenntagende utgift
registerUser	Registrerer en bruker og returnerer et promise om at brukeren blir registrert
userLogin	Logger en bruker på med epost og passord og returnerer et promise om at brukeren får logget inn
addSaving	Lagrer et sparemål
changeSaving	Forandrer et sparemål
setLastUpdate	Setter dagens dato i modellen
getLastUpdate	Henter dagens dato fra modellen
getCredentials	Henter informasjon om brukeren og registrerer callback
logOut	Logger bruker av serveren
update	Oppdaterer modellen når brukeren er tilknyttet serveren
removeShareRequest	Fjerner en deleforespørsel
refreshBudgetItems	Oppdaterer budsjettpostene

addSRListener	Registrerer deleforespørsel listener
deleteSaving	Sletter et sparemål
userModel slutt	

Controller

Disse tjenestene tilbyr tjenester for bruk i kontrolleren



Figur 26: Controller

popupContent		
Hovedansvar	Oversette innholdet som skal til en popup-melding	
Avhengigheter	Ingen	
Tilbyr	createContent	Oversetter innhold som skal til en popup-melding

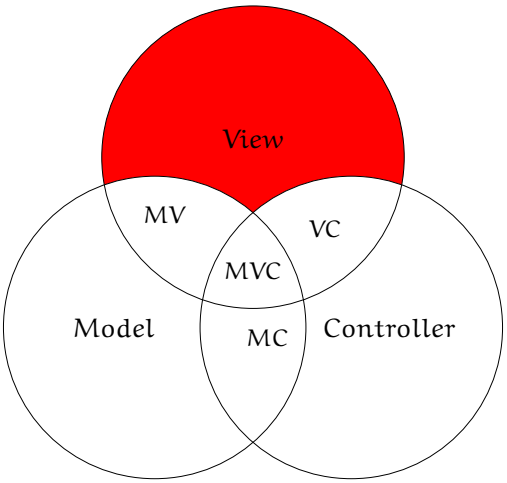
Tabell 9: PopupContent

inputChecker		
Hovedansvar	Sørge for at verdier brukeren skriver inn er korrekte	
Avhengigheter	Ingen	
Tilbyr	textChecker	Sjekker at verdien av en tekst ikke er for lang
	sumChecker	Sjekker at verdien av et tall er et tall og ikke for stort

Tabell 10: InputChecker

View

Disse tjenestene har direkte innvirkning på view-et



Figur 27: View

numberFormat		
Hovedansvar	Vise et tall i formatet som er riktig for enten norsk eller engelsk og vise antall desimaltall som angitt	
Avhengigheter	Ingen	
Tilbyr	numberFormat	En funksjon benyttet som et filter i HTML. Returnerer riktig tallformat

Tabell 11: NumberFormat

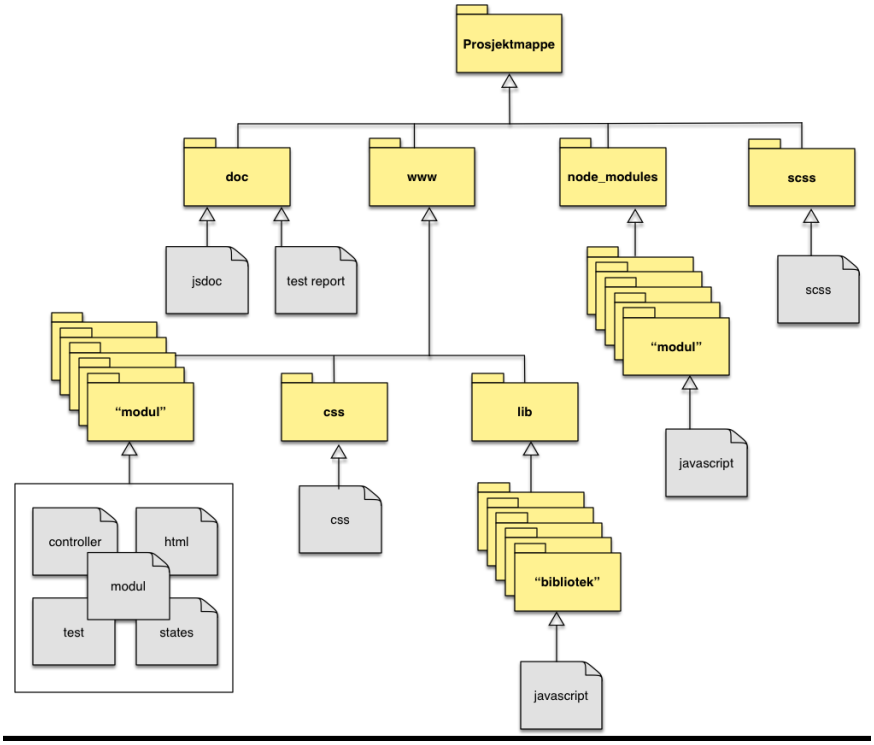
7.5 Filstruktur

Vi har benyttet oss av LIFT [31] prinsippet i vår filstruktur. Det står for

***L**ocate your code quickly.* Vi følger dette ved at vi begrenser dybden til filene med 4 nivåer som maksimum. Alle filene som vi benytter i applikasjonen befinner seg i de to første nivåene <prosjekt-rot>/www/. På den måten er effektiv maksimumsdypde på 2 nivåer. Dette gjør det lettere å lett finne frem til riktig fil raskere.

***I**dentify code at a glance.* Her har vi brukt dette prinsippet om å skjønne med en gang hva en fil gjør på flere måter.

- Mappenavnene indikerer omfanget til filene som ligger inni.
- Lenking av spesifikt ansvar til filene punktsepareres slik <omfang>.<ansvar for filen>.<type> For filer med type .html angis ikke ansvaret da det gir seg selv (se T - Try to keep it DRY).
- Vi navngir mappene og filene med hvilken view-stack de naturlig faller under. Hvis dette ikke er naturlig utelates dette. Her benytter vi som oftes en strekseparert navngiving i omfanget.



Figur 28: Overblikk over mappestruktur i prosjektet

***F**lattest struckture you possibly can* Hver mappe inneholder mellom 3 og 6 filer hver. Unntaket her her ../lib/ som er applikasjonsavhengighetene og ../service/ som inneholder våre applikasjonsjenester. En typisk struktur for en mappe er som følger:

- **omfang.module.js** Oppretter en modul. Her konfigureres også modulen hvis det trengs.
- **omfang.controller.js** Oppretter en kontroller for et scope.
- **omfang.spec.js** Testfil for modulen.
- **omfang.states.js** Fil som har ansvar for navigering.
- **omfang.html** View-et til modulen.

***T**ry to keep it DRY* Vi kan argumenter at vi ikke gjør dette ved at filene har samme navn som mappene bare med endinger, men ved å gi samme navn til filene trenger vi ikke å se på mappen hva slags mappe vi står i. I tillegg kan flere moduler ha samme type funksjonalitet som kan gi forvirring. Vi etterlever dette ved at vi for eksempel ikke kaller html-filer for omfang.view.html.

8 Design

8.1 Gui

8.1.1 GUI-prinsipper

Gjennom hele prosjektperioden var det viktig at brukergrensesnittet ble enkelt å forholde seg til, samtidig som det skulle ha et moderne preg. Derfor har vi forholdt oss til mange av Alan Coopers designprinsipper fra boka About Face [10]. Disse er:

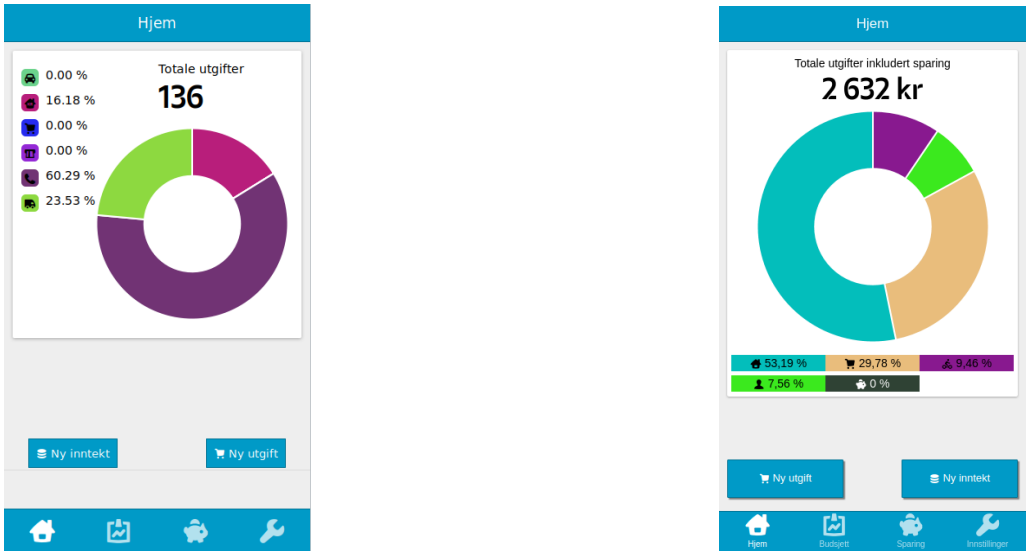
- **"No matter how cool your interface is, less of it would be better"** Vi forholder oss til dette prinsippet ved å ikke forurense grensenittet med mye knapper. Den konseptuelle tankegangen er at brukeren er mer opptatt av informasjon enn handlinger. Derfor skjuler vi så mye som mulig av funksjonaliteten vår.
- **"Follow users' mental models"** Vi forholder oss til dette prinsippet ved at vi stiller spørsmålet: "Hvorfor åpner brukeren applikasjonen?" Svaret gir en innsikt i hvilke designvalg vi bør gi applikasjonen. Vi fant ut at brukeren åpner applikasjonen for å se en oversikt, eller legge til en utgift eller en inntekt. Ved å tilby begge deler på hjemskjermen dekker vi begge behovene til brukeren.
- **"Enable users to direct, don't force them to discuss"** Alt brukeren gjør i applikasjonen kan endres eller trekkes tilbake. Det eneste vi avviker på er ved sletting av elementer. Vi sørger for at applikasjonen handler etter input fra brukeren og ikke etter verifiseringer brukeren aksepterer.
- **"Provide modeless feedback"** Brukeren får tilbakemelding i form av en notis som forsvinner raskt slik at brukeren forstår at en handlingen er utført.
- **"Contextualize information"** Vi forholder oss til dette prinsippet ved at vi presenterer brukeren med informasjon sett i forhold til hverandre. I hjemskjermen ser vi hvor mye utgifter av en budsjettpost forholder seg til de andre, og i skjermen for informasjon om en budsjettpost ser vi utviklingen av månedens utgifter i forhold til andre måneder.
- **"Provide direct manipulation and graphical input"** Vi tilbyr grafisk representasjon av våre data ved to tilfeller. I hjemskjermen vises et kakediagram som viser forholdet mellom utgiftene i budsjettposter. I skjermen for informasjon om en budsjettpost vises utviklingen til månedens utgifter i form av en strekgraf.
- **"Reflect object and application status"** Vi sørger for at applikasjonen er så responsiv som mulig ved at vi sender oppgaver som tar lang tid til bakgrunnen. Ved innlogging eller registrering av bruker er vi avhengig av respons fra serveren før vi bestemmer hva som skal skje videre. Da har vi benyttet oss av dette prinsippet ved at brukeren vil se et spinnende hjul som indikerer at applikasjonen venter.
- **"Avoid unnecessary reporting"** Vi følger dette prinsippet ved å fortelle brukeren at som var ønskelig å gjøre, gikk. Vi forteller ikke brukeren hvordan det gikk bra.
- **"Don't use dialogs to report normalicy"** Dette prinsippet er en naturlig konsekvens av at vi følger ovenfor nevnte prinsipper.
- **"Avoid blank slates"** Ved førstegangs oppstart av applikasjonen blir brukeren bedt om å fortelle om sin husstand. Vi foreslår da et budsjettoppsett for brukeren som kan endres der

og da eller fjernes helt. På den måten blir ikke brukeren møtt med en helt tom applikasjon. Brukeren føler da at han allerede er kommet godt igang med applikasjonen og vil ikke bli avskrekket av at han må gjøre mye jobb før applikasjonen blir tilrettelagt for han.

- **"Ask for forgiveness, not permission"** Som nevnt over er det ingenting i applikasjonen som brukeren legger til som ikke kan endres eller fjernes. Applikasjonen gjør det brukeren ber den om å gjøre. Hvis brukeren har gjort en feil, som for eksempel å legge til en utgift i en inntektspost, kan brukeren flytte utgiften over til rette kategori senere, eller fjerne den helt.
- **"Hide the ejector seat levers"** Vi følger dette prinsippet ved to tilfeller. Disse tilfellene er ved sletting av elementer og ved utlogging. I tilfellet for sletting, må brukeren aktivt åpne et kort for å vise frem sletteknappen. I det andre tilfellet er utloggingsknappen i den ekspanderende delen av informasjonslinja om brukeren.
- **"Optimize for responsiveness; accomodate latency"** Mye av funksjonalitetene brukeren blir presentert med kan kreve mye IO og eller nettverksforespørsler. Vi lar disse forespørselene skje i bakgrunnen slik at brukeren opplever at endringer han gjør på applikasjonen skjer øyeblikkelig.

Tilbakemeldingene ble brukt til å tilfredsstille oppdragsgiverens ønsker, og til å forbedre applikasjonens kvalitet. Ettersom applikasjonen ble utviklet for både Android og iOS, ble enkelte egenskaper tilpasset brukerens platform. Mye av dette styres av Ionic, og tab-menyen er et eksempel på dette. På iOS vil denne befinne seg i bunnen av skjermbildet, mens den ligger i toppen på Androidenheter. Disse plasseringene følger Apple [32] og Android [33] sine retningslinjer for tab-menyer.

Hjemskjerm



(a) Eldre utgave.

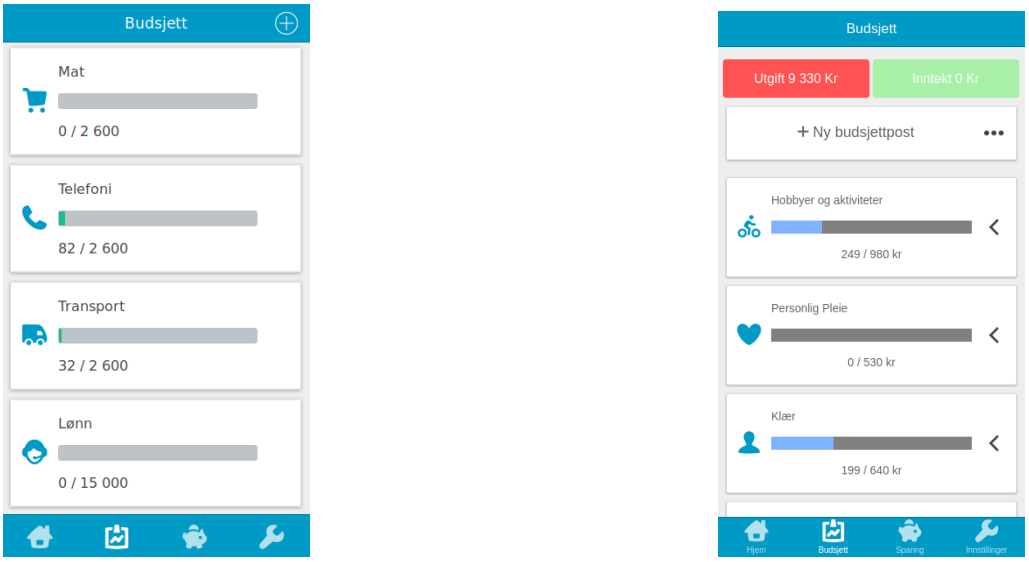
(b) Nyere utgave.

Figur 29: Sammenligning av hjemskjermer.

En eldre utgave av applikasjonen inneholdt små knapper med liten skrift. Diagramforklaringen var også plassert på en måte som forskjøv diagrammet mot høyre, i tillegg til at ikonene tok for stor plass til å enkelt kunne se tilhørende farger.

I den nyere utgaven er diagramforklaringen flyttet ned, slik at diagrammet blir midtstilt. Det forklares også øverst at sparing inngår i diagrammet. Knappene for ny utgift og ny inntekt er større, har skygge og har fått en mer naturlig plassering. I tillegg til disse endringene har tab-menyens knapper fått beskrivende tekst for enklere navigering.

Budsjett



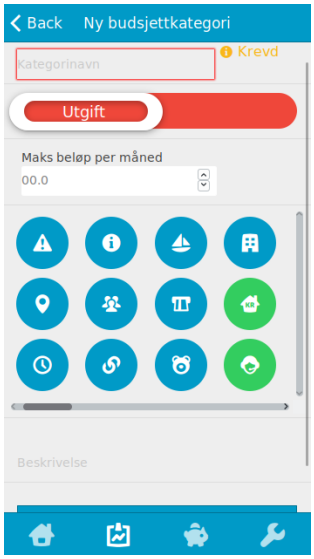
(a) Eldre utgave.

(b) Nyere utgave.

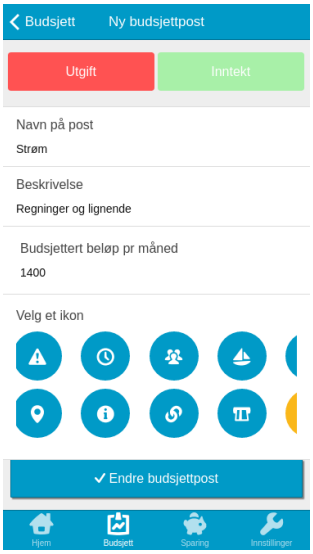
Figur 30: Sammenligning av budsjett.

Listen over budsjettposter i den eldre utgaven blandet både inntekts- og utgiftsposter sammen. Dette gjorde listen uoversiktlig. Knappen for å legge til en ny budsjettpost var også for frakoblet fra selve listen, ettersom den var plassert som et lite plussikon helt øverst til høyre på skjermen. Kortene som representerte budsjettpostene kunne dras til venstre for å endre eller slette postene.

Inntektsposter ble i nyere utgaver separert ut i en egen liste, og brukeren kunne selv bytte mellom listene som også viste totalt budsjettert beløp for inntekt og utgift. Kortene fikk en knapp som utløste en animasjon som dro knappen mot venstre. I tillegg ble knappen for ny budsjettpost endret. I eldre utgaver var den plassert som et plussikon øverst i høyre hjørne, men denne ble flyttet inn øverst i listen over budsjettposter. Grunnen til dette var at den gamle knappen virket distansert fra resten av brukergrensesnittet.



(a) Eldre utgave.



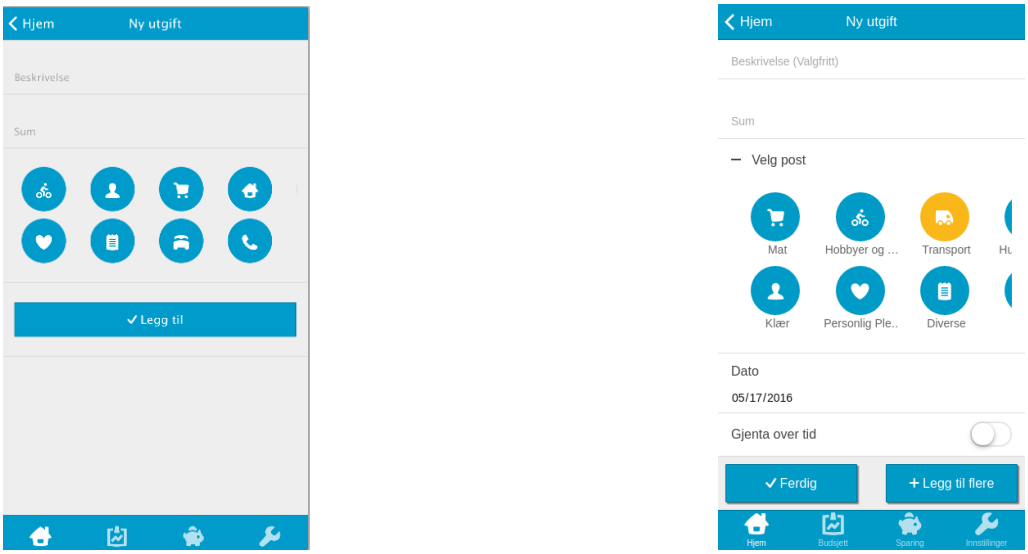
(b) Nyere utgave.

Figur 31: Sammenligning av skjerm for ny budsjettpost.

I det eldre brukergrensesnittet for oppretting av nye budsjettposter opptok ikonene stor plass. Det ble brukt en toggle-knapp for å bytte mellom utgifts- og inntektspost, og knappen for å fullføre var dyttet så langt ned at den ikke fikk plass på små skjermer. Dette resulterte i at brukeren måtte bla nedover for å finne den.

I nyere utgaver ble toggle-knappen byttet ut med to knapper som sto i stil med de to knappene over budsjettpostlisten. Tekstfeltetene ble samlet, som bidro til et mer ryddig grensesnitt. Vinduet med ikoner ble redusert til å vise kun to rader, som reduserte høyden på skjermbildet. Samtidig ble ikonvinduet endret, slik at det nå viser at listen med ikoner fortsetter mot høyre.

Transaksjoner



(a) Eldre utgave.

(b) Nyere utgave.

Figur 32: Sammenligning av skjerm for ny utgift.

Som sett på figuren over, ble det gjort store forandringer i skjermene for nye utgifter og inntekter. Bakgrunnen for inndata fikk hvit farge som andre steder i applikasjonen. Beskrivelse av hva som skal fylles inn ble også gjort klarere. Ikoner fikk tekst for å klargjøre at det velges en budsjettpost og ikke et ikon for transaksjonen. Det finnes også felter for å la den gjentas over tid, men disse er skjult med mindre toggle-knappen nederst på skjermen aktiveres. I tillegg las det til en ekstra knapp som lagrer transaksjonen og fjerner all inndata fra brukergrensesnittet. Brukeren kan dermed fortsette å lagre flere transaksjoner.

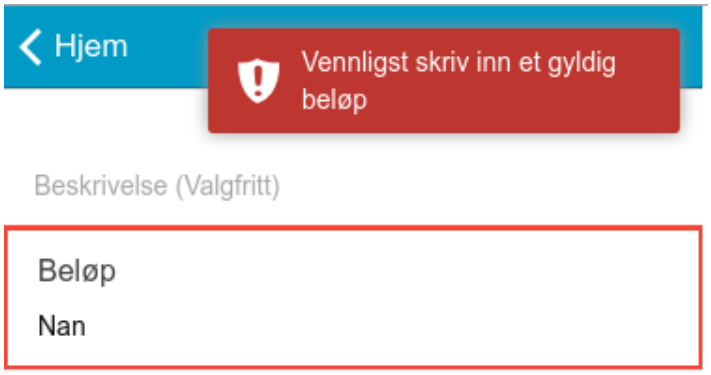
8.1.2 Innlesing av data

Applikasjonen har en tjeneste ved navn ”inputChecker” som inneholder funksjonalitet for å verifisere om data brukeren skriver inn er gyldig og kan brukes videre. Dette blir gjort alle steder brukeren har mulighet til å skrive inn data. Hvis feil eller ikke godkjent data blir skrevet inn vil brukeren motta en beskjed med en beskrivelse av problemet, samt en markering av hvor denne feilen ligger. Dette er et eksempel på endring som er gjort for tilpasning til krav for universell utforming [34].

Felgende data blir sjekket:

- Tekster: Må være definert og ikke en tom streng. Kan heller ikke være lengre enn 75 tegn. Når det gjelder navn så er maksimumslengden satt til å være 25 tegn.
- Tall: Må være definert, kan ikke være null eller 0 og må være et tall som er større enn 0, men mindre 9 999 999.
- Dato: Håndteres selv ved å bruke ’date’ som type for innlesning i HTML. Brukeren vil ikke få muligheten til å skrive inn noe, da mobilens nettleser bruker egne metoder (datepicker).

Grensene er satt på grunnlag av applikasjonens brukergrensesnitt ikke skal fylles med tekst.



Figur 33: Validering av innlest data

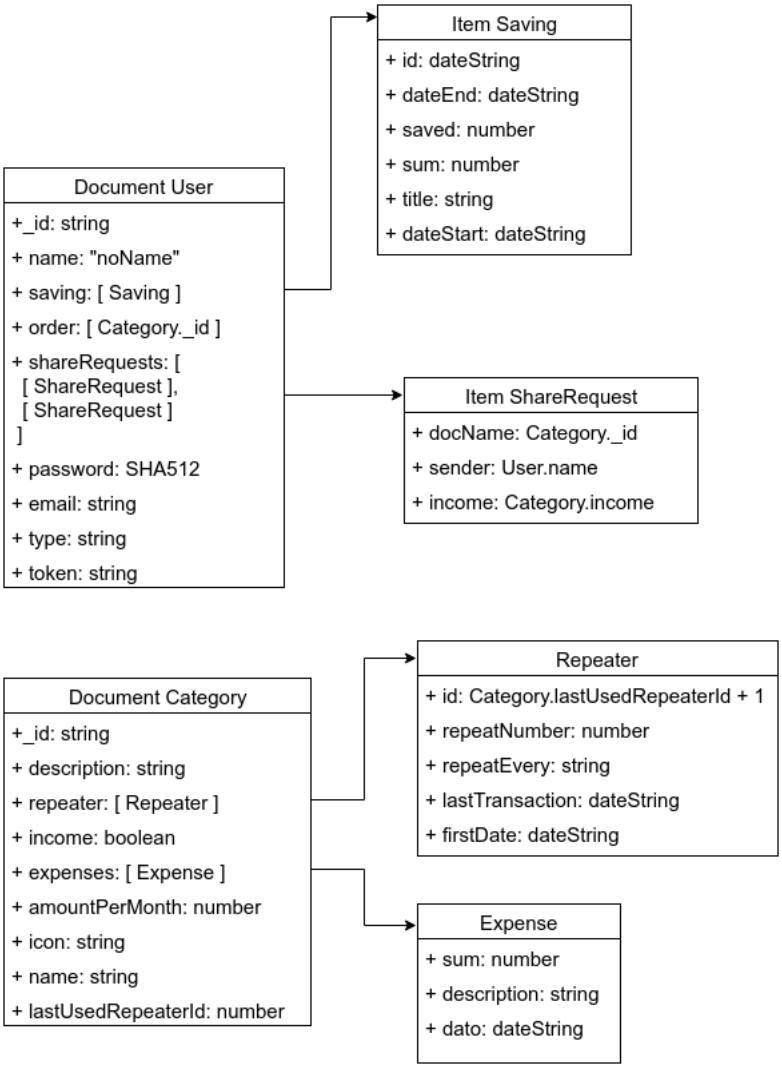
8.2 Database

Siden vi benytter oss av et databasedesign med få relasjoner har vi benyttet oss av en relasjonsløs databasetjeneste. Den lagrer dokumenter istedet for tabeller, rader og kolonner. Vi skiller mellom to forskjellige dokumenter.

- **User** dokumentet inneholder data som ikke skal deles av andre.
- **Category** dokumentene inneholder data for en budsjettpost som kan deles av andre.

Dataene fra dokumentene kommer tilbake fra databasen i form av et JSON-objekt. Et eksempel på dette vises i vedlegg H

På den måten skiller vi det som vi anser som sensitiv data, og informasjon som bruker kan velge å dele med andre. I tillegg deles ikke annen informasjon enn den som er strengt nødvendig. Tjenesten userModel samler dataene fra disse dokumentene og presenterer disse til brukeren.



Figur 34: Databasedesign

Dokument User	
_id	Dokumentets id, satt til “localUser“ som standard
name	Navn på brukeren satt til “noName“ som standard
savings	Et sett med sparemål satt til et tomt sett som standard
order	Et sett med Category._id for å holde orden på rekkefølge
shareRequests	Et sett med sett av delte kategorier
password	Brukerens passord hashet med SHA512
email	Brukerens epost
type	Hva brukeren benytter for autentisering Brukeren kan velge mellom Google, Facebook, og egendefinert autentiseringsmekanisme
token	Benyttes for å autentisere brukeren med backend tjeneren.

Tabell 12: Brukerdokumentet

Element Sparemål	
id	Unik id til et sparemål, satt til datoteksten til når det ble opprettet
dateEnd	Datoteksten til når sparemålet avlsuttes
saved	Hvor mye kroner som er oppspart i tilfelle brukeren endre sparemål
sum	Hvor mye kroner som skal spares
title	Navnet på sparemålet
dateStart	Datoteksten til når man starter sparemålet Hvis brukeren endrer sparemålet, endres denne til ny aktuell dato

Tabell 13: Sparemålelementet

Element Deleforespørsel	
docName	Id til den kategorien som skal deles
sender	Navnet til brukeren som skal dele kategorien
income	Flagget som sier om det er en inntekt- eller utgiftskategori

Tabell 14: Deleforespørselelement

Dokument Kategori	
_id	Unik id til kategorien. Navnet er en kombinasjon av "Category" og navnet til kategorien
description	Utvidet beskrivelse av kategorien
income	Flagg som sier om det er en inntekt- eller utgiftskategori
repeaters	Sett med gjenntagende utgiftsinformasjon
expenses	Et sett av transaksjoner for kategorien
amountPerMonth	Hvor mye kroner en bruker har budsjettert per måned for en kategori
icon	En tekst som forteller hvilket ikon som skal assosieres med kategorien
name	Navnet på kategorien
lastUsedRepeaterId	Antall Repeater objekter som finnes i expense-settet

Tabell 15: Kategoridokumentet

Element Utgift (Vi kaller alle transaksjoner utgifter, selv om kategorien er en inntekt)	
sum	Hvor mye kroner som ble brukt på en utgift
description	Beskrivelse av en utgift
date	Datoen for når utgiften fant sted

Tabell 16: Utgiftselementet

Element Gjenntagende Utgift	
id	Unik id til elementet
repeatNumber	Angir hvor stort intervall innenfor verdien av repeatEvery
repeatEvery	Angir hva slags verdi av dager, uker, måneder eller år et intervall skal ha
lastTransaction	Datotekst som forteller når denne utgiften sist ble lagt til
firstDate	Datotekst som forteller når denne utgiften først ble lagt til

Tabell 17: Gjenntagende Utgiftselement

9 Kvalitetssikring

Under utviklingsperioden hadde vi et stort fokus på kvalitetssikring. Ettersom oppdragsgiver ønsker å publisere applikasjonen, var det viktig å opprettholde gode prosesser for kvalitetssikring. Dette innebar å skrive tilstrekkelig med enhetstester, foreta brukertester og hyppig koderevisjon. I dette kapittelet beskrives disse prosessene og verktøyene som ble brukt.

9.1 Statisk kodeanalyse

Statisk kodeanalyse ble utført ved hjelp av ESLint. Vi tilpasset et sett med regler som skulle håndheves, og verktøyet brukte disse i sine analyser. Hver gang kode skulle opplastes til Git, ble kodeanalyser kjørt. Dette sikret at kode som ikke fulgte de fastsatte standardene ikke kom inn i prosjektets repository.
Reglene som ble håndhevet av ESLint ble definert i en konfigurasjonsfil hvor regler kunne tilpasses vårt prosjekt og stil. Mange av reglene var spesifikke for AngularJS og var definert utifra en stilguide for AngularJS-prosjekter [31]. Ved å installere en utvidelse til ESLint fikk vi tilgang til disse reglene og kunne selv velge hvilke vi ville bruke. De fleste av reglene var ment til å sørge for at applikasjonen ble utviklet med gode kodestandarder og at kodestilen var konsistent i hele prosjektet. Dette inkluderte blant annet navngiving av kontrollere, grenser for AngularJS-komponenter i hver fil og bruk av komponenter spesifikke for AngularJS i stendet for standard JavaScript.

9.2 Testskriving

Det ble skrevet både enhetstester og end-to-end-tester for store deler av applikasjonen. Disse testene var i stor grad skrevet som regresjonstester. Grunnen til dette var både for å forhindre at endringer i kodebasen resulterte i uventede feil, samtidig som testene i seg selv beskriver funksjonaliteten som testes. Dette er en form for dokumentasjon som kan gi oppklaring i kode for eventuelle videreutviklere. Alle hadde ansvar for å skrive unittester for sine egne moduler, men på enkelte tidspunkt ble det tatt initiativ for å utbedre eksisterende tester i kritiske deler av applikasjonen, uavhengig av testenes forfatter. I tillegg ble det skrevet nye tester der det var nødvendig.

9.2.1 Frontend

Enhetstester ble skrevet ved bruk av testrammeverket Jasmine [35]. Dette rammeverket er valgt på grunnlag av flere faktorer. Syntaksen til Jasmine tilrettelegger for at testene skal beskrive funksjonaliteten som testes på en lettleselig måte. Feiler en enhetstest vil det derfor være lettere for utviklere å forstå testens hensikt.

Det var også naturlig å velge Jasmine ettersom veiledningen på nettsiden til AngularJS omtaler Jasmine som det mest populære testrammeverket for AngularJS-applikasjoner [36]. Dette betydde i praksis at mange spørsmål angående rammeverket allerede fantes på diverse nettsteder og forum, noe som kan ha spart oss for mye tid.

Kommunikasjon med databasetjenestene i tillegg til selve dbPutter og userModel er eksempler på kritiske elementer ved applikasjonen som det ble skrevet enhetstester for. Testene for dbPutter og userModel gikk blant annet ut på å sikre at informasjon som ble sendt til funksjonene i disse tjenestene ble videreført til PouchDB. Kommunikasjonen med dbPutter og userModel ble testet ved å sjekke at modulene som tok i bruk tjenestene kommer frem til punktet hvor de kaller på dem, i tillegg til at riktige funksjoner blir kalt. I slike tilfeller ble det laget mock-objekter av tjenestene slik at koden kan bli testet i isolasjon. Eventuelle feil i tjenestene ville dermed kun slå ut i sine egne tester, uavhengig om tjenestene er brukt av andre moduler.

Figur 35: Utsnitt av dekningsrapport

Figuren ovenfor viser hvor stor prosentandel av koden på frontend som er dekket av enhetstester. Antallet enhetstester på frontend var 144 ved prosjektslutt. Hver test er målrettet i den forstand at den sjekker en mindre del av den totale funksjonaliteten. Vi nådde ikke vårt mål om 80% dekning, på grunn av prioritering av ny funksjonalitet mot slutten av prosjektperioden. Vi mener allikevel at vår testdekning burde gi et godt utgangspunkt for eventuell videreutvikling.

Ettersom enhetstestene på frontend ble testet i isolasjon, måtte det opprettes mock-objekter for enkelte tjenester.

I kodeutsnitt 9.1 vises et eksempel på slike mock-objekter. Disse objektene ble brukt i en test som vises senere. Testens hovedformål var å sjekke at applikasjonen gikk bakover i historien etter en transaksjon ble lagt til. Dette ville som oftest bety at brukeren ble tatt tilbake til hjemskjermen. Måten dette ble gjort på i koden var å bruke en tjeneste kalt $ionicHistory, som lot oss manipulere applikasjonens history-stack. Denne tjenesten måtte lages en mock for.

```
1 historyMock = {
2       goBack: jasmine.createSpy()
3     };
```

Kodeutsnitt 9.1: Mock-objekter i Transaction-test

$ionicHistory inneholder en funksjon med navn goBack som brukes i Transaction-modulen vår. Dette er funksjonen som tar brukeren tilbake til hjemskjermen, og vi ønsket å sjekke at denne ble kalt.

Ettersom denne funksjonen lå i en ekstern tjeneste, måtte den mockes. Objektet med navn historyMock er inneholder en variabel kalt goBack. På denne variabelen blir det satt en 'spy', som er en mekanisme Jasmine tilbyr for å overvåke funksjoner. I utganspunktet gjør ikke goBack noe som helst, men et annet sted i testfilen bestemmes det at koden som testes skal kjøre historyMock.goBack i stedet for $ionicHistory.goBack.

Videre følger en enhetstest hvor historyMock ble tatt i bruk. Testen var multifunksjonell i den forstand at den sjekket at to forskjellige funksjoner blir kalt.

```
1 it("calls goBack after expense is added", function() {
2     // Create transaction test objekt
3     controller.transaction = {
4       description: "Test",
5       sum: 438,
```

```
6         date: new Date()
7       };
8       controller.budgetPost.icon = "test";
9       // Save the transaction in the controller, and go back in history
10      controller.saveAndQuit();
11
12      // Check that the transaction was saved with correct parameters
13      expect(userModelMock.addExpense).toHaveBeenCalledWith(controller.budgetPost,
            controller.transaction, -1);
14
15      //Check that goBack was called
16      expect(historyMock.goBack).toHaveBeenCalled();
17  });
```

Kodeutsnitt 9.2: Enhetstest fra transaction-modul

Funksjonen som inneholder funksjonaliteten som testes var kalt ”saveAndQuit”, og var tilgjengelig i transaction controller. Denne funksjonen ble kalt på linje 8. Videre ble det sjekket på linje 9 at transaksjonen ble lagt til i usermodel. Ettersom userModel var en separat tjeneste, ble også den mocket i testen. På testens siste linje blir det sjekket at historyMock.goBack ble kalt. Dersom denne mock-funksjonen blir kalt, betyr det i praksis at $ionicHistory.goBack vil bli kalt i et produksjonsscenario.

Karma ble valgt som testkjører. Dette er et kommandolinjeverktøy som setter opp en lokal webserver og kjører budsjettapplikasjonen i en eller flere valgfrie nettlesere. Alle resultatene blir presentert i kommandolinjen. Den samme veiledningen til AngularJS som omtalte Jasmine omtaler også Karma [36].

E2E-testing, eller end-to-endtesting, ble brukt for å sikre at navigering gjennom applikasjonens moduler var mulig. Slik testing simulerer brukerens interaksjon med applikasjonen ved å kjøre applikasjonen og sjekke at mekanismer i brukergrensesnittet fungerer som forventet. Dette gjøres ved å eksempelvis be testkjøringsverktøyet om å trykke på en knapp, og sammenligne resultatet med testens forventede resultat.

Denne formen for regresjonstesting mente gruppen var nødvendig, ettersom feil i brukergrensesnittet ofte er vanskelige å oppdage under utviklingsfasen. For eksempel blir tilbakeknappene i applikasjonen fremstilt av Ionic, og disse kan forsvinne som følge av endringer i koden. Disse knappene ble det skrevet E2E-tester for, siden det vanskelig å oppdage at de mangler.

For E2E-testing brukes rammeverket Protractor. Det bruker Jasmine, så syntaksen er svært lik enhetstestene i prosjektet. Rammeverket er bygget spesifikt for testing av AngularJS-applikasjoner, og er utviklet av samme utviklere som AngularJS. Testene kjøres en eller flere valgfrie nettlesere, på samme måte som Karma.

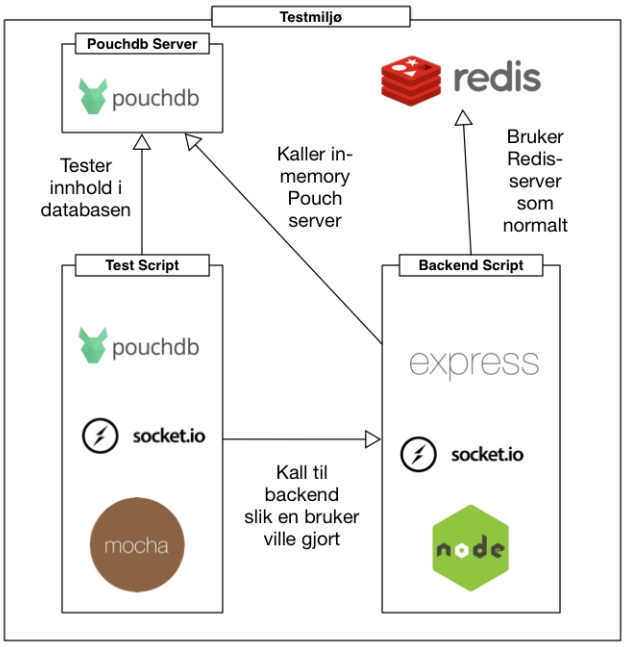
Etter første milepæl ble det bestemt at E2E-testing skulle nedprioriteres. Dette ble bestemt fordi skriving av slike tester er tidkrevende. Et eksempel på dette var at enkelte knapper skulle fremvises etter at brukeren dro et element til siden. Det ble brukt for lang tid på å prøve å få Protractor til å utføre denne oppgaven i forhold til testens nytteverdi. Etter flere lignende situasjoner oppsto så vi oss nødt til å prioritere annet arbeid.

9.2.2 Backend

Unit tester for backend ble gjort på en litt annen måte enn frontend. Her dekker ikke testene små interne deler i koden isolert sett. I steden for er testene laget som klienter som kommuniserer med serveren på en måte de vil gjøre i et produksjonsscenario. En enkel unit test for en funksjon er ofte at man kaller funksjonen med et input og sjekker output opp mot det man forventer funksjonen skal gjøre. Dette blir brukt i enkelte av testene der man sender informasjon til serveren og deretter tester svaret man får. Serveren gjør imidlertid også mye som ikke kan sjekkes om fungerer ved kun å se på responsen til klienten. En viktig jobb til serveren er å håndtere all databasekommunikasjon. For å kunne kjøre serveren og gjennomføre nyttige tester bør man ha en database i miljøet man kjører testene. Derfor blir en PouchDB server som bare lagrer ting i minnet kjørt i testmiljøet. Denne fungerer på samme måte som en CouchDB-server og implementerer samme API. At den kun lagrer ting i minnet betyr at testmiljøet ikke blir fylt med data fra gamle tester. På denne måten kan det kjøres nye tester, eller tester som er endret uten at gamle data skal påvirke resultatet.

Dermed kan det implementeres tester som gjør endringer i databasen, og serveren vil bruke databasen i testmiljøet slik den ville i produksjon. På grunn av HTTP API-et i PouchDB/CouchDB kan derfor testene bare gjøre et HTTP-kall til databasen i testmiljøet og få data derifra. Disse kan da sammenlignes med det som forventes at serveren skal gjøre. Testene oppretter også PouchDB-databaser for brukerne man benytter som klienter i testene.

Disse ligger også kun i minnet og i motsetning til den backenden benytter er det ikke en server men er kun i scope-et til selve test filen. Disse representerer klientene sine lokale databaser og blir brukt til å teste replikering.



Figur 36: Testmiljø for backend

I kodeutsnitt 9.3 ser vi eksempel på en enkelt test som tester registrering av en bruker. Figur 36 viser hvordan denne blir utført. Det blir sendt et kall til backend-scriptet som utfører en databaseoperasjon. Testen kaller deretter databasen direkte og sjekker at verdien der er det som forventes. I dette tilfellet er det et dokument som representerer en bruker i _users databasen.

```
1
2  it("registering a user should put the user in the database", function (done) {
3      request.post({    //send registreringskall til server
4        url: 'http://localhost:6969/register',
5        json: true,
6        body: {                //brukerdata
7          method: 'db',
8          email: 'test@user2.no',
9          password: '1234'
10       }
11     },
12     function(err, res, body){  //svar registreringskall
13       request.get({          //send kall direkte til databasen
14         url: 'http://admin:devonly@localhost:5984/_users/org.couchdb.user:
               test$user2-no'
15       },
16       function(err, res, body){     //svar databasekall
17         var parsed = JSON.parse(body);
18         if (parsed.error) {
19           should.fail();   //fant ikke brukeren
20           done();         //async test ferdig
21         }
22         else {      //sjekk om brukeren er blitt satt i databasen
23           parsed.name.should.equal('test$user2-no');
24           done();    //async test ferdig
25         }
26       });
27     });
28   });
```

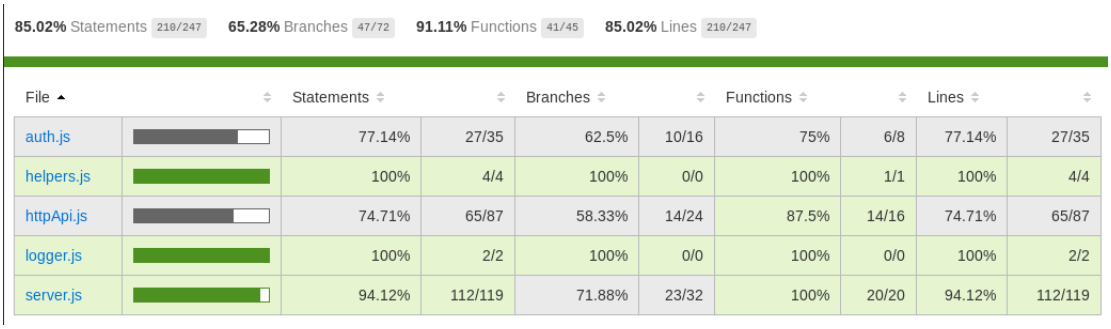
Kodeutsnitt 9.3: Registreringstest

Siden vi implementerte registrering og innlogging for både Facebook og Google+ ønsket vi også å teste dette. Facebook har et system hvor det kan opprettes testbrukere med brukernavn og passord som kan brukes for å logge seg inn. Testbrukeren kan derfor logges inn på Facebook i testen og få en 'access token' og bruke denne for å teste serveren. Resultatet som ønskes er da at serveren klarer å hente brukerinformasjon utifra den 'access token'-en.

Istedenfor Jasmine og Karma som brukes for testing i koden for frontend, bruker testene for serveren Mocha og Chai. Grunnen til dette var at vi opplevde det som problematisk å teste asynkron kode med disse verktøyene. Asynkrone tester var veldig mye lettere å skrive og få til å fungere med Mocha og Chai. Verktøyene brukes veldig likt og det var ikke noe stort problem å ta dem i bruk. I likhet med frontend-testingen brukte vi Istanbul for å generere testdekningsrapporter.

9.3 Koderevisjon

Gjennom hele prosjektperioden ble det opprettholdt et system hvor alle utførte oppgaver ble revidert av et annet gruppemedlem. Dette ble utført ved at oppgavens eier flyttet tilhørende oppgave over i QA-kolonnen på Kanban-tavla i JIRA, og gav beskjed til gruppen. En annen i prosjektgruppen byttet så over over til den branch-en og så på endringer som var gjort i koden. Dersom det fantes kode med forbedringspotensiale ville det QA-ansvarlige medlemmet gi tilbakemelding ved bruk av TODO-kommentarer i selve koden. Slike kommentarer var lett synlige, og IDE-ene vi brukte lot oppgavens eier se alle TODOs i en liste.



85.02% Statements 210/247 **65.28%** Branches 47/72 **91.11%** Functions 41/45 **85.02%** Lines 210/247

File	Statements		Branches		Functions		Lines	
auth.js	77.14%	27/35	62.5%	10/16	75%	6/8	77.14%	27/35
helpers.js	100%	4/4	100%	0/0	100%	1/1	100%	4/4
httpApi.js	74.71%	65/87	58.33%	14/24	87.5%	14/16	74.71%	65/87
logger.js	100%	2/2	100%	0/0	100%	0/0	100%	2/2
server.js	94.12%	112/119	71.88%	23/32	100%	20/20	94.12%	112/119

Figur 37: Testdekning for backend. Enkeltfiler er under 80% men helheten er på 85% som vist øverst.

Dette ble da pushet på samme branch, og oppgavens eier fikk i oppgave å utbedre alle tilbakemeldinger. QA-ansvarlig gav kun tilbakemeldinger; koden skulle alltid forbedres av oppgavens eier, og aldri av QA-ansvarlig. Dette var et prinsipp som ble fulgt opp gjennom hele utviklingsperioden. Etter det ble gjort leste QA-ansvarlig over de nye endringene, og migrerte branchen inn i master. Selve kvalitetssikringen bestod både av å manuelt se gjennom koden, i tillegg til å kjøre Gulp-tasks for end-to-end-testing, enhetstesting og statisk kodesjekk.

Koderevisjon ble også gjort i sammenheng med valgemnet Professional Programming. Her ble imidlertid koden revidert av studenter fra andre bachelorprosjekter. Dette gav oss nøytral og objektiv input fra utsiden som hjalp oss å forbedre kodekvaliteten. Samtidig reviderte vi andre gruppers kode, noe som viste seg å være en positiv erfaring, ettersom flere andre grupper også bruker JavaScript og AngularJS.

9.4 Brukertesting

Ettersom applikasjonens hovedmålgruppe er ung, har det hovedsaklig blitt brukt studenter som testpersoner. Dette inkluderer studenter fra andre bachelorprosjekter, i tillegg til andre bekjente. Både programvareutviklere og personer fra urelaterte fagfelt ble spurt om å være testpersoner. Dette ble gjort for å få større bredde på testene, og mer variasjon i resultatene.

For inspirasjon og veiledning til brukertesting leste gruppen artikler fra NN/g (Nielsen Norman Group), som har et stort antall artikler og rapporter innen UX (user experience) som fagfelt. Et eksempel på dette er en artikkel som inneholder tips og regler for god gjennomføring av brukertesting [37]. Her beskrives det blant annet hvordan testene bør utføres uforstyrret og at testpersonene skal forstå applikasjonen på egenhånd, uten mye informasjon på forhånd. Disse reglene er eksempler på metoder hentet fra NN/g.

9.4.1 Oppgaver

Brukertestingen ble planlagt slik at brukerene fikk mulighet til å prøve ut hovedfunksjonaliteten i applikasjonen. Dette innebærer ikke at brukeren måtte prøve ut alle forskjellige moduler. Meningen var her å kartlegge brukerens forståelse av applikasjonens brukergrensesnitt ved førstegangsbruk i henhold til gitte oppgaver. Følgende introduksjon ble gitt til brukerene før oppgaver ble tildelt:

Dette er en budsjettapplikasjon hvor det er mulig å legge inn budsjettposter eller bruke et allerede ferdig oppsatt budsjett. Poenget med applikasjonen er at brukeren kan legge inn sine utgifter og inntekter inn i disse budsjettpostene for å holde kontroll på sin økonomi. Det er også mulig å legge inn sparemål"

"Ved førstegangsoppstart av applikasjonen møter brukeren en introduksjon som gir mulighet til å velge et forhåndsbestemt budsjett, og eventuelt tilpasse det.“

Det ble ikke gitt mer konkret informasjon enn dette til brukerene, men det ble verifisert at testbrukerene hadde forstått informasjonen før oppgavene ble påbegynt. Oppgavene som videre er listet opp ble også lest opp som de står. Det ble ikke gitt hjelp til brukerene, og de måtte på finne ut hvordan oppgavene skulle løses helt på egenhånd. Ved enkelte tilfeller hvor brukerene sto fast over lengre tid, ble det gitt hint eller en direkte forklaring på hva som skulle gjøres for å komme videre.

Oppgave 1
Start opp applikasjonen og gå gjennom introduksjonen. Legg så til en ny budsjettpost: lønn (15 000/mnd) og en ny budsjettpost: skoleutgifter (600/mnd).

Ved å gå igjennom introduksjonen fikk brukeren et ferdigoppsatt budsjett(SIFO referansebudsjett). Det lå da ferdigdefinerte budsjettposter når applikasjonen startet etter introduksjonen. Ved å legge til en ny kategori for utgift og inntenkt ble det undersøkt om brukeren forstod forskjellen på utgift og inntekt, samt om 'legg til'-knapper var forståelige. 'Legg til' er definert på samme måte over hele applikasjonen.

Oppgave 2
Legg til inntekt i den nye budsjettposten 'lønn' som kommer inn den 8. hver måned av en sum på 15 000/mnd.
Kategori for skoleutgifter er for liten. Øk denne posten med 50,- Legg så inn en utgift fra da du var på Libris i går og handlet blyanter for 100 kr.

Første del gikk ut på å undersøke om brukeren forstod hvordan det legges til en inntekt eller en utgift til en gitt budsjettpost. Her ble det i tillegg kontrollert om brukeren klarte å legge inn en løpende inntekt som kom inn hver måned. Andre del gikk først ut på å redigere en budsjettpost, hvor brukeren måtte finne frem til stedet hvor en budsjettpost kan endres. I tillegg skulle brukeren legge inn en utgift til den budsjettposten som manuelt ble lagt inn i første oppgave. Her var det flere måter å gjøre det på.

Oppgave 3
Legg til et sparemål for sydentur den 15. juli 2016. Det skal spares 14 000,- Slett så sparemålet

Her ble det undersøkt om brukeren fant frem til oversikten over sparemål og forstod hvordan konkrete sparemål kunne legges til. Fremgangsmetoden og knappene var tilsvarende til oppgave 1, men det lå på et annet sted.

Oppgave 4

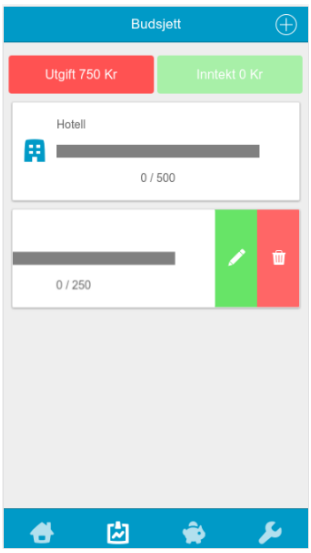
Finn igjen utgiften fra Libris du tidligere la inn og øk summen med 50kr.

Dette var den siste oppgaven hvor brukeren skulle dypere inn i applikasjonen. Brukeren skulle først inn på budsjettposten som ble lagt inn i oppgave 1, for videre å endre en enkelt utgift. Her ble det sjekket om brukeren fant frem til riktig sted og om brukeren forstod navigasjonsmekanismene.

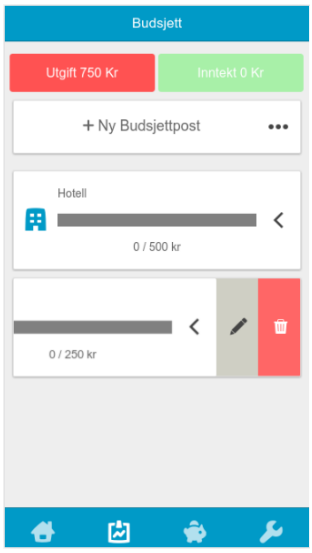
9.4.2 Brukertest - Første runde

Første runde med brukertesting ble avsluttet etter å ha utført alle oppgavene på kun én testperson. Problemet som dukket opp gjaldt navigering for å kunne endre en budsjettpost. Dette gjøres på samme metode over hele applikasjonen og var svært kritisk. Dette problemet kommer spesielt frem i Oppgave 3 som videre ble utført som en isolert test med kun denne funksjonaliteten på ytterligere to personer til for å verifisere problematikken. Og det viste seg at disse slet med nøyaktig det samme. Forbedringer av brukergrensesnittet ble så umiddelbart iverksatt.

Andre elementer som også måtte utbedres var språkbruken generelt over hele applikasjonen. Det ble også funnet et par mindre feil som måtte rettes.



(a) Før første brukertest.



(b) Etter første brukertest.

Figur 38: Budsjettposter før og etter første brukertest

Som vist på venstre figur er det listet opp to forskjellige to forskjellige budsjettposter. Den ene har fått navnet 'Hotell' og dem andre er dratt til siden for å få frem knappene 'endre' og 'slette'. Hver budsjettpost er presentert på et hvit kort.

For å kunne løse oppgave 3 i brukertesten, må brukerene dra kortet til venstre for å få opp disse to knappene som vises på begge figurene.

Dette var den eneste måte å få frem knappene på. Etter utbedringer (høyre figur) var fortsatt muligheten for å dra kortet til siden tilgjengelig, men det ble plassert en knapp ute på høyre side som forteller at kortet var interaktivt. Knappen er illustrert med en pil i den retningen hvor kortet

skal dras. Knappen utløser en animasjon som beveger kortet til siden, og dermed viser de skjulte knappene.

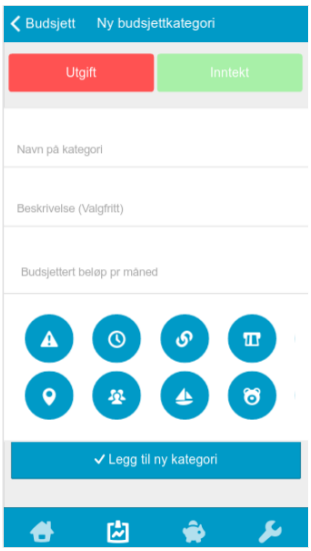
Andre utbedringer ble også gjort før nye brukertester ble utført. Dette innebærer blant annet at pluss-ikonet øverst i høyre hjørne (venstre bilde) ble fjernet og erstattet med en egen knapp over alle kortene. Det ble også implementert en mulighet for å endre sorteringsrekkefølge på kortene. Disse endringene ble foretatt over hele applikasjonen.

9.4.3 Brukertest - Andre runde

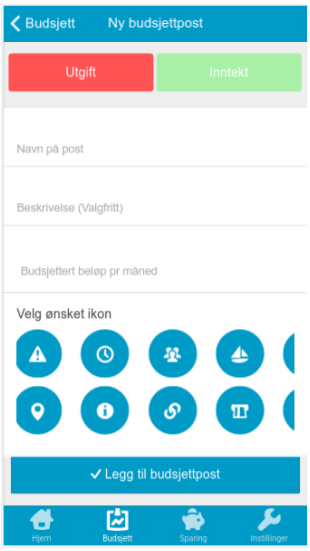
Etter andre runde med brukertesting fikk vi bekreftet at endringene som ble gjort etter første runde var riktige. Brukerne hadde ingen problemer med flytting av kort for å endre eller slette. Disse brukertestene tok også betydelig mindre tid enn ved første runde, noe som antydet at applikasjonen også virket mer logisk totalt sett.

Det som ble spesifikt påpekt under disse seansene var bruk av ikoner for budsjettpostene og hvordan dette var satt opp til å fungere i applikasjonen. Når brukerene fikk beskjed om å registrere en ny kategori ble det listet opp en rekke ikoner. Her så det kun ut til å være mulig å velge blant 8 stykker, selv om det var mulighet for å dra ikonene til siden for å se flere.
Det er også gitt tilbakemeldinger fra dem som utførte brukertestene at mye av ordbruken flere steder i applikasjonen var forvirrende og mangelfull. Dette gjaldt blant annet enkelte feilmeldinger som virket misvisende, i tillegg til teksten på knapper. Når brukerene gikk inn for å endre noe, virket det forvirrende at det sto annet enn 'Endre' på knappen i bunn. Enkelte steder manglet det også merkelapper, noe som gjorde brukeren usikker på hva som skulle gjøres.

Det var også en usikkerhet når det gjaldt forskjellen av ny inntekt/utgift og ny budsjettpost. Sidene var veldig like utseende, noe som forvirret brukeren. Det ble også påpekt at en form for bruksanvisning kunne vært nyttig.



(a) Før andre brukertest.



(b) Etter andre brukertest.

Figur 39: Ny budsjettpost

Som vist i figurene ovenfor er begge inne i skjermen for å legge til en ny budsjettkategori. Den til venstre representerer applikasjonen før utbedringer, den til høyre presenterer applikasjonen etter forbedringer. På høyre bilde er ikonene som er listet opp dratt litt til siden for å vise at det ligger flere hvis brukeren blar til siden. I tillegg er teksten 'Velg ønsket ikon' lagt til for å være sikker på at brukeren forstå hva som skal gjøres.

9.4.4 Brukertest - Tredje runde

Den tredje og siste runden med brukertesting ble foretatt med videreutvikling i tankene. Det var ikke nok tid til å endre på punktene som ble bemerket, men disse kan eventuelt utbedres etter prosjektperioden. Det ble her kun foretatt brukertesting på èn person.

Brukeren hadde problemer med å finne ut hvordan eksisterende utgifter og inntekter kunne endres eller slettes. Disse knappene kommer frem ved å dra kortet til venstre eller ved å trykke ute på høyre side. Til tross for at resultater fra andre runde indikerte at dette var forbedret, kom det frem at det ikke var like innlysende for alle brukere. Brukeren ble også forvirret under oppgaven om å legge inn en gjentagende inntekt med navn Lønn. Her var det problemer med å skille mellom budsjettpost og inntekt.

Etter denne runden ble det utviklet en introduksjonsmodul i applikasjonen som både forklarer begreper, og viser strukturen i applikasjonen. Her blir også de skjulte knappene fremvist for brukeren.

Forhåpentligvis vil denne introduksjonsmodulen gjøre applikasjonen enklere å bruke for førstegangsbrukere.

Det å implementere en bruksanvisning i introduksjonen vil det stride imot krav som er satt opp i kravspesifikasjonen under kapittel 2.4 PACT - Kunnskapsnivå. Det ble brukt tid på å implementere bruksanvisning for å inkludere så mange brukere som mulig. Dette vil også løse tilbakemeldinger fra brukertester som enda ikke er bitt løst. Herunder forskjell på inntekt/utgift kontra budsjettkategori og navigering.

10 Verktøy

10.1 Pakkehåndtering

10.1.1 Bower

Vi har benyttet Bower for håndering av biblioteker som applikasjonen er avhenging av. Bower gjør installasjon av biblioteker til applikasjonen enklere, i tillegg til å at det gir god oversikt over pakker som trengs for at applikasjonen skal fungere. Alle avhengingheter blir definert i en fil kalt 'bower.json'. Versjonsnumrene til bibliotekene blir da også ført inn i denne filen.

Vi har brukt en rekke eksterne bibliotek og rammeverk i vår applikasjon. Under følger en tabell over alle rammeverk og biblioteker, sammen med sine lisenser og versjonsnummer.

Bibliotek	**Lisens**	**Versjon**	**Beskrivelse**
Ionic	MIT	1.2.4	Brukergrensesnitt, tilpasning til OS
Chart.js	MIT	1.0.2	Grafer og diagram i brukergrensesnitt
jsSHA	BSD	2.0.2	JavaScript-implementasjon av Secure Hash Algorithm
MomentJS	MIT	2.12.0	Beregninger av tid og dato
AngularJS	MIT	1.4.3	Rammeverk for MVC i webapplikasjoner
ngCordova	MIT	0.1.24	Rammeverk for å støtte Angular-applikasjoner i Cordova
Platform.js	MIT	1.3.1	Bibliotek for å oppdage klientens plattform
PouchDB	Apache 2.0	5.2.0	Databaserammeverk for klientsiden
pouchdb-replication-stream	Apache 2.0	1.2.6	Bibliotek for streams til replikering mellom PouchDB og CouchDB
socket.io-client	MIT	1.4.5	Bibliotek for klientsiden av socket.io
angular-animate	MIT	1.4.3	Bibliotek for animasjoner i Angular-applikasjoner
angular-chart.js	Apache 2.0	0.1.0	Angular-bindinger for Chart.js
angular-moment	MIT	0.10.3	Angular-bindinger for MomentJS
angular-pouchdb	MIT	4.1.0	Angular-bindinger for PouchDB
angular-toastr	MIT	1.7.0	Bibliotek for fremvisning av toast-beskjeder
angular-translate	MIT	2.9.0	Bibliotek for internasjonalisering

Tabell 18: Pakkeoversikt

10.1.2 Npm

Npm er et pakkehåndteringssystem for javascript. Her finnes alle typer verktøy og moduler som er nyttige for utvikling av alle typer JavaScript-applikasjoner. Pakkene som behøves defineres

her i en fil kalt 'package.json'. Her blir versjonsnummer spesifisert og hvorvidt det er en utviklingsavhengighet eller en runtime-avhengighet. Npm bruker da denne filen når kommandoen for å installere pakker kjøres, og plasserer disse i en mappe kalt 'node_modules'. Denne mappen er ikke inkludert i Git-repoet, men 'package.json' filen er. Dermed kan utviklere enkelt kjøre installasjonskommandoen til npm og få alle avhengigheter installert uten att de blir lagt til i git-repoet.

Pakke	**Lisens**	**Versjon**	**Beskrivelse**
angular-jsdoc	MIT	1.3.3	Jdoc plugin for angular
bower	MIT	1.7.2	pakkehåndering for applikasjonen
eslint	MIT	1.10.3	linting for javascript kode
eslint-plugin-angular	MIT	0.15.0	eslint regler for angular
git-guppy	MIT	1.0.1	git hooks for gulp
guppy-pre-commit	MIT	0.3.0	git pre commit hook for gulp
gulp	MIT	3.9.0	taskrunner / build system
gulp-concat	MIT	2.2.0	gulp plugin for sammenfletting av filer
gulp-cssnano	MIT	2.1.0	gulp plugin for å minimere css filer
gulp-eslint	MIT	1.1.1	gulp plugin for å linte filer med eslint
gulp-protractor	MIT	2.1.0	gulp plugin for kjøring av protractor
gulp-rename	MIT	1.2.0	gulp plugin for å endre filnavn
gulp-sass	MIT	2.1.1	gulp plugin for å kompilere sass
gulp-uglyfy	MIT	1.5.1	minimering av javascript filer i gulp
jasmine	MIT	2.4.1	testrammeverk
jasmine-spec-reporter	Apache 2.0	2.4.0	rapporterer resultater av jasmine tester
jsdoc	Apache 2.0	3.4.0	generering av dokumentasjon
karma	MIT	0.13.19	testrunner
karma-coverage	MIT	0.5.3	genererer testdekningsrapporter fra karma
karma-jasmine	MIT	0.3.6	karma plugin for å benytte jasmine
phantomjs	Apache 2.0	2.1.2	nettleser uten vindu
karma-phantomjs-launcher	MIT	0.2.3	kjører phantom nettleser for karma
protractor	MIT	3.0.0	testrammeverk for e2e tester

Tabell 19: Pakker som kreves for frontend utvikling

I tabell 19 vises de viktigeste modulene vi har benyttet oss av i prosjektet. Disse er moduler som vi benytter til utviklingen til prosjektet da biblioteker som selve applikasjonen krever ble installert ved hjelp av Bower.

Pakke	Lisens	Versjon	Beskrivelse
body-parser	MIT	1.15.0	parser innhold i en HTTP forespørsel
express	MIT	4.13.4	web rammeverk for HTTP kommunikasjon
ioredis	MIT	1.15.1	Javascript driver for redis databaser
jsonwebtoken	MIT	5.7.0	json web token implementasjon
pouchdb-replication-stream	Apache 2.0	1.2.6	replikerings strøm for pouchdb
request	Apache 2.0	2.69.0	Klient som sender HTTP forespørsler
socket.io	MIT	1.4.5	socket kommunikasjon i nodejs
socket.io-client	MIT	1.4.5	klient håndering for socket.io
socket.io-stream	MIT	0.9.0	gjør bruk av streams tilgjegelig i sockets
socketio-auth	ISC	0.0.5	autentisering for socket io kommunikasjon
winston	MIT	2.2.0	avansert logg muligheter for nodejs applikasjoner
pouchdb	Apache 2.0	5.3.1	javascript couchdb database

Tabell 20: Pakker som kreves for å kjøre backend

I tabell 20 beskriver pakkene vi brukte for server siden av applikasjonen. Disse pakkene er runtime avhengingheter.

Pakke	Lisens	Versjon	Beskrivelse
chai	MIT	3.5.0	bibliotek som sjekker data mot forventet resultat i tester
command-exists	MIT	0.1.0	sjekker om en shell kommando finnes i miljøet
eslint	MIT	2.4.0	linting for javascript kode
git-guppy	MIT	1.0.1	git hooks for gulp
gulp	MIT	3.9.1	taskrunner / build system
gulp-eslint	MIT	2.0.0	gulp plugin for å linte filer med eslint
gulp-exit	MIT	0.0.2	gulp plugin for å avslutte en gulp task manuelt
gulp-istanbul	MIT	0.10.3	gulp plugin for generering av testdekningsrapport
gulp-mocha	MIT	2.2.0	kjører mocha tester igjennom gulp
guppy-pre-commit	MIT	0.3.0	git pre commit hook for gulp
memdown	MIT	1.1.2	lar pouchdb lagre all informasjon i minnet istenfor på fil
mocha	MIT	2.4.5	testrammeverk
pouchdb-server	Apache 2.0	1.1.1	pouchdb med HTTP api

Tabell 21: Pakker som kreves for backend utvikling

I tabell 21 er de viktigeste pakkene som krevdes for utvikling av server siden. Flere pakker som er en runtime avhenginghet også en utviklingsavhenginghet men de er da ikke skrevet to ganger.

10.2 Oppgavehåndtering

Gulp ble valgt som program for å automatisere oppgaver i prosjektet. Grunnen til dette var at Ionic allerede benytter seg av Gulp, så det ble lettere å integrere våre egne oppgaver med Ionic sitt kommandolinjeverktøy. Ioinic kommandoer som ”ionic serve” starter en webserver med applikasjonen og åpner den i nettleseren. Denne kommandoen kan kjøre Gulp-oppgaver når webserven starter, og dermed kan oppgaver kjøres hver gang vi skal se arbeidet vi har gjort. Det var tre oppgaver som benyttet seg av dette.

- sass: kompilerer SASS filer til CSS og minimerer dem.
- jsConcat: syr sammen alle JavaScript filene vi har i en stor fil. Det er denne filen applikasjonen bruker når den kjører. Dette lar oss holde koden adskilt under utvikling men å kun inkludere en fil i applikasjonen når den skal kjøres.
- watch: denne oppgaven holder et øye med SASS og JavaScript filer og kjører en av de to forrige oppgavene hvis noen av disse filene skulle forandre seg i mens serveren kjører. Dette gjorde at vi slapp og kjøre de to første oppgavene manuelt hver gang vi endret koden og ville se resultatet.

Oppgaver som ble benyttet under kvalitetssikring var lint, unitTest og e2eTest. Disse kjørte verktøy som eslint, karma og protractor. To av disse ble benyttet en spesiell oppgave som ble kalt av git når en commit skulle gjøres. Denne hette pre-commit og ansvaret var å kjøre lint og unitTest oppgaven hver gang den ble kalt. Dette var et ledd i kvalitetssikringen for å sørge for at slike oppgaver ble kjørt regelmessing og i sammenheng med å laste opp kode til git repoet. Hvis en av de to oppgavene fikk en feil, som for eksempel at en unit test ikke virket ville det stoppe git commit-en. Oppgaven som kjørte protractor for end to end testingen ble brukt hovedsakling under QA delen av en oppgave. En oppgave som ble brukt under testskriving het tdd (test driven development). Denne kjørte unit testene og ventent på endringer filer for selve testen eller koden som ble testet og deretter kjørte testene på nytt.

Øvrige oppgaver er docs som kjører jsDoc som genererer dokumentasjon utfira kommentarer i koden. Vi har også en oppgave som heter uglify som minimerer javascript filer. Denne oppgaven er ment for bruk på den sammenflettede javascript filen som blir generert av jsConcat oppgaven.

10.3 Prosjektstyring og Dokumentasjon

Her blir det sett på hvilke verktøy og metoder som inngår i prosjektstyring.

10.3.1 Repository

Bitbucket er valgt som reporitory for prosjektet, hovedsaklig på grunn av at dette er utviklet av Atlassian som også har utvikler Confluence og Jira som også innegår i prosjektet.

Når både Jira og Bitbucket brukes i prosjektet er det mulig å sammenkobles disse tjenestene med smart-commits. En smart commit begynner alltid med navnet på en oppgave som ligger i Jira. Videre legges det inn en kommentar med hva som er gjort. Dette beskrives med ’#comment’. Til slutt skrives det inn tid med ’#time’.

```
1 git commit -m "BUDGET-165 #comment Added errorchecking to budgetCategory + fixed
      common html styles #time 2h"
```

Kodeutsnitt 10.1: Smart Commit

Smart commits er brukt for å holde kontroll på fremdrift av alle oppaver via Jira, uten å måtte gå via Bitbucket. Under utviklingen har vi da kun hatt behov for å bruke Jira. I tillegg er det lett å holder kontroll over tidsbruk da dette blir rapportert for hver commit.

10.3.2 Confluence

Confluence har blitt brukt til dokumentasjon for egen veiledning, oppbevaring av dokumentasjon og for logg av møtevirksomhet.

Det er opprettet en egen mappe for ressurser hvor alle metoder er beskrevet. Her ligger det retningslinjer for bruk av alle andre verktøy som også er beskrevet i dette kapittelet. Dette innebærer:

- Git: Generell bruk og oversikt over kommandoer
- Git: Retningslinjer og fremgangsmåte for commit, merge og push metoder. Samt hvordan branching skal gjøres.
- QA: Sjekkliste for fremgangsmetode
- JSDoc: Retningslinjer for å dokumentere kode

Mappen for møter inneholder referat fra møter med veilededer, oppdragsgiver og fra egne retrospektive møter som ble gjennomført i slutten av hver milpæl.

Alle felles standarder som ble bestemt under utviklingsfasen ble dokumentert her. Alle dokumentasjon som ligger på confluence er skrevet på en slik måte at hvis prosjektet skal leveres over til andre personer, kan alle metoder finnes igjen her og fortsettes på.

10.3.3 Jira

Jira er prosjektstyringsverkøyet i prosjektet. Her finnes tavlen hvor alle oppgaver er definert. Det brukes samme type tavle som beskrevet i kapittel 5.6 Taskboard.

Backlog	Milestones	Development	QA	Done

Tabell 22: Taskboard

I Jira finnes hovedkategorier som kalles for 'Epic'. Følgende kategorier er opprettet: History, Info / Setting, Login, Savings, UI Specifics, Home, General tasks, Budget, Backend, Intro og Bugs. Disse er opprettet for å kunne holde kontroll, slik at alle oppgaver som opprettes vil tilhøre en kategori eller 'epic'. Når en oppgave opprettes får den også en prioritet som kan være:

- Blocker: Feil som gjør at applikasjonen ikke vil starte eller fungere som normalt.
- Critical: Kritiske oppgaver som kreves for at applikasjonen skal fungere som normalt.
- Major: En viktig oppgave. Hovedsaklig endringer av underliggende funksjonalitet.
- Minor: Mindre viktig oppgave. Hovedsaklig endringer i brukergrensesnittet.
- Trivial: Ekstraoppgaver som er kjekt å ha med, men som ikke kreves for vanlig drift av applikasjonen.

Det er hovedsaklig prioritetene 'Major' og 'Minor' som er blitt brukt. Grunnen til dette er gruppe gode rutiner for kvalitetssikring som har gjort det mulig å unngå spesielt oppgaver prioritert som

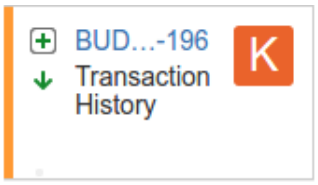
'Blocker'. 'Trivial' har også blitt brukt en del.

Hver oppgave får også et eget navn når den blir opprettet. Alle oppgaver tildeles navnet 'BUDGET-xx' hvor 'x' tislvarer et nummer. Nummer på oppgavene blir ikke definert selv, da dette er noe Jira bestemmer. Oppgavenumrene blir tildelt fortløpende ettersom oppgavene lages. Vi gir hver oppgave et konkret navn. Eksempel:

- BUDGET-196 - Transaction History
- BUDGET-193 - Google refreshtoken from backend
- BUDGET-116 - Login and registration (View)

Samme navngiving benyttes også når en oppgave får tildelt sin egen branch i repository. En branch bruker kun nummereringen og ikke beskrivelsen (Eksempelvis BUDGET-116).

Hver oppgave har også en tilhørende eier, som vil være den som opprettet oppgaven. Her er det ikke definert en spesifikk person som har ansvar for å opprette oppgaver, da alle kan gjøre dette. Når en oppgave ligger i 'Backlog' eller 'Milestones' er den ikke tildelt en person. Men i det en oppgave settes til 'Development' kreves det at noen tildeler den til seg selv eller en annen etter avtale.



Figur 40: Oppgave i Jira

Figuren ovenfor viser en konkret oppgave hentet ut fra tavlen i Jira. Helt til venstre i figuren ligger en fargekode som tilsier hvile kategori eller 'epic' den tilfører. Videre ligger to ikoner: et pluss-tegn som tilsier at dette er Ny funksjonalitet og en grønn pil som tilsier at oppgaven har prioriteten 'Minor'. I tillegg vises 'BUDGET-196 - Transaction History' som er navnet. Ikonet på høyre side viser hvem som har oppgaven tildelt.

Til å begynne med ble oppgaver tildelt til en person og dette ble stående hele veien fre 'Development' til 'Done'. Etter første retrospektive møte ble dette endret på slik at kun den som faktisk jobber på oppgaven der og da vil stå som oppført. Dette fungere slik at når oppgaven flyttes fra 'Development' til 'QA' så vil den stå uten tildeling i vente på at en annen person skal utføre kvalitetssikring på den. Ved å bruke en slik tilnærming er det alltid klart hvilke oppgave som jobbes med og hvilken oppgave som står ledig.

10.3.4 JSDoc

JSDoc er brukt til å kommentere all JavaScript-kode som er skrevet. Innen JSDoc finnes det allerede fastsatte regler for hvordan dette skal brukes [38]. Kun de mest nødvendige reglene ble brukt.

Alle JavaScript-filer med kode skal inneholde utviklerens navn og dato for opprettelse. Dette skal stå øverst.

```
1  /**
2   * @author Kristian Dragerengen
3   * @since 28.01.2016
4   */
```

Kodeutsnitt 10.2: JSDoc: Navn og dato

Alle funksjoner kommenteres. Kommentaren skal begynne med en generell beskrivelse av hva funksjonen gjør. Videre skal alle parameter kommenteres, også med en beskrivelse hver og hvilke type det er.

```
2  /**
3   * Return the index of a specific item in a array
4   * @param {object} item The choosen item
5   * @returns {number} Index of Item
6   */
7
8  function getIndex(item) {
9    return vm.cards.indexOf(item);
10 }
```

Kodeutsnitt 10.3: JSDoc: Eksempel på kommentar

Alle funksjoner skal også ha en kommentar som tilsier hva som returneres etter funksjonen er kjørt.

Følgende 'tags' er brukt:

- @author - Navnet på vedkommende som har skrevet kode.
- @param - Parameter til funksjoner.
- @returns - Returnverdi for funksjon.
- @todo - Brukes til å kommentere endringer som må gjøres.
- @tutorial - Henvisning til ekstern lenke for informasjon.

Når det gjelder bruk av '@param' så er det også nødvendig å angi hvilke type parameter dette er:

- {number}
- {text}
- {object}
- {service}
- {factory}

Ved å angi type parameter er det også lettere å bruke funksjoner uten å vite hvilke parametere den skal ha på forhånd. I utviklerverktøy brukt under prosjektet vil det komme opp automatiske forslag, her inngår også paramtertype når en funksjon blir navngitt. Kommentarer er også inkludert slik at den som programmerer slipper å lete opp funksjonen som skal brukes.

11 Sikkerhet

11.1 Brukerdata

Her beskrives hva slags brukerdata applikasjonen tar vare på og hvordan dette er håndteres sikkerhetsmessig.

11.1.1 Personopplysninger

Ved bruk av applikasjonens system for innlogging og registrering er det et krav å registrere navn, epost og passord. Det er ikke et krav at brukeren skriver inn sitt fulle navn. Det blir heller ikke sjekket om brukeren skriver inn sin egen epostadresse eller om det er en gyldig adresse. At e-post adressen ikke blir verifisert er ikke gunstig. Å verifisere denne med en epost under registrering er noe som hadde er viktig å implementere ved videre utvikling.

Når det gjelder eksterne påloggingsmetoder som Facebook og Google blir det ikke gitt tilgang til annet enn brukerens e-postadresse og fulle navn.

11.1.2 Krav til brukerdata

Dette gjelder krav til brukeren navn, e-post og passord for appliksasjonens system for brukerregistrering.

Navn

Navn er en tekststreng som skal være innenfor en gitt lengde på minimum 3 og maksimum 35 tegn. Brukeren kan potensielt benytte seg av hele ASCII-alfabetet. Her kunne det med mer tid blitt laget mekanismer som sikrer at navnet ikke inneholder spesialtegn ved bruk av regulere uttrykk.

Epost

Epostaddresse blir sjekket etter følgende regulert uttrykk.:

```
1 var emailPattern = new RegExp("^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@
      ((\\[[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\])|(([a-zA-Z
      \\-0-9]+\\.)+[a-zA-Z]{2,}))$");
```

Kodeutsnitt 11.1: Validering av epostadresse

Her blir det sjekket om brukeren skriver inn en epost i formatet ”Ola@Nordmann.no”. Innholdet før ”@” kan inneholde store og små bokstaver, tall og enkelte spesialtegn. Innhold etter ”@” kan inneholde store og små bokstaver inkludert tall. Innholdet etter ”.” gjelder kun store og små bokstaver.

Passord

Kravet til brukerens passord er at det inneholder et minimum på 8 tegn fra ASCII-alfabetet [39].

11.2 Personvernerklæring

Det er utarbeidet en egen policy med regler for personvern. Denne finnes i vedlegg I. Denne er til for å informere brukeren om hvordan data som lagres i applikasjonen både lokalt og sentralt

behandles, brukes og hvem som har tilgang til det. Personvernsregler skal også ligge inne på selve applikasjonen, lett tilgjengelig for brukeren.

11.3 Autentisering

Vi benytter oss av totalt tre forskjellige måter å logge inn på, men vi kan overordnet si at de krever seks ting for å fungere.

- Navn for å kunne søke opp navnet for deling av budsjettposter.
- Epostadresse/unik brukerid for å lage unik database, logge inn på tjenestetilbydere inkludert applikasjonens.
- Passord for å logge på tjenestetilbydere inkludert applikasjonens.
- Token for å autentisere seg mot tjenestetilbydere inkludert applikasjonens.
- Utløpstado for token. Vi vil ikke at tokens skal leve evig.
- Type tjenestetilbyder

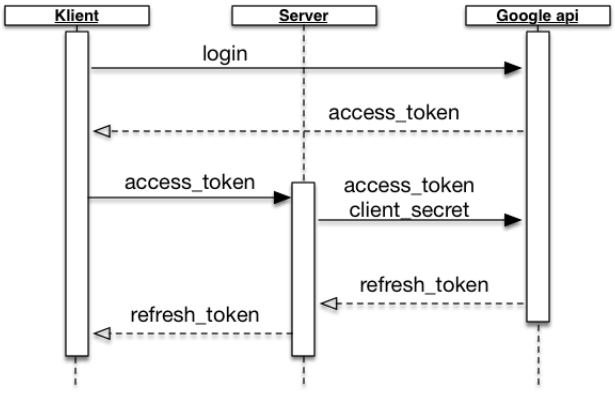
Disse blir lagret i den lokale databasen og blir benyttet under innloggingssekvensen (se kapittel 11.3.3 Innlogging). Passordet blir bare lagret hvis applikasjonens egen registreringsfunksjon benyttes, og da blir det hashet.

11.3.1 Oauth login

Brukere kan velge mellom to tredjeparts tjenester for innlogging i vår applikasjon. Disse er Google og Facebook.

Begge tilbyr en SDK for å gjøre innloggingsprosessen lettere, men vi fant noe av funksjonaliteten til å være mangelfull for vår bruk. Vi bestemte oss for å implementere det på egenhånd uten SDK. Vi lagde en tjeneste som hadde ansvaret for å logge inn på begge tjenestetilbyderene. Det vi trengte som skulle komme fra tjenestetilbyderene var navn, epost og en autentiseringstoken. For begge trengte vi ikke be om ekstra tillatelse for å få navn og epost eller autentiseringstoken. Selv om vi fikk autentiseringstoken fra tjenestetilbyderene, hadde de en utløpstid som gikk ut for tidlig til at det kunne være til praktisk nytte for oss.

Forlenget autentiseringstoken fra Google og Facebook



Figur 41: Sekvensdiagram for forlenget autentiseringstoken fra Google

For å hente ut en forlenget autentiseringstoken fra Google og Facebook er vi avhengig av en 'client secret' som vi får når vi registrerer applikasjonen i Google og Facebook. Denne må være

hemmelig og holdes unna kildekoden til selve applikasjonen. Løsningen ble å be serveren hente forlengede autentiseringstoken og sende den tilbake til applikasjonen. Den sender også tilbake en utløpsperiode som vi kan følge med på. Hvis utløpsperioden nærmer seg slutten, kan vi be om å få tilsendt en ny forlenget autentiseringstoken. Dette er implementert for Facebook men dette må også gjøres for Google.

Det er backend som sitter med informasjonen om applikasjonen, slik som applikasjons id og client secret, og står for kommunikasjon med tredjeparts tjenestetilbyder med kritisk informasjon.

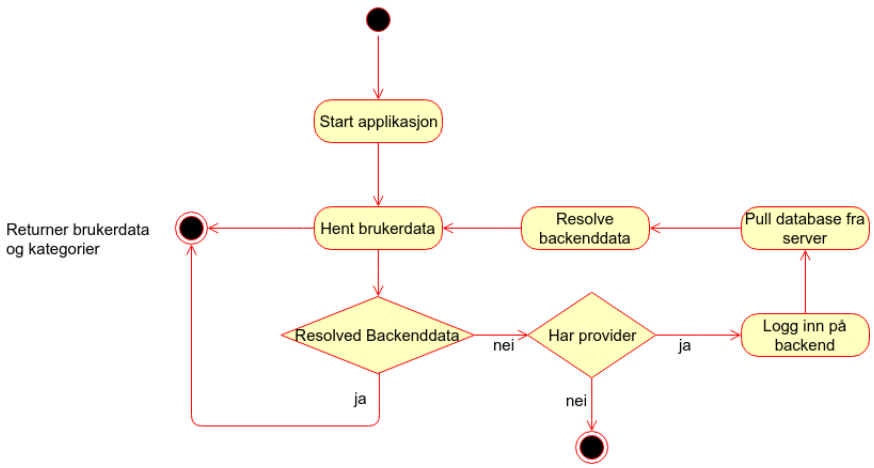
11.3.2 Brukerregistrering med applikasjonens registreringsmekanismer

Brukere som ikke ønsker å logge inn med brukerkontoer fra andre tjenester kan opprette en bruker for vår applikasjon. Selv om det virkelige navnet til brukeren ikke kreves, er det oppfordret ettersom navn brukes til å søke opp andre brukere for deling av budsjettposter. Som nevnt tidligere i dette kapittelet er det heller ikke et krav at brukeren legger inn gyldig epost, men et gyldig format. Siden epostadresser alltid er unike er registrering med epostadresse et krav. Brukere må også legge inn passord. Hensikten med passordet er ikke for å sikre applikasjonen i tilfelle noen får tak i telefonen, men for å la brukeren benytte sin konto på flere enheter. Passordet lagres i databasen etter det har blitt hashet med en SHA-512 algoritme for å sikre det. Brukerens data registreres deretter til serveren med en "register" forespørsel (se Figur 43) og får tilbake 200 hvis databasen ikke eksisterte og ble lagd for brukeren eller 202 hvis databasen allerede fantes med den epostadressen. Ut ifra denne responsen kan man avgjøre hva som skal gjøres videre.

Innloggingsdataene blir slik systemet fungerer idag synkronisert med resten av dataene til den sentrale serveren. Dette er ugunstig og unødvendig siden brukere allikevel må fylle dette inn ved bruk av en ny enhet. Vi har flere løsninger på dette. Det er mulig å filtrere ut denne informasjonen i replikeringsprosessen eller oppbevare det i i en egen adskilt PouchDB database på enheten som også kan være kryptert for ekstra sikkerhet.

11.3.3 Innlogging

En registrert bruker vil automatisk bli logget på hver gang applikasjonen åpnes. Brukeren ønsker ikke å aktivt logge seg på for hver gang, men heller ikke at applikasjonen må vente på at innloggingssekvensene kjører ferdig.

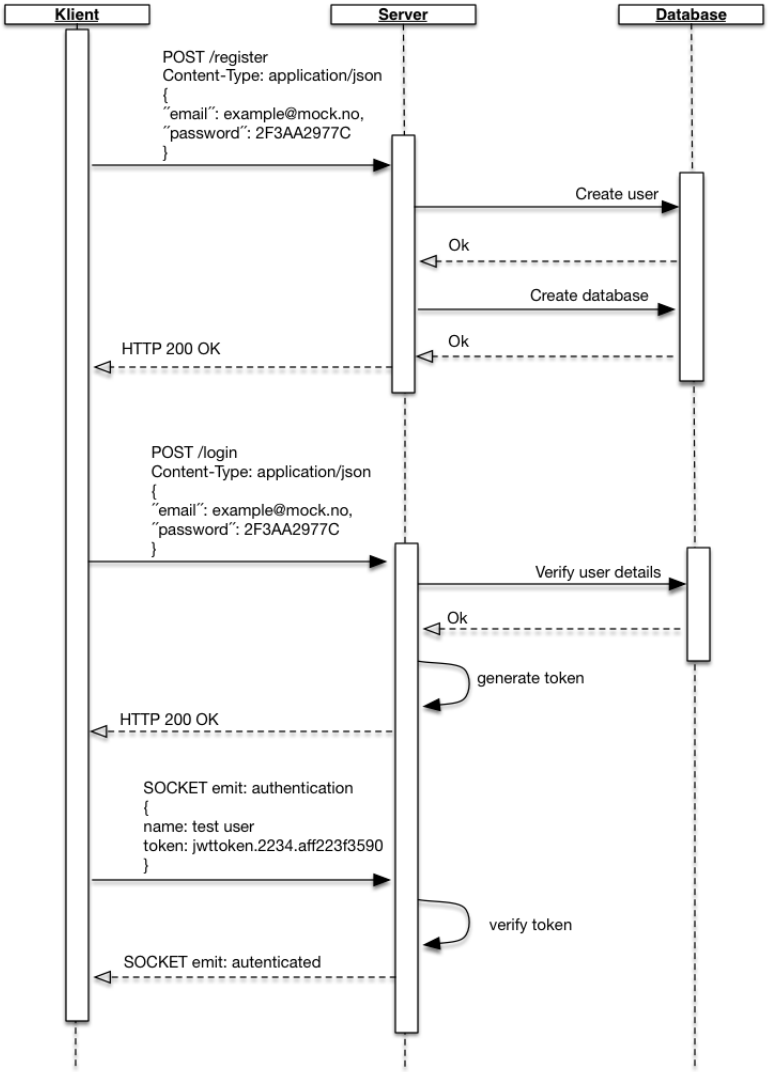


Figur 42: Innloggingsflyt

Dette er løst ved å hente brukerdata fra den lokale databasen og returnere det med en gang, og deretter starte en innloggingssekvens. Dataen fra databasen sjekkes om de inneholder en tjenestetilbyder. Hvis den ikke inneholder tjenestetilbyder avsluttes sekvensen. Under innloggingen til backend sjekkes utløpsdatoene for autentiseringstoken og eventuelt oppdaterer disse. Når en socket-kobling mot serveren er opprettet, replikeres den sentrale databasen mot den lokale. Hvis noen andre har lagt til nye utgifter i delte budsjettposter vil modellen også oppdateres. Eventuelt oppdaterte brukerdata blir da sendt med til ”hent brukerdata” som returnerer de oppdaterte brukerdataene

11.3.4 Autentisering på backend siden

Autentiseringen her foregår via CouchDB sin autenstiseringsmekanisme for brukere av databasene. Når backend-en mottar en HTTP-POST-forespørsel på <server-IP>/register med et brukernavn og passord vil backend-en opprette en bruker i CouchDB i tillegg til en database for denne brukeren. Brukeren kan da logge inn med en HTTP POST forespørsel til <server-IP>/login med samme brukernavn og passord som ble brukt til å registrere seg. Serveren vil da generere en JSON web token [40]som sendes til brukeren. Denne tokenen blir signert av serveren og vil inneholde brukerens epost adresse. Deretter kan brukeren koble seg på en socket og sende tokenen den fikk fra inloggingen. Denne blir da versifisert av serveren og brukt til å identifisere brukeren utifra epost adressen i tokenen.



Figur 43: Registrering og autentisering for brukere

Den kristiske informasjonen her er passordet til brukerene. For å minimere antall ganger passordet sendes frem og tilbake mellom klient og server har vi benyttet oss av json web tokens. Disse vil bli lagret på enhetene og vil bli brukt for å få tilgang til en socket når brukeren starter applikasjonen. En token er gyldig i 60 dager. Etter dette vil klienten be om å få en ny token tilsendt ved en ny forespørsel til /login. 60 dager blir brukt siden en long access token fra facebook varer i 60 dager [41]

Når en token sendes ut vil denne bli signert av serveren med en kode som består av 256 tilfeldige tegn. Denne kan bli generert på forskjellige måter som er tilstrekkelige unike for dette bruket.

```
1    #via linux urandom
2    $ < /dev/urandom tr -dc _A-Z-a-z-0-9 | head -c 256 > hemmelig.fil
3
4    #via openssl
```

```
5    $ openssl rand -base64 256 > hemmelig.fil
```

Kodeutsnitt 11.2: Generering av jwt kode

Secret-en skal leses fra fil når backend-en starter. Den samme koden blir brukt for å sjekke om tokenen er gyldig ved senere autentisering. Som vist på figur 43 blir brukernavn og passord brukt for å få tilsendt en token.

11.4 Sikkerhetstiltak

En del sikkerhetstiltak ble gjennomført og planlagt for backend-en. Vi benyttet oss av anbefalinger og best practices for sikkerhet i Node.js som vi mente var relevente for vårt system [42]. Dette var for å sikre brukerdataene og den sentrale delen av systemet.

Et av tiltakene var å bruke et kommandolinjeverktøy kalt nsp [43] som sjekker package.json filen mot kjente sårbarheter i node moduler.

Flere tiltak som er relevante men ikke ble implementert av tidsmessige årsaker er å fjerne X-Powered-By headeren som gir informasjon om hvilken teknologi som brukes på serveren. Denne informasjonen kan brukes til å utføre målrettede angrep mot serveren basert på programvaren den benytter. Derfor er det gunstig å fjerne denne headeren fra svarene serveren gir. En NodeJS-modul vi også ville benyttet som også hjelper med sikkerheten er express-rate-limit. Denne begrenser antall ganger serveren svarer på HTTP-forespørsler fra samme klient. Denne grensen er aktuell for forespørsler som omhandler registering, innlogging og anskaffelse av tokens som varer lenger enn den brukeren får ved førstegangs innlogging for tredjeparts innloggingstjenester. Disse handlingene blir ikke utført ofte og grensen på antall forespørsler som kan sendes kan derfor være lav. For eksempel gir /login ut en token som varer i 60 dager så denne vil sjelden bli brukt av samme personen. En grense på 10 forespørsler fra samme klient i løpet av en time er en bra grense basert på disse faktorene. Dette begrenser effektiviteten av 'brute force'-angrep for gjetting av passord samt begrenser mulighetene for å overbelaste systemet med f.eks en stor mengde forespørsler for registrering.

I stedet for å blokkere brukerkontoen som angripes blokkeres klienten som prøver å logge seg inn. Dersom brukerkontoen ble blokkert kunne angriperen hentet gyldige brukernavn utifra hvilke som ble blokkert. Siden innlogging foregår via NodeJS videre til CouchDB vil feil brukernavn og/eller passord gi eksakt samme respons til klienten. Dette gjør at en angriper ikke kan hente nyttig informasjon utfra responser på forespørsler. Et annen problem med å låse brukerkontoen er at dette kan brukes som tjenestenektangrep mot brukere for å stenge dem ute fra sine kontoer. Dette tiltaket dekker derimot ikke distribuerte brute force-angrep som utføres ved hjelp av en liste med proxy-er som angriperen rullerer på å bruke [44].

Et annen svert kritisk tiltak og implementere videre er SSL for kommunikasjonen mellom server og klient. Dette vil være både for socket og HTTP trafikken. Passord og json web tokens gir brukere tilgang til sine data og det er dermed viktig at disse ikke sendes i klartekst.

12 Sammendrag

12.1 Resultater

Vi har klart å implementere all hovedfunksjonaliteten som ble definert i samarbeid med oppdragsgiver. Dette er snakk om budsjetthåndtering på enkeltmannsnivå. En del av den mer avanserte funksjonaliteten er også implementert. Dette gjelder en serverside av applikasjonen som lar brukere logge inn og få tilgang på sine data på flere enheter, i tilleg å kunne dele budsjettposter mellom seg. Den mer avanserte delen ble påbegynt så snart vi hadde kontroll på hovedfunksjonaliteten.

Strukturen på applikasjonen og kildekoden har vært god gjennom hele prosjektet. Retningslinjene og strukturen vi avtalte tidlig hjalp oss i å holde prosjektet strukturert. Dette gjorde at det var lett for oss å refaktorere og legge til funksjonalitet selv om kodebasen har blitt mye større og inneholder et relativt stort antall filer.

Mye arbeid har blitt lagt i å utforme brukergrensesnittet. Dette er inspirert av Android-applikasjonen som tidligere ble utviklet, og ble gradvis forbedret gjennom hele prosjektperioden. Brukertestene vi gjennomførte påvirket også utseendet og funksjonaliteten til grensesnittet.

På serversiden av applikasjonen har vi oppnådd kravene for den mer avanserte funksjonaliteten for applikasjonen. Mye prøving, feiling og forstudier ble gjort før vi kom fram til den endelige løsningen med både et HTTP-API og et Socket-API. Mindre rammer og planlegging ble gjort på denne siden før vi satte i gang med selve utviklingen og derfor ble dette en mer dynamisk prosess der strukturen og funksjonaliteten ble endret ved behov. De rammene og prosessene som var relevant fra frontend utviklingen ble allikevel benyttet. Resultatet av dette er en server med funksjonaliteten vi behøver samt mekanismer for avdekking av feil og en struktur som er proporsjonal i forhold til behovene slik applikasjonen står i slutten av prosjektperioden. I forhold til skalering er dette noe vi har hatt i tankene under utviklingen. Dette har påvirket hvordan vi har løst oppgaver da vi ikke bare skal finne en løsning som fungerer når vi prøver dette i liten skala under utvikling En av disse løsningene er å bruke Redis in-memory-database for rask aksessering av data som er sentral for driften av serveren. Vi testet socket delen av serveren som vil bli hyppigst brukt med en stress test hvor 400 brukere logget inn iløpet av 4 sekunder , noe som gikk bra.

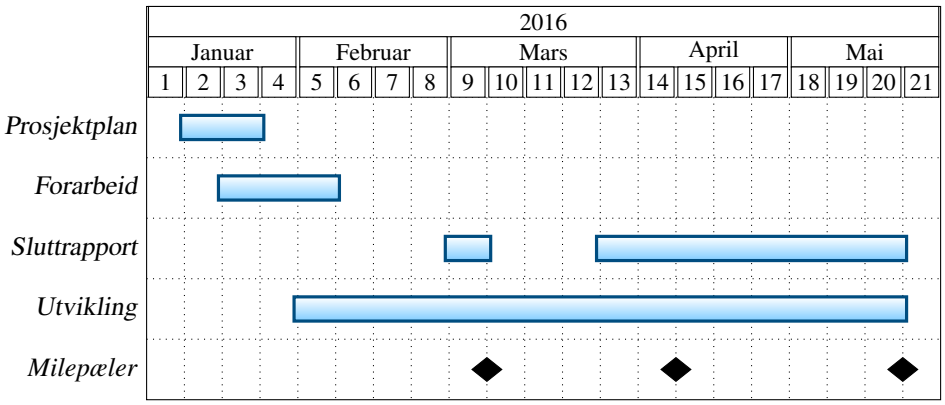
12.2 Evaluering av eget arbeid

12.2.1 Plan og utviklingsmodell

Prosjektet ble startet opp den 11. januar 2016. I oppstartsfasen var prosjektplanen prioritert de to første ukene. Uken etter ble det i tillegg startet opp research, opplæring og forarbeid som pågikk parallelt med prosjektplanleggingen. Etter tre uker ble utviklingen også påbegynt. Den første uken av utviklingen inneholdt hovedsaklig implementering av forarbeid og research som ble gjort. Utviklingsfasen nådde aldri en slutt og pågikk helt til innleveringsdato (deadline). Når

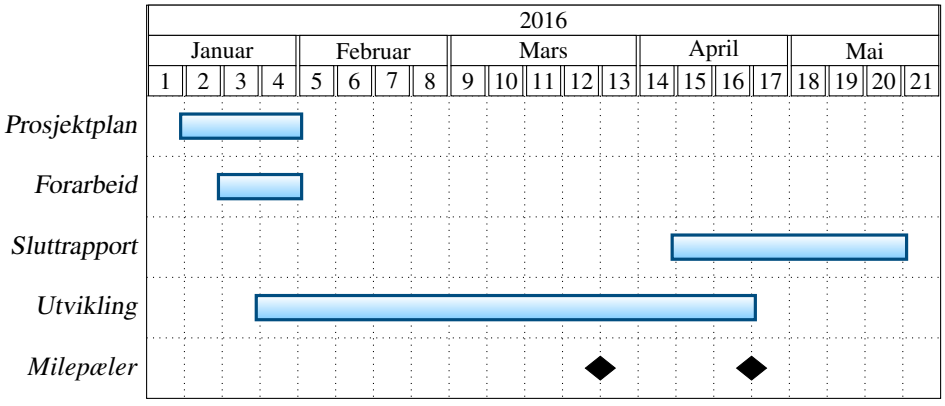
det gjelder sluttrapporten ble denne påbegynt i uke 9, hvor det hovedsaklig ble jobbet med å strukturere rapporten. Selve rapportskrivingen ble satt i gang i uke 13.

I tidslinjene nedenfor vises bare hovedaktiviteter. Aktiviteter som møter, brukertesting, rapportering etc vil likevel bli kommentert. Opplæring og research inngår i forarbeid.



Figur 44: Tidslinje - Etter

Begge tidslinjene inneholder like periode for prosjektplan, opplæring og forarbeid. Den store forskjellen utgjøres ved sluttrapporten og utviklingsperioden. I tidslinjen nedenfor som representerer planen, skulle utviklingsprosessen avsluttes fire uker før deadline. Realiteten var at utviklingsprosessen ikke ble avsluttet da, som vist i tidslinjen ovenfor som representerer resultatet. Planen var også å ha en overlappende periode hvor utviklingen gradvis ble trappet ned, samtidig som rapportskriving ble trappet opp og var full fokus de siste fire uker. Resultatet ble heller at rapporten ble påbegynt tidligere og ble jobbet med parallelt med utvikling til deadline. Når det gjelder milepæler ble det planlagt to stykker. Den første definerte budsjett for enkeltpersoner, mens den andre definerte budsjett for flere brukere. Derimot endte vi opp med tre milepæler hvor budsjett for enkeltpersoner inngikk i de to første, mens budsjett for flere brukere inngikk hovedsaklig i den siste (Server siden ble påbegynt i andre milepæl).



Figur 45: Tidslinje - Før

Det ble planlagt fem møter med oppdragsgiver. Det første møtet skulle være et innledningsmøte i starten av hele prosessen. De fire neste skulle komme annenhver uke i utviklingsfasen. I tillegg ble det planlagt å levere fem statusrapporter i samme tidsrom som disse møtene. Statusrapportene skulle også leveres annenhver uke i tidsrommet for utviklingsperioden. Det var i tillegg planlagt å holde brukertester for hver milepæl.

Mye av grunnen til valg av en smidig utviklingsmetodikk uten større mengder fastsatte regler var på grunnlag av behovet av å kunne endre på plan og metoder underveis. Når det gjelder møter med oppdragsgiver ble det holdt et innledningsmøte som planlagt, etterfulgt av et møte etter første milepæl. Statusrapporter ble nedprioritert, da det kun ble utstedt en enkel i slutten av første milepæl. Grunnlaget til at det ble holdt færre møter og utsted færre statusrapporter er at kommunikasjonsmodellen endret seg underveis i prosjektet. Etter de to første møtene mente vi det å holde et møte ikke gav nok i forhold til det vi fikk tilbake, slike at denne kommunikasjonen kunne gjøres i andre kanaler som var mindre tidkrevende. I steden for statusrapport har vi hatt ukentlige møter med veileder, og kommunikasjonen med oppdragsgiver ble utført ved bruk av e-post. Oppdragsgiver i har tillegg hatt mulighet til å prøve ut applikasjonen gjennom egne løsninger for Ionic. Disse gav oss muligheten til å dele applikasjonen under utvikling for å få tilbakemelding. Oppdragsgiver kunne da prøve ut siste oppdatering av applikasjonen rett på sin egen mobiltelefon når det selv passet dem. Planen ble ikke fulgt til den grad den var satt opp, men dette var tatt høyde for i utviklingsmodellen.

12.2.2 Milepæler

Tidsestimering for milepæler har ikke fungert etter planen. Det ble i starten av hver milepæl definert en mengde oppgaver som skulle løses. Etterhvert som oppgaver ble løst, ble det i tillegg lagt inn tilsvarende oppgaver for forbedringer eller feilløsning til de forhåndsdefinerte oppgavene. Dette medførte at antall oppgaver gikk over hva som var planlagt og estimert. I tillegg kunne også oppgavefordelingen bli skjev slik at enkelte oppgaver var mye større enn andre. Dette førte til at enkelte satt med store oppgaver og andre satt med små. På slutten av en milepæl kunne da enkelte sitte uten arbeid mens andre hadde arbeid igjen. For å utlikne arbeidet ble det da laget tilleggsoppgaver eller flyttet inn andre oppgaver fra produktets backlog. Dette er hovedgrunnen til at utviklingsperiodene ble lengre.

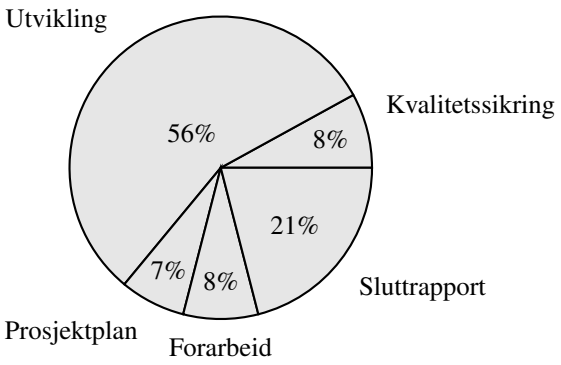
12.2.3 Gruppen

Gruppesamarbeidet har fungert godt. Vi har hatt en god dynamikk innad i gruppen. Alle sammen har vært fokusert på sine oppgaver sammen med gruppens målsetning. Valg og beslutninger underveis har vært preget av en demokratisk styreform hvor alle har hatt mulighet til å komme med sine innspill før et valg er blitt tatt. Gruppeleder har ikke hatt ansvaret for å ta spesifikke valg, men heller å sørge for at alle blir hørt slik at en beslutning blir foretatt sammen.

Når det gjelder timefordeling i gruppen har vi ingen helt eksakte data, men basert på timelogg har alle bidratt omtrent like mye. Det er vanskelig å sette et skille. Arbeidet har i alle hovedsak fungert i fellesskap, men det er også en del individuelle timer som er ført. Individuelt arbeid er noe alle har gjort.

12.2.4 Arbeidsfordeling

Grafen i figur 46 viser den totale oversikten av arbeidsfordeling gjennom hele prosjektperioden.



Figur 46: Fordelinge av timer

Grafen viser at mest tid er brukt til utvikling. I denne posten inngår all programmering, i tillegg til retting av kode i forbindelse med koderevisjon. Retting og utbedring av kode burde vært logget som Kvalitetssikring. Kvalitetssikring inneholder kun andelen tid som er blitt brukte til koderevisjon og brukertesting. Opplæring og research inngår i andelen med forarbeid.

12.3 Kritikk av oppgaven

Under dette punktet reflekteres det over uheldige valg vi tok i løpet av utviklingsperioden. Disse valgene hadde ikke nødvendigvis store følger, men kan ha påvirket utviklingsfasen og dens struktur.

Antallet møter med oppdragsgiver kunne vært høyere. Det lave antallet kan forsvares med at vi utviklet en prototype av applikasjonen i et tidligere prosjekt, og dermed hadde god forståelse av hva oppdragsgiveren ønsket og hva oppgaven gikk ut på. Mye av tiden i andre halvdel av utviklingsperioden gikk til utvikling av backend, noe som oppdragsgiver ikke hadde mange meninger om, i tillegg som backend-programmering er vanskelig å presentere visuelt.

Oppdragsgivers fokus var design og brukervennlighet, og vi fant etterhvert ut at tid ville spares ved å kommunisere via e-post. Samtidig som det ikke var direkte behov for mange fysiske møter og det er positivt å spare tid, kunne forholdet til oppdragsgiver fått et mer profesjonelt preg ved å holde fysiske møter oftere.

Målet vårt om testdekning på 80% ble ikke nådd. Mye tid ble brukt på revisjon av andres kode, og noe av denne tiden kunne blitt brukt på testskriving.

Vi kunne også bedt om hjelp til testing av asynkron funksjonalitet. Dette var noe som var vanskelig å gjøre i praksis selv om testrammeverket skulle støtte dette. Denne funksjonaliteten er ofte ikke triviell og det er derfor viktig at den blir testet.

Vi mangler noe viktig funksjonalitet i tillegg til en del trivielle oppgaver som var ønsket å få med i produktet. Dette har vi ikke fått tid til å implementere grunnet feilprioritering av oppgaver.

Et eksempel på dette er problemer som oppstår når det replikeres mellom databaser på dokumenter som er delt mellom flere brukere. Da kan det oppstå konflikter der begge brukerene har gjort endringer samtidig. Dette resulterer i at en av brukernes endringer blir overskrevet. Det burde vært flere mekanismer på plass som håndterte dette, i tillegg til enhettester som forsikret at disse fungerte.

12.4 Videre utvikling og utvidelse

Her presenteres viderearbeid som eventuelt kan være aktuelt for applikasjonen.

OCR

Optisk tegngjenkjenning er en teknikk som kan automatisert prosessen ved å legge inn utgifter for brukeren. Isteden for at brukeren manuelt må skrive inn en sum tilknyttet en utgift vil det være mulig å få tilgang til telefonens kamera og ta bilde av en kvittering hvor beløpet og metadata til en utgift står. I tillegg til at totalbeløpet automatisk settes inn kan også bildet av kvitteringer lagres slik at brukeren kan hente dette opp igjen senere. Selve prinsippet bak OCR vil ikke hjelpe applikasjonen i å nå sitt mål om effektivitet, da en registrering med kamera vil ta lengre tid med tanke på tidsbruk for oppgaver brukeren må igjennom for å legge til en utgift. Dette er også noe av grunnen til at dette er en utvidelsesmulighet og ikke en prioritert oppgave.

Tilknytning til bankkonto

Oppkobling mot brukerens bankkonto var i prosjektperioden uaktuelt, men eventuell implementering av dette i ettertid tid vil åpne for et bredt spekter av funksjonalitet. Brukerens transaksjoner kan potensielt leses ut fra loggen i nettbanken, noe som kan spare brukeren for manuelt arbeid. Dette gjelder også for gjentagende utgifter som kan hentes fra brukerens avtalegiroavtaler og faste oppdrag. Siden alle transaksjoner vil ligge i nettbankens logg, vil dette kunne sikre mot mørketall, ettersom brukeren kan glemme å legge inn utgifter og lignende.

Funksjonaliteten i modulen for sparemål kan også forbedres ved en oppkobling mot bankkonto. Her kan det settes av automatisk overføring fra brukskonto til sparekonto, basert på brukerens sparemål i budsjettapplikasjonen.

Metoder for kunstig intelligens kan også være aktuelt om applikasjonen kunne fått tak i transaksjoner som skjer tilknyttet en konto. Ved bruk av mønstergjenkjennelse kunne prosessen blitt automatisert over tid, slik at brukeren hadde sluppet å registrere all data selv.

Oppkobling mot betalingsapplikasjoner

Det finnes allerede applikasjoner som kan brukes til å betale med mobiltelefonen i butikker. En mulighet her er at betalingsapplikasjonene kan dele metadata fra betalinger med budsjettapplikasjonen. Dette vil spare brukeren for manuell registrering av transaksjoner. Det vil trolig kreve tett samarbeid med utviklerne av betalingsapplikasjonen.

Deling av sparemål

Deling av sparemål er en utvidelsesmulighet som ville være naturlig å implementere. Ettersom deling av budsjettposter allerede er støttet, ville sparemål kunne implementeres på samme måte. Sparemål kan også utbedres på en måte som lar brukerne selv skrive inn hvor mye penger de setter av, og på den måten manuelt øke progresjonen mot målet.

Lazy caching mekanisme

Noen steder i applikasjonen kaller kontrollerene på enten eksterne eller interne filer som den parser. En lazy caching mekanisme vil være en tjeneste som kalles fra disse kontrollerene som gjør HTTP-forespørslene på vegne av kontrollerene når det trengs og lagrer resultatet. Neste gang kontrolleren trenger data fra fila vil den gi den det lagrede resultatet.

Administrasjonspanel

Med tanke på at oppdragsgiver ønsker kontaktinformasjon inne i applikasjonen kan dette medføre at brukere kontakter banken for kundeservice i forbindelse med bruk av applikasjonen. I et slikt tilfelle hadde det vært aktuelt at personell som betjener kundeservice også hadde hatt mulighet til å administrere data som er laget sentralt.

12.5 Konklusjon

Utviklingen av en applikasjon som skal være lettvint å bruke og retter unges oppmerksomhet mot sin personlige økonomi har vært spennende. Dette har utfordret oss som programvareutviklere på flere områder. Prosjektet har vist seg å være mer omfattende enn det i utgangspunktet skulle tilsi. Mye grunnet vårt fokus på enhetstesting, kvalitetssikring, backend-løsning og brukergrensesnitt. Vi har fått erfaring med både nye og veletablerte rammeverk og biblioteker, samtidig som vi også har måttet tilpasset funksjonaliteten rundt enkelte av disse for vår bruk.

Beslutningen om å utvikle en hybrid applikasjon var avgjørende for læringsutbyttet. Ettersom det ble utviklet en prototype for Android i forkant av bachelorprosjektet, hadde vi allerede lært mye rundt native mobilutvikling for denne platformen. Utviklingen av en hybrid løsning har gitt oss verdifull kunnskap rundt teknologi, verktøy og rammeverk for både webutvikling og mobilutvikling. I tillegg har løsningen latt oss utforske fordelene med velstrukturerte applikasjoner.

Sikkerhet har vært et viktig aspekt av prosjektet ettersom systemet har en serverside med kritisk funksjonalitet. Oppdragsgiver ønsket et produkt som var sikkert og hadde et profesjonelt preg på brukergrensesnittet. Mye arbeid ble lagt ned for å tilfredstille disse ønskene.

Erfaringen av å arbeide på et stort prosjekt over lang tid har gitt oss et nytt perspektiv på profesjonell programvareutvikling. Dette setter vi stor pris på, ettersom det har vært en tankevekker, og vil prege våre arbeidsmetoder i fremtiden. Som en avslutning for våre bachelorgrader er vi svært fornøyde med prosjektet som helhet, og har store håp for dets videreutvikling og nytteverdi for forbrukere.

Bibliografi

[1] Unge nordmenn har 1 milliard i kort-gjeld. [Internettside]. [Sist oppdatert: 11.06.2015, Sitert: 14.03.2016]. Tilgjengelig fra: http://www.nrk.no/norge/unge-nordmenn-har-1-milliard-i-kort-gjeld-1.12403922.

[2] Norske ungdommer drukner i gjeld. [Internettside]. [Sist oppdatert: 2013, Sitert: 14.03.2016]. Tilgjengelig fra: http://www.nettavisen.no/na24/norske-ungdommer-drukner-i-gjeld/3648351.html.

[3] Lærer ungdom om pengebruk. [Internettside]. [Sist oppdatert: 04.04.2016, Sitert: 14.03.2016]. Tilgjengelig fra: http://www.nrk.no/ho/laerer-ungdom-om-pengebruk-1.12883281.

[4] Referansebudsjett for forbruksutgifter 2016. [Internettside]. [Sist oppdatert: 2016, Sitert: 06.05.2016]. Tilgjengelig fra: http://www.sifo.no/files/Referansebudsjett2016.pdf.

[5] Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger. [Internettside]. [Sist oppdatert: 2013, Sitert: 06.05.2016]. Tilgjengelig fra: https://lovdata.no/dokument/SF/forskrift/2013-06-21-732.

[6] Kva seier forskrifta? [Internettside]. [Sist oppdatert: 19.04.2016, Sitert: 06.05.2016]. Tilgjengelig fra: https://uu.difi.no/krav-og-regelverk/kva-seier-forskrifta.

[7] WCAG 2.0-standarden. [Internettside]. [Sist oppdatert: 2013, Sitert: 06.05.16]. Tilgjengelig fra: https://uu.difi.no/krav-og-regelverk/wcag-20-standarden.

[8] PACT Analysis. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 17.05.2016]. Tilgjengelig fra: http://hci.ilikecake.ie/requirements/pact.htm.

[9] Lifetimes of cryptographic hash functions. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 16.05.2016]. Tilgjengelig fra: http://valerieaurora.org/hash.html.

[10] Cooper, A. 2007. *About Face - The Essantials of Interaction Design 3*. Wiley Publishing Inc.

[11] StatCounter Global Stats. Top 8 Mobile Operating Systems in Norway from May 2015 to Apr 2016 [Internettside]. [Sist oppdatert: 2016, Sitert: 23.04.2016]. Tilgjengelig fra: http://gs.statcounter.com/#mobile_os-NO-monthly-201505-201604-bar.

[12] Salesforce Developers. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options [Internettside]. [Sist oppdatert: 04.2015, Sitert: 03.05.2016]. Tilgjengelig fra: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options.

[13] Tech Crunch. Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5 [Internettside]. [Sist oppdatert: 11.09.2012, Sitert: 24.01.2016]. Tilgjengelig fra: http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5

[14] Agile software development methodologies and how to apply them. [Internettside]. [Sist oppdatert: 30.12.2013, Sitert: 13.05.2016]. Tilgjengelig fra: http://www.codeproject.com/Articles/604417/Agile-software-development-methodologies-and-how-t.

[15] Kanban vs Scrum - How to make the most of both. [Internettside]. [Sist oppdatert: 29.06.2009, Sitert: 13.05.2016]. Tilgjengelig fra: https://www.crisp.se/file-uploads/Kanban-vs-Scrum.pdf.

[16] ISTQB Exam Certification. What is Planning Poker? Effort estimation in Agile methodology [Internettside]. [Sist oppdatert: Ukjent , Sitert: 13.05.2016]. Tilgjengelig fra: http://istqbexamcertification.com/what-is-planning-poker-effort-estimation-in-agile-methodology/.

[17] AngularJs. Superheroic Javascript MVW Framework [Internettside]. [Sist oppdatert: Ukjent , Sitert: 13.05.2016]. Tilgjengelig fra: https://angularjs.org/.

[18] Syntactically Awesome Style Sheets. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 13.05.2016]. Tilgjengelig fra: http://sass-lang.com.

[19] Apache Cordova [Internettside]. [Sist oppdatert: Ukjent]. Tilgjengelig fra: https://cordova.apache.org.

[20] Ionic: Advanced HTML5 Hybrid Mobile App Framework. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 01.04.2016]. Tilgjengelig fra: http://ionicframework.com/.

[21] CAP Twelve Years Later: How the rules Have changed. [Internettside]. [Sist oppdatert: 30.05.2012, Sitert: 12.05.2016]. Tilgjengelig fra: http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed.

[22] Visual Guide to NoSQL Systems. [Internettside]. [Sist oppdatert: 15.03.2010, Sitert: 13.05.2016]. Tilgjengelig fra: http://blog.nahurst.com/visual-guide-to-nosql-systems.

[23] about Pouchdb. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 13.05.2016]. Tilgjengelig fra: https://pouchdb.com/learn.html.

[24] Node.js foundation. about Node.js [Internettside]. [Sist oppdatert: Ukjent , Sitert: 13.05.2016]. Tilgjengelig fra: https://nodejs.org/en/about/.

[25] API basics. [Internettside]. [Sist oppdatert: 06.02.2015, Sitert: 17.05.2016]. Tilgjengelig fra: https://github.com/apache/couchdb-documentation/blob/master/src/api/basics.rst.

[26] Per document access control. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 17.05.2016]. Tilgjengelig fra: https://ehealthafrica.github.io/couchdb-best-practices/#per-document-access-control.

[27] An introduction to Redis data types and abstractions. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 17.05.2016]. Tilgjengelig fra: http://redis.io/topics/data-types-intro.

[28] Moment.js. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 13.05.2016]. Tilgjengelig fra: http://momentjs.com/.

[29] MVC Architecture. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 15.04.2016]. Tilgjengelig fra: https://developer.chrome.com/apps/app_frameworks.

[30] Java Fundamentals Tutorial. Design Patterns - NewCircle [Internettside]. [Sist oppdatert: Ukjent , Sitert: 17.05.2016]. Tilgjengelig fra: https://newcircle.com/bookshelf/java_fundamentals_tutorial/design_patterns.

[31] Angular Style Guide. [Internettside]. [Sist oppdatert: 27.04.2016, Sitert: 21.04.2016]. Tilgjengelig fra: https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md.

[32] Apple Inc.. iOS Human Interface Guidelines: Bars (Tab Bar) [Internettside]. [Sist oppdatert: 21.03.2016, Sitert: 16.04.2016]. Tilgjengelig fra: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW52.

[33] Tabs - Components - Google design guidelines. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 03.05.2016]. Tilgjengelig fra: https://www.google.com/design/spec/components/tabs.html.

[34] 3.3.3 Forslag ved feil (Nivå AA). [Internettside]. [Sist oppdatert: 01.09.2015, Sitert: 17.05.2016]. Tilgjengelig fra: https://uu.difi.no/artikkel/2015/07/333-forslag-ved-feil-niva-aa.

[35] Pivotal Labs. Introduction.js [Internettside]. [Sist oppdatert: Ukjent , Sitert: 17.05.2016]. Tilgjengelig fra: http://jasmine.github.io/2.0/introduction.html.

[36] Additional tools for testing Angular applications. [Internettside]. [Sist oppdatert: 2016, Sitert: 14.05.2016]. Tilgjengelig fra: https://docs.angularjs.org/guide/unit-testing#additional-tools-for-testing-angular-applications.

[37] Usability 101: Introduction to Usability. [Internettside]. [Sist oppdatert: 04.01.2012, Sitert: 15.04.2016]. Tilgjengelig fra: https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

[38] Use JSDoc. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 16.02.2016]. Tilgjengelig fra: http://usejsdoc.org/.

[39] Password length complexity. [Internettside]. [Sist oppdatert: 15.09.2013, Sitert: 17.05.2016]. Tilgjengelig fra: https://www.owasp.org/index.php?title=Password_length_%26_complexity&oldid=208871.

[40] JSON Web Token (JWT). [Internettside]. [Sist oppdatert: 05.2015, Sitert: 16.05.2016]. Tilgjengelig fra: https://tools.ietf.org/html/rfc7519.

[41] Expiration and Extension of Access Tokens. [Internettside]. [Sist oppdatert: Ukjent , Sitert: 16.05.2016]. Tilgjengelig fra: https://developers.facebook.com/docs/facebook-login/access-tokens/expiration-and-extension.

[42] Node.js Security Checklist. [Internettside]. [Sist oppdatert: 13.10.2015, Sitert: 24.04.2016]. Tilgjengelig fra: https://blog.risingstack.com/node-js-security-checklist/.

[43] Node Security Project [Internettside]. [Sist oppdatert: 2016, Sitert: 16.05.2016]. Tilgjengelig fra: https://github.com/nodesecurity/nsp.

[44] The Open Web Application Security Project (OWASP). Blocking Brute Force Attacks [Internettside]. [Sist oppdatert: 04.03.2016, Sitert: 17.05.2016]. Tilgjengelig fra: https://www.owasp.org/index.php?title=Blocking_Brute_Force_Attacks&oldid=210449.

A Backlog

ID	Navn	Beskrivelse
1	Hovedvindu	
1.1	Vise totalbeløp	Viser resterende beløp som er igjen pr måned. Hentes fra database.
1.2	Grafisk fremstilling	Grafisk fremstilling av totalbeløp i forhold til gjenstående.
1.3	GUI	Relevante ikoner og plassering av dem
1.4	Lenke til nytt innskudd	En knapp som sender brukeren videre til nytt innskudd
1.5	Lenke til ny utgift	En knapp som sender brukeren videre til ny utgift
2	Theme	Theme gjelder på alle vinduer
2.1	Farge	Finne riktiger farger i forhold til Designhåndbok fra Totens Sparebank
2.2	Topbar	Inneholder knapp til kundeservice og logo
2.3	Toolbar	Definere ikoner
2.4	Splash Screen	Lage splash screen som går i sleep mens applikasjonen lastet i bakgrunnen.
3	Nytt innskudd	
3.1	Fast innskudd	Legge til fast innskudd, returnere til oversikt over faste inn\ut. Kommunisere med database.
3.2	Enkelt innskudd	Legge til enkelt innskudd, returnere til main. animert totalbeløp i main som endre (tilbakemelding til bruker)
3.3	GUI	Likt for fast \ enkelt.
3.4	oppdatere budsjett	oppdater budsjettkategori
4	Ny utgift	
4.1	Fast Utgift	Legge til fast utgift, returnere til oversikt over faste inn\ut. Lagre i database
4.2	Enkel utgift	Legge til enkel utgift, returnere til main. animert totalbeløp i main som endre (tilbakemelding til bruker). Lagre i db
4.3	GUI	Likt for fast \ enkelt.
5	Legg til sparemål	
5.1	Fastsatt fremtidig mål	Sette et fremtidig mål med dato(Kalender , timepicker) og beløp
5.3	GUI	
5.4	Beregne sparemål	totalbeløp delt på tid + vise dette
5.5	Se Sparemål	Se alle sparemål i liste
5.6	Endre sparemål	Endre et sparemål
6.1	Kalkulator	Legge inn ønsket sparebeløp
6.2	Liste	Scroll nedover listen for å vise antall dager x sparebeløp
6.3	GUI	Legge inn sparebløp, velge tidshorisont, beregnes automatisk. Egen knapp for å legge til
7	Topbar Menu	Øverst i høye hjørne
7.1	Faste utgifter og inntekter	Lenker til oversikt over faste oppdrag
7.2	Transaskjonshistorikk	Lenker til en oversikt over alle transaksjoner, Disse skal kunne sorteres etter dato\pris. 3 visninger: utgift, innskudd, al
7.3	Kontakt oss	Telefon, Nettside og kart
7.3.1	Kart	Kart fra Google Maps med oversikt over alle kontorer, addresse og åpningstider
8	Standardtall fra SIFO	
8.1	Hente ut standardtall	Koble opp mot nettsiden, laste ned xls fil og konvertere til leselige verdier.
8.2	Kategorisere standardtall	Sortere verdier etter kjønn og alder
9	Database	
9.1	Selve databasen	Sette opp struktur for databasen og dens tabeller
10	Faste utgifter og inntekter	Vise oversikt over faste utgifter inntekter. Lenke til ny intekt\utgift
10.1	Slette	
10.2	Endre	
10.3	Vise liste	Viste liste over faste utgifter \ inntekter
10.4	Bytte mellom U/I	Bytte mellom utgift og inntekt
11	Transaskjonshistorikk	Vise liste med utgifter og innskudd.
11.1	Slette	
11.2	Endre	
11.3	Vise liste	Vise liste over transaksjoner. Filter skal også inkluderes.
11.4	Bytte mellom U/I	Bytte mellom utgift og inntekt
12	Back-End/Verktøyskonfigurasjon	
12.1	Sette opp web-server	
12.2	Installere nødvendige pakker	node.js, npm, cordova, ionic, pouchdb, couchdb, Apache2
12.3	Konfigurere web-server	Gjøre nødvendige konfigurasjoner for installerte pakker
12.4	Opprette git-repo	
13	Intro	Setup som gjennomgås første gang brukeren tar i bruk applikasjonen
13.1	Informasjonsside	
13.2	Ferdigkonfigurert forslag	
14	Budsjett	

14.1	Legge til ny budsjettpost	Legge inn navn, velge ikon og velge et beløp for posten
14.2	Slette en budsjettpost	Slette
14.3	Endre en budsjettpost	Endre navn, ikon, beløp
15	Login	
15.1	Login form	
15.2	Verifisering mot database	
15.3	Sesjons cookie	
15.4	Sync mot DB	
16	Registrere Bruker	
16.1	Login med facebook	
16.2	Registrere manuelt	Navn, epost, profilbilde, passord

B Prosjektplan

Budsjettapplikasjon

Prosjektplan

Kristian Dragerengen
Per-Kristian Nilsen
Jardar Tøn
Per Arne Drevland

26.01.2015
NTNU Gjøvik

Innholdsfortegnelse

1. Introduksjon

Denne bacheloroppgaven dreier seg om utvikling av en mobilapplikasjon for budsjettering av personlig økonomi. Dette gjelder budsjett for enkeltmann, familie og mindre organisasjoner.
Oppdragsgiver er Totens Sparebank.

Denne prosjektplanen vil inneholde kartlegging av gjennomføringen av hele prosjektperioden. Ansvarsfordelinger, utviklingsmetodikk og løsningsdiskusjon er blant emnene som vil dekkes.

2. Oppgavebeskrivelse

Oppgaven baserer seg på å hjelpe brukere til å kontrollere sin økonomi, uavhengig av om de er kunde i banken eller ikke, ved hjelp av en applikasjon til mobile enheter. Oppdragsgiver ønsker en slik applikasjon da ønsket funksjonalitet, behov og innhold ikke finnes per dags dato. Oppdragsgiver ønsker at applikasjonen er tilgjengelig på flere plattformer. Dette er for å gjøre seg selv mer attraktiv for en yngre målgruppe. Applikasjonen kan brukes til å sette opp både enkle og mer avanserte budsjetter. Kjøp og inntekter blir lagt til av brukeren fortløpende og kan sammenlignes med det oppsatte budsjettet. Kombinert med at brukeren kan sette seg sparemål, vil gjøre at brukeren enkelt kan holde kontroll over sin økonomi og følge med på hvor mye som spares og brukes til ulike formål. Disse sparemålene vil bli fordelt utover perioden fram mot målet slik at brukeren opplever et stort sparemål som oppnåelig siden det ikke krever mye sparing hver dag. Brukeren skal også kunne se oversikt over alle sine utgifter i en utgifts-historikk å se hvordan dette er i forhold til det oppsatte budsjettet. Oversikt over faste utgifter og inntekter skal også vises. En graf som viser oversikt over sparingen vil gi brukeren en statistikk på hvordan sparingen har gått bakover i tid. Oppdragsgiver ønsker at applikasjonen skal utformes på en slik måte at det vil være lett å bruke og enkel å forstå, for å bidra til at brukere finner motivasjon til å holde oversikt over sin økonomi.

Applikasjonen skal også inkludere funksjonalitet utover enkeltmannsbruk, slik at familier, grupper eller mindre organisasjoner kan føre et budsjett sammen. Dette skal fungere på en slik måte at all endring av en budsjettpost skal synkroniseres mellom de aktuelle enhetene. Det skal også tas hensyn til at det blir gjort endringer på en enhet som midlertidig er ute av stand for synkronisering, slik at eventuelle endringer blir foretatt i det enheten har mulighet for synkronisering igjen. Til det må det utvikles mekanismer som ligger i bakgrunnen som lytter etter endringer, lagrer brukerdata, og synkroniserer enhetene både lokalt og sentralt. For å kunne ta i bruk sentral lagring og aksessering av felles informasjon, kreves registrering av brukere.

Applikasjonen skal også kunne kjenne igjen brukere, slik at autentisering ikke trenger å bli gjort hver gang applikasjonen er i bruk. Det må også utvikles sikkerhetsmekanismer slik at brukerinformasjon ikke kommer på avveie.

Applikasjonen skal også gi brukeren rask mulighet til å nå kundeservice. Herunder hjemmesiden, kundeservice på telefon og oversikt over kontorer i et kart.

Oppgaven splittes opp i en prioritert liste etter ønske fra oppdragsgiver:

1. Bruk for enkeltpersoner.
2. Bruk for familie; kommunikasjon mellom brukere.
3. Bruk for mindre organisasjoner.

Oppgavene gjennomføres etter rekkefølge nevnt ovenfor. Dette legger til rette for at sluttproduktet tilfredsstiller oppdragsgiver.

3. Tidligere arbeid

Et utviklingsprosjekt for deler av denne oppgaven er allerede blitt utført i faget IMT3672 Mobile Development Project hvor en uferdig prototype ble utviklet til bruk for Android plattformen. Det ble parallelt utført et systemutviklingsarbeid for denne prototypen i faget IMT3102 Objektorientert systemutvikling. Tidligere utført arbeid gir en god pekepinn på hvordan denne oppgaves skal løses. Intet materiell fra prototypen kan gjenbrukes bortsett fra selve tankegangen og ideene.

4. Krav

4.1. Funksjonelle krav

- Tilby forslag til forhåndsdefinert budsjett via SIFO referansebudsjett.
- Legge til, fjerne og endre budsjettposter.
- Legge til, endre og fjerne utgifter og inntekter i budsjettet.
- Legge til, endre og fjerne sparemål.
- Få oversikt over budsjett med transaksjoner i de forskjellige postene.
- Tilgang til bankes kundeservice og hjemmeside.
- Dele budsjett med andre brukere. Eks. Fellesbudsjett i en familie.
- Ha budsjettet tilgjengelig på flere enheter via en brukerkonto.

4.2. Krav for utforming

- Utforming i henhold retningslinjer etter designhåndbok fra Totens Sparebank.
- Utforming med hensyn til retningslinjer for universell utforming.
- Applikasjonen skal være tilpasset en yngre aldersgruppe.

4.3. Utvidelsesmuligheter

- Optical character recognition (OCR). Kan brukes for å ta bilde av kvitteringer og gjenkjenne innholdet.
- Web grensesnitt.

5. Tidsfrister

- Prosjektplan leveres i Fronter innen 28. januar.
- Grupperegler signeres og leveres innen 28. januar.
- Prosjektavtale signeres og leveres innen 28. januar.
- Sluttrapport leveres Fronter innen 18. mai.

6. Problemløsning

Det største problemet innen denne oppgaven er å lage et produkt som kan brukes på forskjellige plattformer som iOS, Android og Windows. Dette kan i hovedsak løses på tre forskjellige måter;

- Løsning 1. Utvikle native for alle plattformer.
- Løsning 2. Webapplikasjon som fungerer likt på alle plattformer.
- Løsning 3: Hybrid, utvikle med samme kodebase for alle plattformer hvor deler av applikasjonen bruker et webgrensesnitt, og andre deler bruker native API.

6.1. Native, hybrid eller web

Webapplikasjoner har en fordel i at de ikke trenger å installeres på mobilenheten. Dette fungerer ved at brukeren besøker en nettside, lagrer et bokmerke, og lager en snarvei til bokmerket på hjemskjermen. Slike applikasjoner kjøres i nettleseren. Dette gjør at applikasjonens funksjonalitet vil være låst til hva nettlesere kan tilby. Samtidig vil ikke denne typen applikasjoner kunne legges tilgjengelig i App Store og

Google Play Store, noe som er en stor ulempe. Dette til sammen gjør at webapplikasjon trolig ikke vil være riktig retning å ta.

Når det gjelder native- kontra hybridapplikasjon, ville valget vært enklere for noen år siden. Selv om hybrid ville gitt en mye bedre kryssplattform-kapabilitet var ikke teknologien utviklet nok til å gitt en god nok brukeropplevelse. Facebook prøvde seg før de gikk over til native, og sa på «TechChrunch disrupt» i september 2012 “Vår største tabbe var å legge for mye lit til HTML5”[2]. De tok steget fra en webapplikasjon som var kryss-plattform til native, som var en hel del merarbeid for Facebook, men de fikk det som de ville når det gjaldt brukeropplevelse og ytelse.

Det er lenge siden 2012 og teknologien har blitt langt bedre enn den gang. HTML5 gir tilgang til sensorer, og applikasjonens størrelse må tas hensyn til når det gjelder mulighetene for kryssplattform. Alle på gruppen har erfaring med native utvikling for Android etter emnene Mobile Development Theory og Mobile Development Project. Denne erfaringen vil kunne spare gruppen for mye tid dersom løsningen ender opp som en native applikasjon for Android. Derimot har ingen av oss erfaring med utvikling for iOS. Dette betyr at en del tid ville gått til å sette seg inn i programmeringsspråket Swift og tilhørende rammeverk. Det samme gjelder for Windows. I tillegg vil selve utviklingen av applikasjonen for alle operativsystemene ta veldig lang tid. Allikevel har gruppen som mål å oppfylle så mange av oppdragsgiverens ønsker som mulig. Dermed blir hybridapplikasjon det beste alternativet for dette prosjektet.

6.2. Valg av rammeverk

Valg av rammeverk og løsning har blitt diskutert internt i prosjektgruppen. Mulighetene ved utvikling av en hybrid applikasjon har blitt utforsket på forhånd av prosjektperioden. Å utvikle applikasjonen på en slik måte åpner for å enkelt kunne tilpasse det grafiske brukergrensesnittet mot både Android, iOS og Windows Phone, og å bruke samme kodebase for alle operativsystemene. Det vil også åpne for å utvikle en nettløsning, ment for å brukes på PC, for å tilrettelegge for mer avansert funksjonalitet. Særlig funksjonalitet for mindre organisasjoner er relevant i en slik løsning. For å oppnå dette vil gruppen måtte oppfriske kunnskapene innen webutvikling, ettersom hybrid utvikling baserer seg på teknologier som HTML, CSS og JavaScript. Dette vil ta en del tid å komme i gang med, spesielt med tanke på rammeverkene rundt JavaScript.

Prototypen som ble laget av applikasjonen i høstsemesteret 2015 ble utviklet med fokus på design. Dette førte til at applikasjonen hadde et grafisk brukergrensesnitt som var sterkt preget av Totens Sparebanks profil, samtidig som den beholdt en “Android look-and-feel”. Dette designkonseptet er ønskelig å videreføre i dette prosjektet. Vi har sett på et rammeverk kalt Ionic, som er bygget på AngularJS. Ionic

forenkler prosessen av å utvikle grafiske brukergrensesnitt for hybride mobilapplikasjoner. Rammeverket inneholder CSS og JavaScript komponenter som kan brukes til å utvikle et brukergrensesnitt tilpasset mobile plattformer.

7. Begrensninger

Ionic bygger på cordova-rammeverket som ikke er støttet av eldre versjoner av Android, iPhone og Windows phone[3].

- For iOS støttes versjon 6 og oppover. Dette tilsvarer en markedsandel på 98,8% av alle iOS enheter som brukes i dag [5]
- For Android støttes ikke versjoner er eldre enn API level 13. Disse innehar en markedsandel som på verdensbasis utgjør <4% [4]
- Det er bare brukere med Windows Phone 10 som vil være aktuelle brukere for Windows plattformen.

8. Omfang

8.1. Fagområder

Økonomi og budsjettering vil være viktig å sette seg inn i for samtlige i prosjektgruppen. Budsjett for mindre organisasjoner kan innebære mange detaljer gruppen ikke har kjennskap til, blant annet diverse regler for skatt.

Ettersom applikasjonen skal utvikles som en hybrid mobilapplikasjon, blir teknologier for webutvikling sentralt. HTML, CSS og JavaScript har gruppen som nevnt over erfaring med fra et tidligere emne (WWW-Teknologi), mens rammeverkene som skal tas i bruk er nye. Dette gjelder spesielt rammeverk for JavaScript.

Prosjektgruppen ønsker erfaring med avanserte verktøy som blir brukt av profesjonelle utviklere på arbeidsplasser, og har valgt å bruke slike verktøy for å automatisere byggeprosesser, statisk kodesjekking og lignende. Herunder kommer også avansert bruk av git. Det vil si benytte profesjonelle metoder for utgreninger og sammenfletting til hovedgrenen.

Oppdragsgiver er spesielt opptatt av omdømme. Da er det spesielt viktig at alle produktene deres er pålitelige og til å stole på. Det blir derfor viktig for oppgaven at produktet ikke inneholder sårbarheter eller programmatiske hull. Applikasjonen vil inneholde lite sensitiv informasjon, men enhver lekkasje av noe som helst

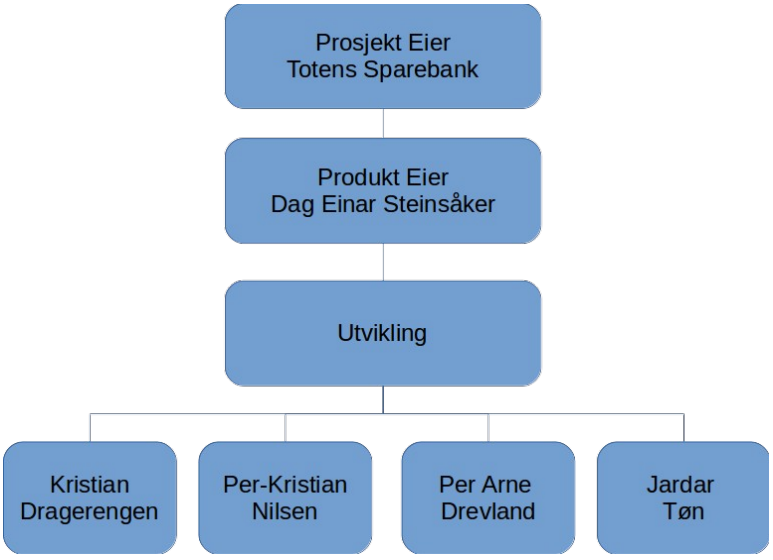
informasjon om brukere, vil oppleves som et tillitsbrudd. Ikke bare for applikasjonen, men også banken. Det blir da viktig for prosjektet å utelate enhver svakhet som kan skade oppdragsgivers omdømme.

Siden det skal lages en applikasjon som baserer seg på JavaScript, planlegges det med å bruke en NoSQL database server. Siden gruppen er mest komfortabel med SQL-databaser slik som MySQL, må det brukes en del tid for å sette seg inn de nye databasesystemene.

OCR - tekstgjenkjenning via kamera for å lese av kvittering slik at utgifter kan legges til uten at brukeren trenger å taste inn en sum.

For å oppnå et intuitivt brukergrensesnitt, vil prosjektet vektlegge design og brukervennlighet. Her må systematiske brukertester inngå som en vesentlig del av prosjektet.

9. Prosjektorganisering



Figur 1 - Organisasjon

9.1. Ansvar og roller

Prosjektleder: Kristian Dragerengen
- Ansvarlig for kontroll og koordinering av gruppen. Oppfølging: ajourføring av timer, vedlikehold av timeplan.

Assisterende prosjektleder: Per Arne Drevland
- Tiltrer ved prosjektleders fravær.

Ansvarlig for kommunikasjon mot oppdragsgiver: Kristian Dragerengen.
- Avtale og holde møtet.

Ansvarlig for kommunikasjon mot veileder: Per-Kristian Nilsen.
- Avtale og holde møtet.

Ansvarlig for statusrapporter og bloggposter: Per-Kristian Nilsen.
- Kontroll på at statusrapporter og bloggposter blir gjennomført Samt delegering av disse oppgavene.

Ansvar for å bestille grupperom er definert i kalenderen. Her er hvert medlem av gruppen ansvarlig for én uke per måned. Bestilling av grupperom skal skje en uke i forveien.

Veileder: Tom Røise.
Oppdragsgiver: Totens Sparebank.
Kontaktperson hos oppdragsgiver: Dag Einar Steinsåker.

9.2. Rutiner og regler

9.2.1. Faste møter

- Daily standup meeting.
- Retrospective meeting.
- Møte med veileder alle torsdager 14:30.
- Møte med oppdragsgiver hver tredje uke.

9.2.2. Loggføring og rapportering

- Hver deltaker av gruppen er selv ansvarlig for å loggføre antall timer og en beskrivelse av utført arbeid.
- Statusrapporter skal sendes to ganger i hver måned. Sendes til veileder og oppdragsgiver.
- Blogginnlegg på hjemmesiden skal publiseres i slutten av hver uke.

9.2.3. Grupperegler

1. Oppmøtetidpunkter i henhold til timeplan. Alle avvik fra timeplan skal meldes ifra til gruppen.
2. Alle påløpende kostnader føres i eget regneark. Oppgjør gjøres innad i gruppen i ettertid.
3. Uenigheter diskuteres i første omgang innad i gruppen. Det er alltid ønskelig at alle medlemmer av gruppen er enige før en beslutning tas. Hvis et medlem er uenig, bestemmer flertallet. Hvis det ikke er flertall kontakter gruppen veileder eller en annen fagperson for råd.
4. Et medlem av gruppen kan utestenges fra gruppen grunnet liten innsats eller lite oppmøte. Dette diskuteres innad i gruppen før eventuelt utestengelse. Gruppemedlemmet skal bli advart før utestengelse blir aktuelt. Veileder skal inkluderes i prosessen.
5. Alle gruppemedlemmer kan signere på vegne av gruppen så lenge det som skal signeres er kjent og akseptert for alle.

9.2.4. Grupperutiner

- Skype skal brukes til felles kommunikasjon når gruppen ikke er samlet.
- Ved kommunikasjon med oppdragsgiver pr. epost skal alle gruppedeltakere ligge som kopiadressat.
- Hver gruppedeltaker er ansvarlig for booking av grupperom etter arbeidstid i kalenderen.
    - Hvem som er ansvarlig for hver enkelt uke står i kalenderen.

10. Utviklingsmetodikk

Ved å ta i bruk en smidig utviklingsmetodikk vil hyppigheten på kommunikasjon mellom utviklingsgruppen, oppdragsgiver og veileder bli ivaretatt ved at kommunikasjon er en del av selve prosessen. En iterasjon kan inneholde utvikling av et enkelt modul hvor fremgangen direkte kan rapporteres rett til veileder og

oppdragsgiver. Dette sikrer kvaliteten av produktet ved å hele tiden ha en løpende kommunikasjon for å verifisere kvaliteten av produktet, samt at utviklingen er i tråd med oppgavebeskrivelsen.

Kanban brukes som grunnleggende utviklingsmodell. Dette er på grunnlag av dens fleksible og dynamiske metodikk som gir mulighet for tilpasninger underveis. Her er det ikke fastsatt store mengder med regler som må følges. Det er derimot åpent for tilpasning. Dette lar oss som liten gruppe kunne gjøre tilpasninger i henhold til utviklingsmetodikk som passer best for gruppen og prosjektet.

Det defineres en løsere “time-box” som milepæl. Disse milepælene blir definert før en utviklingsperiode starter og inneholder et sett med oppgaver som må være ferdigstilt før en milepæl ansees som nådd. Det er ønskelig å bruke rundt 10 arbeidsdager per milepæl, men dette er bare et mål og er veiledende. Alle mindre oppgaver ligger definert i produktets backlog.

Det er foretatt enkelte tilpasninger for å lettere fasilitere for prosjektet. For å kunne evaluere utviklingen underveis i prosjektet vil det være hensiktsmessig å avholde retrospektive møter. Møtet vil holdes etter at fastsatte milepæler er fullført, noe som medfører at estimering vil inkluderes i utviklingsmodellen. Dette for å lettere kunne velge oppgaver eller aktiviteter som ikke vil dra retrospektive møter for langt unna målet om å avholde de hver andre uke.

10.1. Backlog

Alle oppgaver ligger definert i en backlog. Backlog er listet opp med hovedkomponenter som tilsvarer milepæler som hver inneholder et sett med mindre oppgaver.

10.2. Kanban-board

Milepæls-oppgaver	Utvalgte oppgaver	Utvikling	Testing	QA	Ferdig
x WIP	5 WIP	3 WIP	3 WIP	3 WIP	x WIP

10.2.1. Milepælsoppgaver:

Oppgaver som er listet opp her er oppgaver som inngår i denne og neste milepæl. En milepæl er oppnådd når alle oppgavene for innehavende milepæl får status som ferdig. For at et gruppemedlem ikke skal bli “arbeidsledig” legges også neste milepælsoppgaver til slik at disse kan påbegynnes.

10.2.2. Utvalgte oppgaver:

Oppgaver som er listet opp her er oppgaver som har slektskap til hverandre. Det er for å kunne lettere ferdigstille funksjonaliteter som kan rulles ut i produksjon og være tilgjengelige så tidlig som mulig. Disse vil naturlig følge hverandre gjennom forskjellige statuser på kanbanboard-et.
Vi vil ha flere oppgaver her enn andre steder for å sørge for at ingen medlemmer skal gå uten å ha noe å gjøre. Når dette feltet er tomt vil det være naturlig å flytte flere oppgaver fra “Milepælsoppgaver” til “Utvalgte oppgaver”

10.2.3. Utvikling:

Her er alle oppgavene som er påbegynt og lagt til i git repo. Under utvikling kommer også enhets-testing. WIP settes til 3 for at en person skal kunne jobbe med andre oppgaver enn utvikling.

10.2.4. Testing:

Her er alle oppgavene som skal testes opp mot de modulene som de utviklede oppgavene skal jobbe med. Det kan være naturlig at de som har utviklet, også tester ferdig kode. Før en oppgave tolkes som ferdig i testing skal koden inspiseres og tilfredsstille fastsatte kodestandarder (se punkt x) og fastsatte statisk kodeinspeksjonskrav (se punkt x). I denne kolonnen kan det ligge like mange poster som ligger i “Utvikling” fordi det kan være at man blir ferdig samtidig med oppgaver som ligger i “Utvikling”

10.2.5. QA:

Oppgaver som ligger i “Quality assurance” skal gjennomgås av en annen enn den som har skrevet koden. Da skal koden ha gjennomgått 1. linjes koderevidering, all kode skal være dokumentert og enhetstester skal være skrevet. Her skal det forsikres at koden er lesbar og forståelig. Her er det mulighet for å ha 3 WIP. Dette kan bli en potensiell flaskehals, men ved å ha minst en person som ikke driver med utvikling kan en del oppgaver som ligger her bli unnagjort.

10.2.6. Ferdig:

Oppgaver her er oppgaver som er klar til å rulles ut til produksjon. Disse oppgavene er blitt en del av hovedgrenen i gir repo.

Dette oppsettet vil være gjeldende til og med første milepæl. Da vil det holdes et retrospektivt møte hvor bl.a. neste milepælsoppgaver defineres, revidering av kanbanboard, ny estimering av oppgavene og evaluering av milepæl vil diskuteres.

For at gruppen skal være oppdatert på hvordan det ligger an i forhold til oppnåelse av milepæl vil det være hensiktsmessig å avholde daglige stand-up møter. På disse møtene vil hver enkelt fortelle om hva som ble gjort dagen før, og det vil som gruppe bli valgt hvilke oppgaver eller aktiviteter som skal jobbes med denne dagen.

Det vil også bli delt inn ansvarsområder for prosjektet. Med dette menes at noen får ansvar for eksempelvis database. Det vil ikke si at denne personen skal gjøre alle oppgavene som har med database å gjøre, men denne personen har oversikt og definerer arbeidsoppgavene som har med det ansvarsområdet å gjøre.

10.3. Faglige ansvarsområder:

Kanban-board:	Per-Kristian
Database:	Per Arne
Budsjett og økonomifaglig:	Kristian
Nettside:	Per-Kristian
Brukergrensesnitt:	Kristian
Systemarkitekt:	Jardar
Automatisering:	Jardar
Testing:	Per-Kristian
Back-End:	Per Arne

10.4. Kanban Justeringer

Andre smidige utviklingsmetoder som XP, RUP eller Scrum er utelukket på grunn av dets større sett med regelverk og rutiner som ikke passer til et slikt utviklingsprosjekt på grunn av prosjektet og gruppens størrelse. Enkelte elementer fra disse modellen kan likevel benyttes.

Continuous integration fra eXtreme Programming skal brukes for å forhindre problemer med sammenfletting av forgreininger i koden. Automatiserte tester og

verifisering av koden vil skje før integrering. Dette vil forhindre at hovedgrenen ikke fungerer eller får en lavere standard.

Underveis i utviklingsprosessen vil det hyppig leveres versjoner av programvaren som både oppdragsgiver og besøkende til vår nettside fritt kan teste. Dette vil bli en variant av continuous delivery, selv om det ikke planlegges å lansere alle disse versjonene. Denne fremgangsmåten tilrettelegger for at oppdragsgiver kan komme med tilbakemelding på ny funksjonalitet så tidlig som mulig.

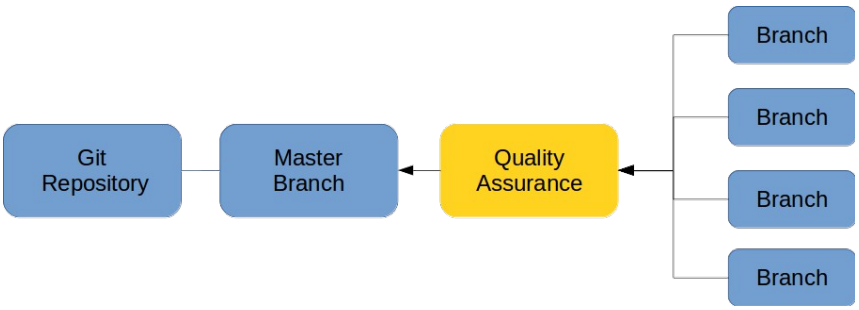
10.4.1. Estimering

Alle milepæloppgaver som ligger definert i produktets Backlog skal estimeres. Estimering utføres med planning poker slik at hvert gruppemedlem kan sette seg inn i hver enkelt deloppgave for så å gi et totalt estimat for en milepælsoppgave. Innholdet og omfanget i en deloppgave blir diskutert om estimatet er feiltolket av enkelte. Alle estimering gjøres i størrelsesorden av dagsverk.

11. Kvalitetssikring

11.1. Verktøy og retningslinjer:

Git benyttes for versjonskontroll.
Versjonskontroll skal fungere på en slik måte at all funksjonell og ferdig kode ligger på en hovedgren (Master Branch) som ligger på “Git Repository”. For hver funksjonalitet som påbegynnes, skal det lages en ny gren (Branch) ut av hovedgrenen hvor all koding skal foregå. Når en oppgave er fullført, skal den først kvalitetssikres av en annen før grenen kan tilknyttes hovedgrenen. På en slik måte kvalitetssikres koden som ligger i hovedgrenen.



Figur 2 – Versjonskontroll

- All kode skal kommenteres før det flettes inn i hovedgrenen.
- All kode skal verifiseres av et statisk kodeanalyseverktøy før det i flettes sammen med hovedgrenen.
- Unit tester skal skrives for all kode som forventes å gjennomgå hyppig forandring.

11.2. Automatiske sjekker:

For å kvalitetssikre kode som skal lastes opp i hovedgrenen i Git, vil det være fornuftig å revidere hverandres kode. Ideelt sett skal hovedgren alltid ha ferdig og gjennomført funksjonalitet med god kodekvalitet, dokumentasjon i form av kommentarer og nødvendige unit tester. På denne måten vil en slik gren kunne fungere som en presentabel versjon av programvaren.

For å sikre at disse kravene blir overholdt skal det være automatiske prosesser som sjekker disse opp mot koden når det behøves.

Når en forgrening skal flettes sammen med resten av koden vil det kjøres sjekker som ser om kodestilen følger bestemte standarder; at programmet bygges og at tester bestås. Hvis noe av dette ikke stemmer vil ikke sammenflettingen bli gjennomført før alt er ordnet. Et verktøy som analyserer koden og lager en rapport om kodekvalitet og testdekning skal også benyttes. Denne analysen kan brukes av andre som skal vurdere koden og vil gi en indikasjon på hvordan kodekvaliteten er totalt sett underveis i prosjektet. Programkoden og testene vil deretter vurderes av en annen på gruppen, som avgjør om arbeidet er som forventet.

11.3. Testing og feedback:

Applikasjonen vil gjøres tilgjengelig for oppdragsgiver underveis i utviklingsfasen. Dette vil gjøres ved at oppdragsgiver enten bruker IonicViewer (Tjeneste for å prøve applikasjoner laget i Ionic grensesnitt) for å kunne teste siste versjon av applikasjonen, eller at applikasjonen implementeres på gruppens nettside, slik at både oppdragsgiver og andre interessenter fritt kan prøve en interaktiv og oppdatert versjon. Denne versjonen kan enten komme direkte og automatisk fra hovedgrenen i Git, eller at gruppen selv legger den ut manuelt. Dette tilrettelegger for at oppdragsgiver kan komme med tilbakemeldinger så tidlig som mulig, i tillegg til at det gjør brukertesting lettere gjennomførbart.

12. Verktøy

- For utvikling i JavaScript og andre web-teknologier er WebStorm fra JetBrains valgt som IDE.
- Atlassian JIRA brukes for issue tracking og kanban board.
- Google Drive brukes for skriving av rapporter og andre dokumenter.
- ShareLaTeX brukes for ferdigstilling av rapporter.
- Diagrammer for bruk i rapporter lages i OmniGraffle.
- For prosjektstyring og dokumentasjon brukes Atlassian Confluence.
- Git: versjonskontroll mot online repository
- Bitbucket er valgt som Git-repository.

13. Risikoanalyse

Skjema er definert etter følgende regler:

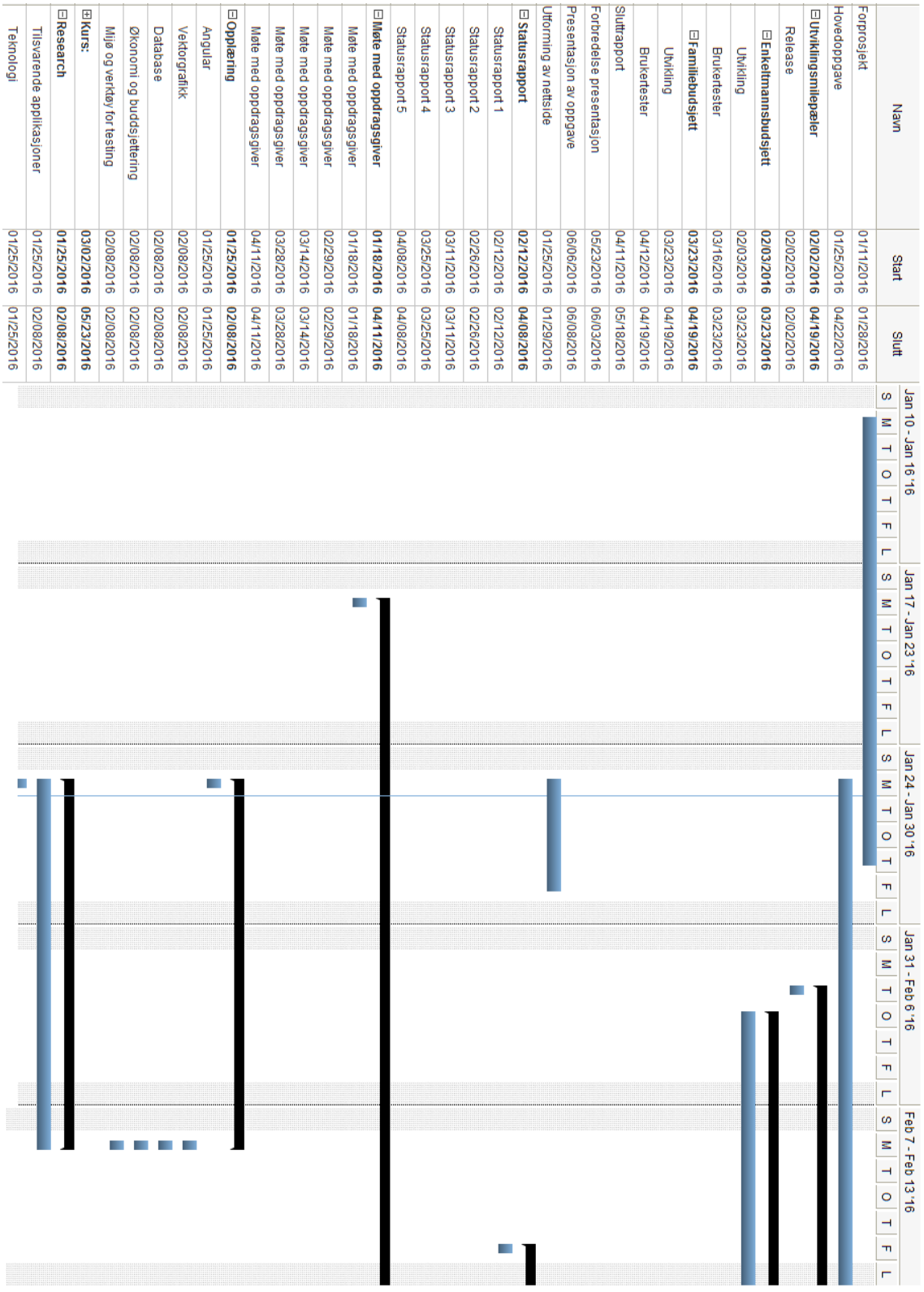
(1) Konsekvens: 5-Kritisk/svært alvorlig, 4-Alvorlig, 3-Moderat, 2-Lav/mindre, 1-Ubetydlig.
(2) Sannsynlighet: 5-Svært stor, 4-Stor, 3-Moderat, 2- Liten, 1-Meget liten.
(3) Risiko = Konsekvens * sannsynlighet.
(4) Restrisiko er risiko etter at tiltak er iverksatt.

Risiko 1 - 5: Risiko er akseptabel, men vurder likevel om risikoen kan reduseres ytterligere.
Risiko 6 - 12: Kan fortsettes. Vurder også andre alternativer.
Risiko 12 - 25: Oppgave bør ikke gjennomføres. Andre alternativer må iverksettes.

								Restrisiko (4)			
Nr:	**Hovedaktivitet:**	**Fare:**	**Årsak:**	**K: (1)**	**S: (2)**	**R: (3)**	**Tiltak:**	**K:**	**S:**	**R:**	**Merknad:**
1	Tidsramme sprekker	Oppdragsgiver får ikke forventet produkt ved prosjektslutt	Kompleksitet, underestimering	3	3	9	Justere oppdragsgivers forventninger, ha mange delleveringer.	1	3	3	Oppgaven deles opp i mindre biter slik at det mest essensielle utføres først.
2	Oppgavebeskrivelse endrer seg	Oppdragsgiver legger på for mye og for kompleks ønsket funksjonalitet	Oppdragsgiver ønsker endring av funksjonalitet	4	2	8	Gi klare forventningsavklaringer til oppdragsgiver over hva som er mulig og hva oppgaven innebærer	2	1	2	Ved å estimere hver oppgave vil det indikere hvor lang tid hver oppgave tar, og hvor mye tidsressurser det er til rådighet.
3	Versjonskontroll	Egne innstillingsfiler blir lastet opp	Brukerfeil	2	2	4	Konfigurere .gitignore på forhånd	2	1	2	
4	Uventet sykdom, utestengning eller andre årsaker gjør at gruppemedlemmer blir borte	Applikasjonssegment mister sin spesialist. Redusert arbeidskraft.	Forskjellig årsak	4	1	4	Grupperegler, ansvarsområder og oppgavefordeling. Dokumentasjon av kildekode.	2	1	2	
5	Ustabil levering av tjenester og ressurser.	Tjenester som Confluence og JIRA blir utilgjengelig.	Fusjon med NTNU	3	1	3	Backup	1	1	1	
6	Feiltolkning av teknologivalg i forhold til oppgave	Teknologivalg gir ikke mulighet til å løse oppgaven	Feil valg av teknologi	5	2	10	Sette seg inn i teknologien	5	1	5	

Basert på risikovurdering og iverksettelse av tiltak er det ingen prosesser som må stansen eller bli gjort på en annen måte. Alle definerte hovedaktiviteter gjennomføres i henhold til skjema med tiltakene som er definert. Alle tiltak er iverksatt.

14. Gjennomføring

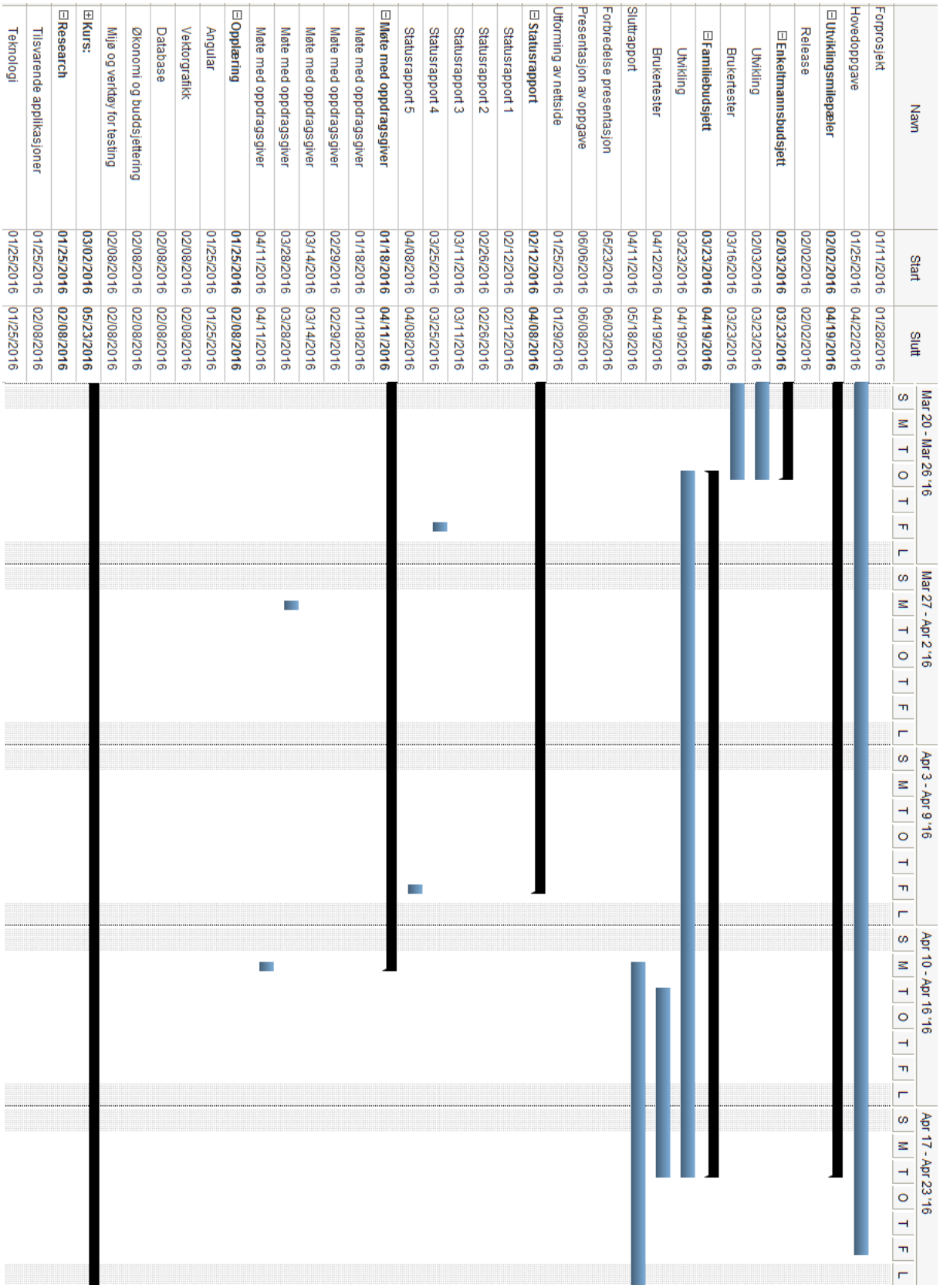


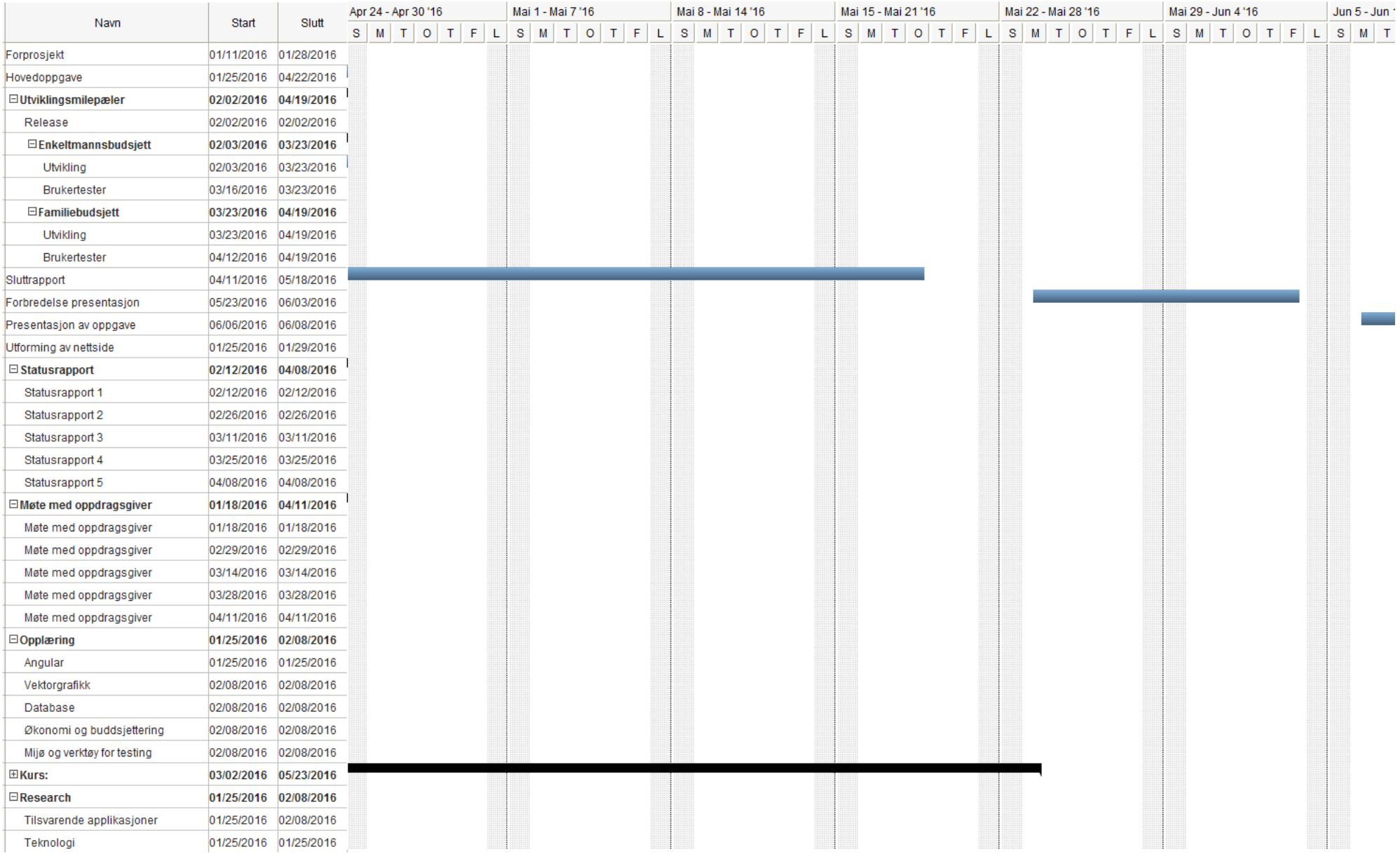
Navn	Start	Slutt
Forprosjekt	01/11/2016	01/28/2016
Hoveddoppgave	01/25/2016	04/22/2016
Utviklingsmilepæler	**02/02/2016**	**04/19/2016**
Release	02/02/2016	02/02/2016
Enkeltmannsbudsjett	**02/03/2016**	**03/23/2016**
Utvikling	02/03/2016	03/23/2016
Brukertester	03/16/2016	03/23/2016
Familiebudsjett	**03/23/2016**	**04/19/2016**
Utvikling	03/23/2016	04/19/2016
Brukertester	04/12/2016	04/19/2016
Sluttrapport	04/11/2016	05/18/2016
Forbredelse presentasjon	05/23/2016	06/03/2016
Presentasjon av oppgave	06/06/2016	06/08/2016
Utforming av nettside	01/25/2016	01/29/2016
Statusrapport	**02/12/2016**	**04/08/2016**
Statusrapport 1	02/12/2016	02/12/2016
Statusrapport 2	02/26/2016	02/26/2016
Statusrapport 3	03/11/2016	03/11/2016
Statusrapport 4	03/25/2016	03/25/2016
Statusrapport 5	04/08/2016	04/08/2016
Møte med oppdragsgiver	**01/18/2016**	**04/11/2016**
Møte med oppdragsgiver	01/18/2016	01/18/2016
Møte med oppdragsgiver	02/29/2016	02/29/2016
Møte med oppdragsgiver	03/14/2016	03/14/2016
Møte med oppdragsgiver	03/28/2016	03/28/2016
Møte med oppdragsgiver	04/11/2016	04/11/2016
Opplæring	**01/25/2016**	**02/08/2016**
Angular	01/25/2016	01/25/2016
Vektorgrafikk	02/08/2016	02/08/2016
Database	02/08/2016	02/08/2016
Økonomi og buddsjettering	02/08/2016	02/08/2016
Miljø og verktøy for testing	02/08/2016	02/08/2016
Kurs:	**03/02/2016**	**05/23/2016**
Research	**01/25/2016**	**02/08/2016**
Tilsvarende applikasjoner	01/25/2016	02/08/2016
Teknologi	01/25/2016	01/25/2016

Budsjettapplikasjon - Prosjektplan

Navn	Start	Slutt
Forprosjekt	01/11/2016	01/28/2016
Hovedoppgave	01/25/2016	04/22/2016
⊟Utviklingsmilepæler	**02/02/2016**	**04/19/2016**
Release	02/02/2016	02/02/2016
⊟Enkeltmannsbudsjett	**02/03/2016**	**03/23/2016**
Utvikling	02/03/2016	03/23/2016
Brukertester	03/16/2016	03/23/2016
⊟Familiebudsjett	**03/23/2016**	**04/19/2016**
Utvikling	03/23/2016	04/19/2016
Brukertester	04/12/2016	04/19/2016
Sluttrapport	04/11/2016	05/18/2016
Forbredelse presentasjon	05/23/2016	06/03/2016
Presentasjon av oppgave	06/06/2016	06/08/2016
Utforming av nettside	01/25/2016	01/29/2016
⊟Statusrapport	**02/12/2016**	**04/08/2016**
Statusrapport 1	02/12/2016	02/12/2016
Statusrapport 2	02/26/2016	02/26/2016
Statusrapport 3	03/11/2016	03/11/2016
Statusrapport 4	03/25/2016	03/25/2016
Statusrapport 5	04/08/2016	04/08/2016
⊟Møte med oppdragsgiver	**01/18/2016**	**04/11/2016**
Møte med oppdragsgiver	01/18/2016	01/18/2016
Møte med oppdragsgiver	02/29/2016	02/29/2016
Møte med oppdragsgiver	03/14/2016	03/14/2016
Møte med oppdragsgiver	03/28/2016	03/28/2016
Møte med oppdragsgiver	04/11/2016	04/11/2016
⊟Opplæring	**01/25/2016**	**02/08/2016**
Angular	01/25/2016	01/25/2016
Vektorgrafikk	02/08/2016	02/08/2016
Database	02/08/2016	02/08/2016
Økonomi og budsjettering	02/08/2016	02/08/2016
Miljø og verktøy for testing	02/08/2016	02/08/2016
⊞Kurs:	**03/02/2016**	**05/23/2016**
⊟Research	**01/25/2016**	**02/08/2016**
Tilsvarende applikasjoner	01/25/2016	02/08/2016
Teknologi	01/25/2016	01/25/2016

Budsjettapplikasjon - Prosjektplan




Navn
Start
Slutt
Apr 24 - Apr 30 '16
Mai 1 - Mai 7 '16
Mai 8 - Mai 14 '16
Mai 15 - Mai 21 '16
Mai 22 - Mai 28 '16
Mai 29 - Jun 4 '16
Jun 5 - Jun
S M T O T F L
Forprosjekt 01/11/2016 01/28/2016
Hovedoppgave 01/25/2016 04/22/2016
Utviklingsmilepæler 02/02/2016 04/19/2016
Release 02/02/2016 02/02/2016
Enkeltmannsbudsjett 02/03/2016 03/23/2016
Utvikling 02/03/2016 03/23/2016
Brukertester 03/16/2016 03/23/2016
Familiebudsjett 03/23/2016 04/19/2016
Utvikling 03/23/2016 04/19/2016
Brukertester 04/12/2016 04/19/2016
Sluttrapport 04/11/2016 05/18/2016
Forbredelse presentasjon 05/23/2016 06/03/2016
Presentasjon av oppgave 06/06/2016 06/08/2016
Utforming av nettside 01/25/2016 01/29/2016
Statusrapport 02/12/2016 04/08/2016
Statusrapport 1 02/12/2016 02/12/2016
Statusrapport 2 02/26/2016 02/26/2016
Statusrapport 3 03/11/2016 03/11/2016
Statusrapport 4 03/25/2016 03/25/2016
Statusrapport 5 04/08/2016 04/08/2016
Møte med oppdragsgiver 01/18/2016 04/11/2016
Møte med oppdragsgiver 01/18/2016 01/18/2016
Møte med oppdragsgiver 02/29/2016 02/29/2016
Møte med oppdragsgiver 03/14/2016 03/14/2016
Møte med oppdragsgiver 03/28/2016 03/28/2016
Møte med oppdragsgiver 04/11/2016 04/11/2016
Opplæring 01/25/2016 02/08/2016
Angular 01/25/2016 01/25/2016
Vektorgrafikk 02/08/2016 02/08/2016
Database 02/08/2016 02/08/2016
Økonomi og buddsjettering 02/08/2016 02/08/2016
Mijø og verktøy for testing 02/08/2016 02/08/2016
Kurs: 03/02/2016 05/23/2016
Research 01/25/2016 02/08/2016
Tilsvarende applikasjoner 01/25/2016 02/08/2016
Teknologi 01/25/2016 01/25/2016

15. Kilder

[1] Crips. Kanban vs Scrum [Internettside]. Kniberg, Henrik og Sakrin, Mattias [oppdatert 29.06.2009, sitert 12.01.2016] Tilgjengelig fra: https://www.crisp.se/file-uploads/Kanban-vs-Scrum.pdf

[2] Tech Crunch. Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5 [Internettside]. Olanoff, Drew [oppdatert 11.09.2012, sitert 19.01.2016] Tilgjengelig fra: http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/

[3] Ionic. Installing Ionic and its Dependencies [Internettside]. Ionic [sitert 19.01.2016] Tilgjengelig fra: http://ionicframework.com/docs/guide/installation.html

[4] Android Developers. Dashboards [Internettside]. Android Developers; [sitert 21.01.2016] Tilgjengelig fra: http://developer.android.com/about/dashboards/index.html

[5] David-Smit.com. iOS Version Stats [Internettside.] Smith, David; [oppdatert 21.01.2016, sitert 21.01.2016] Tilgjengelig fra: https://david-smith.org/iosversionstats/

16. Bilder og figurer

17. Vedlegg

C Grupperegler

Grupperegler for bacheloroppgave

1. Oppmøtetidpunkter i henhold til timeplan. Alle avvik fra timeplan skal meldes ifra til gruppen.
2. Alle påløpende kostnader føres i eget regneark. Oppgjør gjøres innad i gruppen i ettertid.
3. Uenigheter diskuteres i første omgang innad i gruppen. Det er alltid ønskelig at alle medlemmer av gruppen er enige før en beslutning tas. Hvis et medlem er uenig, bestemmer flertallet. Hvis det ikke er flertall kontakter gruppen veileder eller en annen fagperson for råd.
4. Et medlem av gruppen kan utestenges fra gruppen grunnet liten innsats eller lite oppmøte. Dette diskuteres innad i gruppen før eventuelt utestengelse. Gruppemedlemmet skal bli advart før utestengelse blir aktuelt. Veileder skal inkluderes i prosessen.
5. Alle gruppemedlemmer kan signere på vegne av gruppen så lenge det som skal signeres er kjent og akseptert for alle.

25.01.2016
NTNU Gjøvik

Kristian Dragerengen

Per-Kristian Nilsen

Per Arne Drevland

Jardar Tøn

D Prosjektavtale

NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

D. Steinsiker Totens Sparebank (oppdragsgiver), og

Kristian Draycrengen, Jardar Ten, Per Arne Dreuland og Per-Kristian Nilsen

(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.16 til 18.05.16.

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

   Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.

11. Når NTNU/AIMT også opptrer som oppdragsgiver, trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Tom Røise

Oppdragsgivers kontaktperson (navn): Dag Einar Steinsåker

Student(er) (signatur): _____ dato 25.01.16

_____ dato 25.01.16

_____ dato 25.01.16

Per-Kristian Nilsen dato 25.01.16

Oppdragsgiver (signatur): D. Steinsåker dato 25.01.2015

Signert avtale leveres digitalt i Fronter(IMT3912)
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): _____ dato _____

E Møtereferat

E.0.1 Retrospektivt møte

Retrospektivt møte 001

Dato:

25.02.2016

Tilstede:

- Jardar Tøn
- Per-Kristian Nilsen
- Kristian Dragerengen
- Per Arne Drevland

Agenda

Tema	Konklusjon
Oppnådd arbeid	God struktur på kode, men kunne vært bedre. Kommet bra i gang med unit-tester og jsdoc. Har grunnmuren for prosjektet på plass og har dermet er bra utgangspunkt. Utseende ser greit ut, men her er det mest å hente. Bra progresjon og produktet ser bra ut.
Arbeidsmetode	Det er nyttig å sitte på skolen for å kunne jobbe sammen, men det er også ønskelig å kunne variere med å sitte hjemme å jobbe. Utviklingsmodellen ved bruk av milepæler fungerer ikke like godt som planlagt, her blir det for mye små-oppgaver som tilkommer etterhver. Metode med Git og Jita fungere svært bra. Dette sammen med smart-commits gir en skikkelig god struktur. Navngivning av branch er forvirrende, de har bare tildelt et tall og en vanskelig å kjenne igjen. Burde også beskrives med ord. Sluttrapporten burde vært påbegynt. Vi må bli flinkere til å brukes eksterne ressurser, spørre andre etter hjelp og veiledning.
Fremdrift	Det har gått utrolig tregt hittil, da det er vanskelig med nye rammeverk og et tildels ukjent språk. Dette ble bedre som tiden gikk da vi etterhvert fikk en større kodebase for gjenbruk/inspirasjon. Løse rammer fra oppdragsgiver gjør oppgaver utfordrende, da vi må formulere dem selv.
Forbedringspotensiale	Gjenbruk av kode: Her skal det opprettes en service, slik at koden ligger et sted. Navngivning på branch er opp til den enkelte. Kommunikasjonen innad i gruppen er for dårlig, vanskelig å vite hvem som gjøre hva og hvem å spørre. Generell informasjon skal tas felles slik at alle får dette med seg. Viktige beslutninger og diskusjoner bør dokumenteres på confluence. Adminstrative timelogg må forbedres. NavnGivning kode kan bli bedre. Testing burde utføres oftere på mobile enheter. Mer definerte oppgaver og tidsfrister på milepeler: prioriter større ting.

Retrospektivt møte 002

Dato:

08.04.2016

Tilstede:

- Jardar Tøn
- Per-Kristian Nilsen
- Kristian Dragerengen
- Per Arne Drevland

Agenda

Tema	Konklusjon
Oppnådd arbeid	Det har vært effektiv jobbing med ny funksjonalitet uten og fokusere for mye på detaljer, som har vært et mål for denne milepælen. De aller fleste oppgavene som var planlag, er blitt ferdige. Det mangler en liten del på grunnfunksjonalitet for regelmessige transaksjoner.. For login er de viktigeste delene på plass, de må bare settes sammen. Vi har også gjort mer en forventet på server siden.
Arbeidsmetode	Kommunikasjon når noen sitter fast har blitt bedre. Å jobbe hjemme funger ikke, så dete slutter vi med. QA-prosessen frem til nå har vært for tidskrevende og det har blitt foretatt retting av unødvendige ting som har vært bortkastet tid. Vi har blitt flinkere til å kode, samt at elle har rammeverk, språk og kodestil mye mer i fingrene. Arbeidstid på skolen fungerer bra, men det er viktig å minne hverandre på å ta pause i ny og ne.
Frammdrift	Vi har fått bra fart på ting. Løsninger på oppgaver er mye mer åpenbare siden man er flinkere.
Forbedringspotensiale	Har blitt flinkere på gjenbruk av hverandres kode. Har ikke helt kontroll på hvem som gjør hva for QA på Jira: Dette løses med å ha oppgaver tildelt kun når en person gjør noe på oppgaven. Hvis ingen gjør noe, står oppgaven ledig. QA-prosessen omjusteres: mer prioritet på at applikasjonen fungerer riktig for brukeren og mindre prioritet av kodestil og kode som kunne gjort annerledes. Bruker forsatt ikke resurspersoner nok.

E.0.2 Møte med oppdragsgiver

Møte med oppdragsgiver 001

Dato

18 januar 2016

Tilstedeverende

- Jardar Tøn
- Per-Kristian Nilsen
- Kristian Dragerengen
- Per Arne Drevland
- Dag Einar Steinsåker
- Geir Sindre von Schantz Nyborg

Temaer

- Presentasjon av tidlegere arbeid (android app)
- Hybridapplikasjon
- Prosjektets retning
- Utvidelsesmuligheter
- Oppklaring bedriftsdel

Konklusjoner

Sak:	Konklusjon:
Presentasjon av allerede eksisterende prototype. Diskusjon rundt utvidelser og forbedringspotensiale.	• Bruker av farger og design var veldig bra. • Det var merkelig at applikasjonen inneholdt en oversikt over transaksjoner. • Ønske om OCR funksjonalitet. • Applikasjonen må inneholde funksjonalitet for å føre budsjett. Herunder å inkludere standardtall fra SIFO. • Navn på menyer. • Mer kategorier i faste utgifter • Mal for budsjett ut i fra hvilken rolle man innehar i en familie osv. • Mulighet for å kunn synke opp to enheter på samme budsjett
Gjennomgang av oppgave	Oppgaven skal deles opp i tre med prioritert rekkefølge: 1. Applikasjonen for enkeltmannbruk. 2. Kommunikasjon mellom brukere. Tilpasning for bedrift.
Diskusjon rundt hybrid applikasjon og native	Oppgaven tilsier at applikasjonen bør lages på iOS, Android og Windows. Dette medfører vesentlig mer arbeid om dette skal gjøre hvor hver enkelt plattform. Derfor vil det være mer hensiktsmessig å kunne lage en hybrid applikasjon slik at produktet kan lages en gang til alle plattformer.
Presentasjon av driftsløsning	Totens Sparebank er klar over at en egen server må driften for å kunne ha funksjonalitet som kommuniserer mellom enheter, samt egen nettside for bedrifter.
Oppfølging av arbeid	Møter med oppdragsgiver skjer omtrent hver tredje uke. Vi skal se på mulighetene for å gi banken tilgang til en versjon av appliksjonen via internett som vil bli oppdatert fortløpende.
Tidsfrister	Totens Sparebank hadde ikke behov for å sette noen tidsfrister. Tidsfrister i henhold til Bahceloroppgave og Gant-skjema skal følges.

Referent: Kristian Dragerengen

Møte med oppdragsgiver 002

Dato

25 februar 2016

Tilstedeverende

- Jardar Tøn
- Per-Kristian Nilsen
- Kristian Dragerengen
- Per Arne Drevland
- Dag Einar Steinsåker
- Geir Sindre von Schantz Nyborg

Temaer

- Presentasjon av tidlegere arbeid (android app)
- Hjem-skjerm
- Innlogging
- Stilsett
- Nedleggelse av kontorer

Konklusjoner

Sak:	Konklusjon:
Gjennomgang av produktet til nå. Ønskelig med kommentarer fra banken	• Budsjettposter blir røde ved overskridelse. • Formatering på tall: 2 500 000, 2 531, 250 000 • Cards: Utseende skal være tilsvarende budsjettpostene - Avstand mellom cards. • Ny Knapp: Knapp øverst i høyre hjørne • Lagre Knapp: bred med ikon på. Skal ikke strekkes ut over hele skjermen, men nesten. • Overskrifter: som vi allerede bruker
Diskusjon rundt hjem skjermen - Hva skal denne inneholde - Hvordan skal grafen se ut?	• Legg inn en forklaring på tallene på forsiden. • Skal være mulig å trykke på graf for å forandre utseende mellom % og kr
Innlogging med facebook og ikke eget system for dette	Bruk av ekstrerne + egne
Fonter og ikoner: Skal vi bruke deres eller valgfritt?	• Kutte ned på antallet ikoner som brukes inne i applikasjonen ved kategorier. • Egen font for brødtekst (Lucida Sans).
Hvilke kontorer skal legges ned. Slik at vi ikke legger inn disse i kartet.	Eina, Skreia og Feiring: Legges ikke inn på karte

Referent: Kristian Dragerengen

F Statusrapport

Statusrapport 001

Utvikling
Fremdriften går som planlagt. Vi har i skrivende stund fullført prosjektets første milepæl som betyr at mye grunnleggende funksjonalitet er på plass og applikasjonen kan brukes. Vi har brukt veldig mye tid og energi på å komme hit vi er nå. Dette er mest på grunn av at programmeringsspråk og rammeverk er helt nytt for oss. Det har vært en lang opplæringsprosess. Nå har alle mann skrevet sin egen kode, laget tester for dette, samt fått det kvalitetssikret. Ved å ha gjennomført alt dette står vi sterkere for å gå i gang med neste milepæl.

Ny Milepæl
- Mekanismer for innlogging.
    - Eget innloggingssystem.
    - Eksternt innloggingssystem (eks Facebook).
- Backend for innlogging.
- Intro for applikasjon: Sette opp inntekt \ vanlige poster.
- Implementering av SIFO tall.
- Legge inn inntekt.
- Få sparemål til å snakke med budsjettet.
- Generell retting av feil

Rapportskriving
Vi har enda ikke kommet i gang med å skrive rapport. Dette begynner å bli relativt kritisk, siden det er rapporten vi blir vurdert på og ikke selve produktet. Vi har tatt bilder og notert ned materiell til rapporten, men selve skrivingen må vi komme i gang med så fort som mulig.

Problemer
Det største problemer i forhold til selve produktet er tid. Tid kommer til å være en mangelvare uansett. Vi må videre prioritere om vi velger kvantitet eller kvalitet når det gjelder funksjonalitet.

Kommunikasjon med veileder
Vi har hatt løpende møter med veileder torsdag hver uke. Dette har vi fått mest ut av gjennom oppstartsfasen med prosjektplanen. Under starten av utviklingsfasen var ikke utbyttet veldig stort, da vi egentlig ikke hadde det store behovet for møtet. Men å kunne rapportere sin progresjon er alltid greit. På siste møte viste vi frem applikasjonen som den så ut, og fikk masse gode tilbakemeldinger. Kommunikasjonen med veileder kommer til å bli viktigere etter som produktet får mer funksjonalitet og ettersom vi begynner å skrive prosjektplan.

Kommunikasjon med oppdragsgiver
Vi har besøkt oppdragsgiver to ganger for å få svar på spørsål som har dukket opp under utviklingsfasen. Disse møtene kunne blitt byttet ut med kommunikasjon via epost eller liknende,

men det å kunne møte ansikt til ansikt gjør det mye letter å kommunisere rundt produktet og stille kritiske spørsmål. Tilbakemeldingene vi har fått har vært utrolig nyttige i forhold til videre utvikling.

Gruppe og arbeidsmetodikk
Vi samarbeider veldig bra og motivasjonen er på topp. Vi har hatt noe problemer med kommunikasjon innen gruppen som nå er tatt og og mest sannsynlig vil bli bedre. Dette handler mest om at vi jobber hver for oss, og hvis en enkelt person står fast med et problem, pleier vedkommende å prøve å løse det selv enn å høre med gruppen hvor en kanskje har løst dette fra før. Vi vil fortsette i samme tempo ved å jobbe i lag på skolen fra 8 - 16, mandag til torsdag. Når påsken kommer har vi blitt enige om å ta seg litt fri fra gruppearbeidet og jobbe hjemme. Vi ser nok ikke samme produktivitetsnivå i påsken, men er lurt å få en liten periode med noe annet også.

Etter avslutning av første milepæl ble det holdt et retrospektivt møte for blant annet å luke ut dårlige vaner og dyrke de gode. Vi fikk mye ut av dette møtet. Ytterligere detaljer blir ikke beskrevet her da det mest dreier seg om tekniske detaljer.

Kristian Dragerengen
Gruppeleder

26.02.2016
NTNU Gjøvik

G Justitia rapport

NORGE

CONSUMER PAYMENT REPORT 2015

Intrum Justitia har innhentet data fra 22 000 forbrukere i 21 europeiske land for å få innsikt i de europeiske forbrukernes hverdag; hvilke utgifter og hvilken evne de har til å håndtere egen månedlig husholdningsøkonomi. Fullstendig oversikt er presentert i European Consumer Payment Report 2015. For å få enda dypere innsikt har vi utviklet en landrapport som presenterer data fra denne rapporten.

intrum justitia

DE VIKTIGESTE FUNNENE

› 65 prosent legger av penger hver måned. Det gjør de først og fremst for å kunne håndtere uventede utgifter. 56 prosent kan håndtere en uventet regning på opp til 22 000 kroner. Andelen som sier at de kan håndtere denne type utgifter stiger med alderen.

› 46 prosent av nordmennene forbinder skilsmisse med økonomiske problemer. Dette er en høyere andel enn de som knytter økonomiske utfordringer med kostnader til bensin, drivstoff og elektrisitet.

› 31 prosent sier at de noen ganger har problemer med å få pengene til å strekke til. Blant respondentene i aldersgruppen 24 år eller yngre, svarer hele 41 prosent at dette er et problem. I samme aldersgruppe er 26 prosent enig i påstanden om at de ikke har penger nok til en anstendig tilværelse.

ØKONOMISK UTVIKLING	Norge	EU, Gjennomsnittlig
BNP per innbygger i Euro	73 400	27 300
BNP prosentvis vekst	2,2	1,3
Inflasjon	1,9	0,6
Arbeidsledighet	3,5	10,2

Kilde: European Payment Report

BETALINGSVANER (konsumenter til bedrifter)

Gjennomsnittlig betalingsbetingelser for konsumenter 18 dager

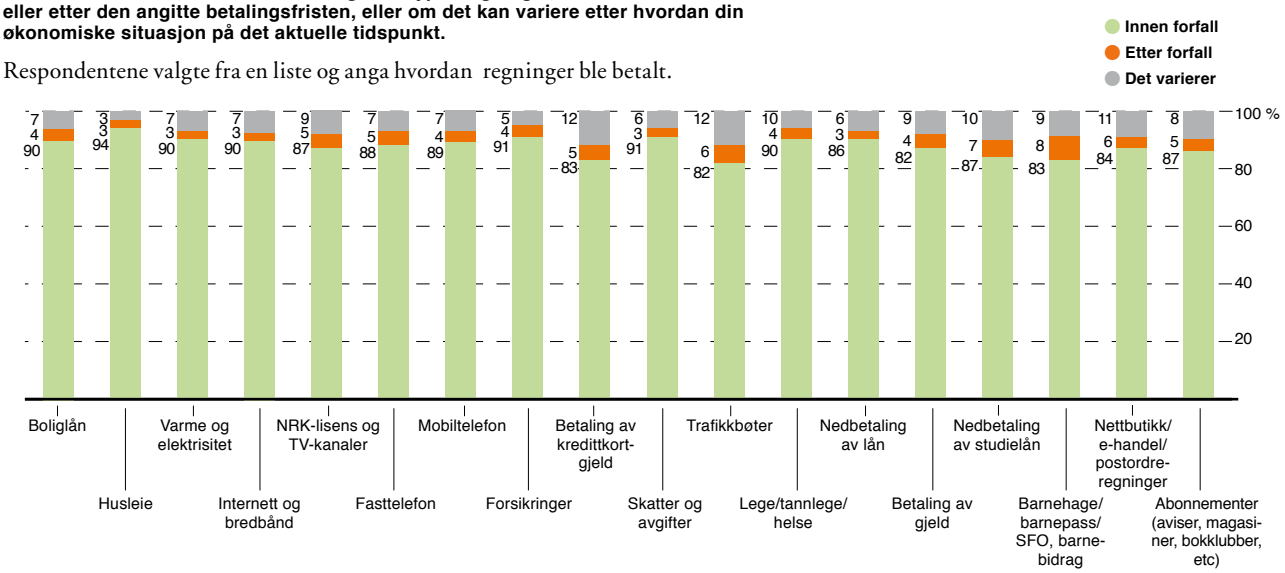
Gjennomsnittlig faktisk betalingstid 21 dager

Kilde: European Payment Report

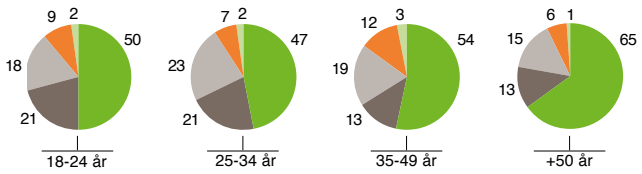
REGNINGER SOM ER BETALT FØR ELLER ETTER DEN ANGITTE BETALINGSFRIST (%)

Kan du fortelle oss, for hver av de følgende typer regninger, om du betaler disse innen eller etter den angitte betalingsfristen, eller om det kan variere etter hvordan din økonomiske situasjon på det aktuelle tidspunkt.

Respondentene valgte fra en liste og anga hvordan regninger ble betalt.

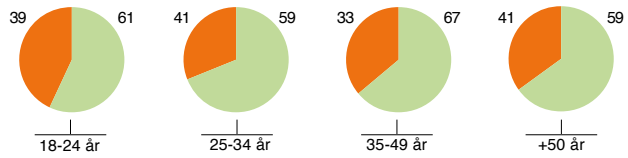


Antall regninger som ikke er betalt i tide i løpet av de siste 12 månedene (%)



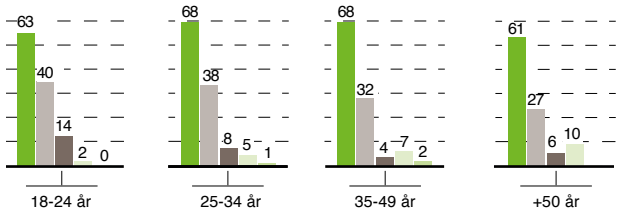
Nei · Ja, en regning · Ja, 2-4 regninger · Ja, fler enn 5 regninger · Vet ikke

Sparer du penger månedlig? (%)



Ja · Nei

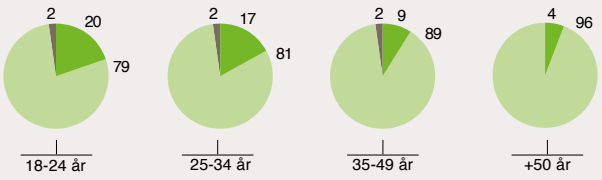
ÅRSAKER FOR IKKE Å BETALE EN REGNING I TIDE (%)



Uoppmerksom · Manglende evne · Vilje · Annet · Vet ikke

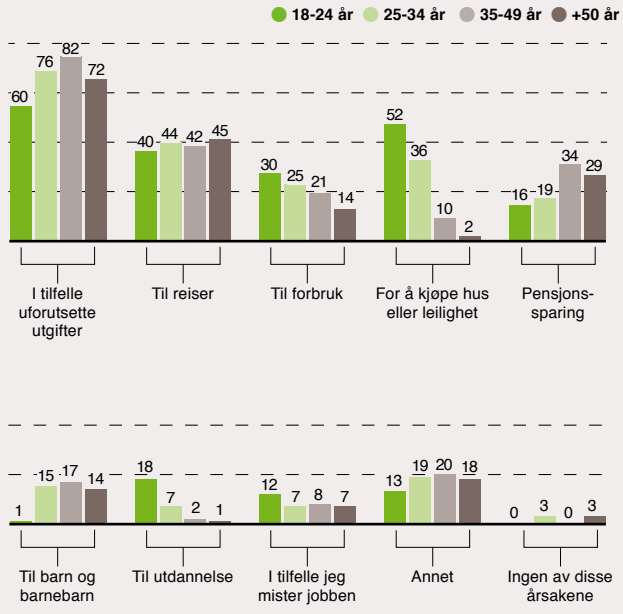
Så mye sparer nordmennene i gjennomsnitt per måned NOK 2 000

Bortsett fra boliglån, har du lånt penger de siste seks månedene for å betale regninger? (%)



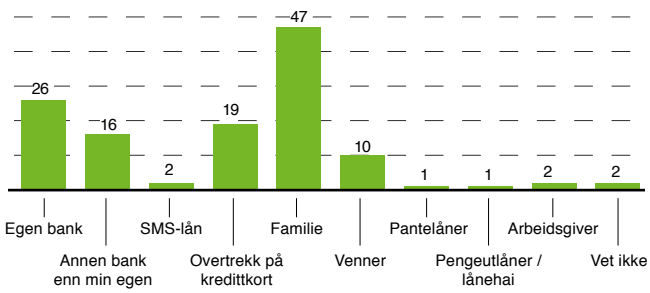
Ja · Nei · Vet ikke

Hvorfor sparer du penger månedlig? (%)



PENGER TIL REGNINGER BLE LÅNT FRA (%)

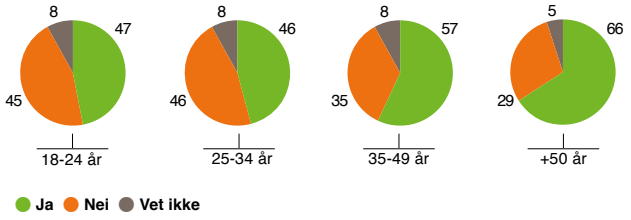
18-24 år · 25-34 år · 35-49 år · +50 år



EVNE TIL Å HA MIDLER TIL UFORUTSETTE UTGIFTER UTEN Å MÅTTE LÅNE (%)

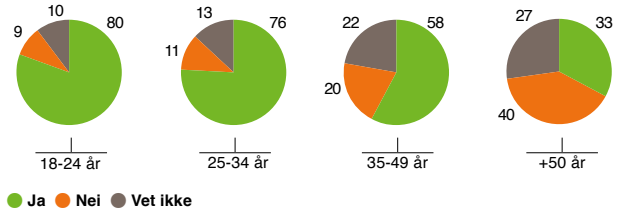
Har du råd til en uforutsett utgift på 26 000 NOK uten å måtte låne?

Beregnet til omtrent halvparten av en gjennomsnittlig månedslønn i Norge.

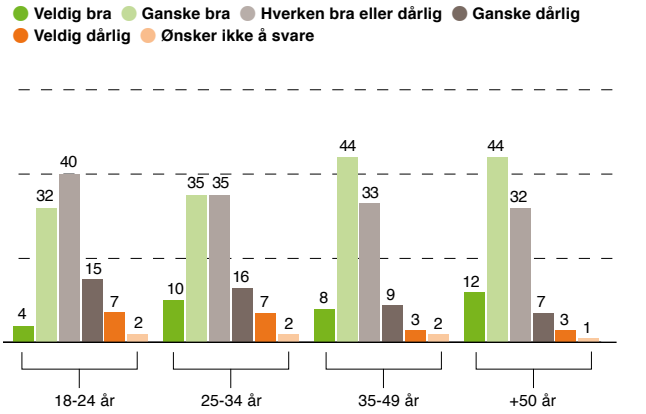


Tror du at du har en rimelig god sjanse til vesentlig å forbedre din økonomiske situasjon?

(%)

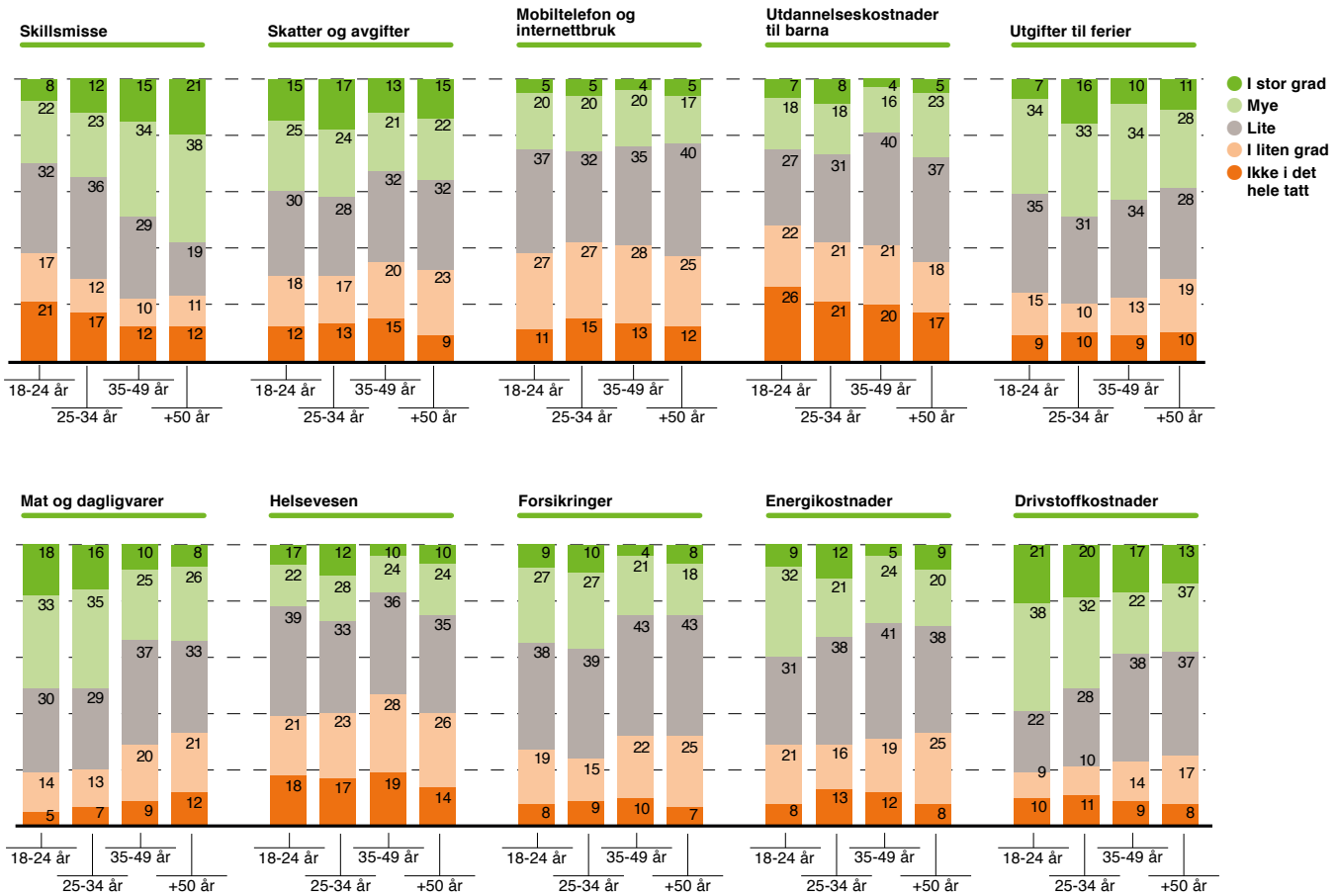


BESKRIVELSE AV MIN GENERELLE ØKONOMISKE SITUASJON (%)



MULIGE ÅRSAKER SOM BIDRAR TIL ØKONOMISKE PROBLEMER FOR NORDMENN (%)

Respondentene ble bedt om å si sin mening om mulige årsaker til nordmenns økonomiske problemer.



UTSAGN, GRAD AV ENIG ELLER UENIG (%)

1 18-24 år 2 25-34 år 3 35-49 år 4 +50 år

Enig Hverken enig eller uenig Uenig

10 TIPS FOR EN SUNN HUSHOLDNINGSØKONOMI

1. Lag et budsjett, slik at du har kontroll på hvor mye penger du kan bruke. Få grep om husholdningsutgiftene ved å liste opp alle utgiftene du har hver måned, slik at du ikke bruker mer penger enn du har til disposisjon.
2. Tenk deg om to ganger før du kjøper noe. Er dette noe du virkelig trenger?
3. Sett av penger til regninger du vet vil komme, slik at du har nok til å betale disse.
4. Ikke inngå økonomiske forpliktelser eller undertegn avtaler før du er helt sikker på at du har forstått vilkårene.
5. Les alltid det med liten skrift, og betal i tide så du unngår ekstra kostnader.
6. Hvis du ikke kan betale en regning i tide, informer de du skylder penger så raskt som mulig – du kan be om å få en nedbetalingsavtale eller betalingsutsettelse.
7. Hvis du ikke klarer å betale det du skylder, er sjansen stor for at du blir kontaktet av et inkassoselskap. I så tilfelle må du reagere raskt, slik at du unngår ekstra kostnader. Forklar situasjonen og be om hjelp til alternative løsninger.
8. Ved å ha en aktiv dialog og være løsningsorientert, kan dere sammen finne utveier på dine betalingsproblemer.
9. Hvis du har en betalingsplan, hold deg til avtalen. Hvis det ikke går, er det viktig at du gir beskjed til inkassoselskapet umiddelbart. Seriøse inkassoselskap, som Intrum Justitia, er til for å hjelpe deg ut av betalingsproblemene.
10. Ikke gjem unna eventuelle økonomiske problemer du måtte få. Prøv å ta grep om din egen økonomi, og husk at det ofte er lurt å søke råd hos andre.

intrum justitia

H Databaseeksempel

```
///// Brukerdokumentet
  {
  "_id": "localUser",
  "name": "Navn Navnesen",
  "expires": "12345667899",
  "token": "SDOFJAPSPASIDNOSIDJFPWEOJFPWOEJFOPWJVAFOJ...",
  "revi": "",
  "email": "navn@online.no",
  "type": "google",
  "savings": [


    /// Sparemalelement
    {
      "dateStart": "2016-05-04T10:23:41.718Z",
      "dateEnd": "2016-07-04T10:23:41.718Z",
      "sum": "15000",
      "id": "2016-05-04T10:23:41.718Z",
      "saved": 0,
      "title": "Ferie"
    },


    /// Sparemlelement
    {
      "dateStart": "2016-05-04T10:28:09.509Z",
      "dateEnd": "2017-03-04T11:28:09.509Z",
      "sum": "30000",
      "id": "2016-05-04T10:28:09.509Z",
      "saved": 0,
      "title": "Ny sykkel"
    }
  ],
  "shareRequests": [
    [
        /// Deleforesporselelement
        {
            "docName": "Boutgifter",
            "sender": "Navny Navnesen",
            "income": false
        },
        /// Deleforesporselelement
        {
            "docName": "Matutgifter",
            "sender": "Navny Navnesen",
            "income": false
        }
```

```
    ],
    [   /// Deleforesporselelement
        {
            "docName": "Felleskonto",
            "sender": "Navny Navnesen",
            "income": true
        }
    ]
  ],
  "_rev": "432-d320bda078ac9f1c3cf8eb821c74d56e"
}

/// Kategoridokument

{
  "_id": "categoryBilutgifter",
  "income": false,
  "name": "Bilutgifter",
  "description": "Alle utgifter tilknyttet bilhold",
  "amountPerMonth": "2000",
  "icon": "icon-bil",
  "expenses": [

    /// Utgiftselement
    {
      "description": "Drivstoff",
      "sum": 562,
      "date": "2016-05-04T10:37:43.029Z"
    }
  ],
  "$$hashKey": "object:481",
  "_rev": "2-cf3d988a3604048e2265909ab69e3b6d"
}

/// Kategorielement

{
  "_id": "categoryLonn",
  "lastUsedRepeaterId": 1,
  "repeaters": [

    /// Gjenntagende utgiftselement
    {
      "id": 1,
      "repeatNumber": 1,
      "repeatEvery": "months",
      "lastTransaction": "2016-05-04",
      "firstDate": "2016-05-04"
    }
  ],
  "income": true,
  "name": "Lonn",
  "description": "Lonninger fra jobb",
```

```
  "amountPerMonth": "23000",
  "icon": "icon-betaling",
  "currentAmount": 0,
  "expenses": [

    /// "utgift" element
    {
      "description": "Fastlonn",
      "sum": 19000,
      "date": "2016-05-04T11:14:31.504Z",
      "repeaterId": 1
    }
  ],
  "$$hashKey": "object:467",
  "_rev": "11-f3bfee7b9d97c4ea0a2e765768e85058"
}
```

Kodeutsnitt H.1: Eksempel på database

I Personvernregler

Personvernerklæring

Sist oppdatert: 10.05.2016

---

1. **Om tjenesten**
   BudgetBuddy er en budsjett-applikasjon levert av Totens Sparebank. Applikasjonen tillater dens brukere å opprette et komplett budsjett basert på SIFO referansebudsjett. Delingsmekanismer gir også muligheten til å dele budsjett slik at flere kan benytte seg av dette samtidig over flere enheter.

2. **Hva samles**
   Applikasjonen lagrer data på brukerens enhet ved normalt bruk. Dette er opplysninger brukeren selv legger inn som budsjett, sparemål og transaksjoner som blir registrert i applikasjonen. Disse dataene blir også lagret sentralt på våre servere hvis brukeren oppretter en konto og logger seg inn. Ved registrering og innlogging blir også navn, epost og passord lagret. Innlogging med sosiale medier henter informasjon fra tjenesten brukeren logger seg inn med. Informasjon som hentes fra slike tredjeparts tjenester er epost og navn. Loggføring for å avdekke feil vil også skje ved kommunikasjon med serveren.

3. **Innsamlingens formål**
   Innsamlet data oppbevares for å kunne levere funksjonalitet applikasjonen tilbyr.
   Data blir lagret sentralt for at brukere skal kunne ha tilgang til sine data på flere enheter.
   Epost, passord og navn blir brukt for å tilby innlogging samt at brukere skal kunne dele budsjett mellom seg.

4. **Samtykke**
   Ved å registrere seg på applikasjonen eller logge seg inn med tredjeparts tjenester vil brukeren gi tilgang til opplysninger nevnt i punkt 2. Dette kreves ikke for enkel bruk. Det er kun hvis brukeren vil ha tilgang til sine data på flere enheter eller dele informasjon med andre brukere dette er påkrevd. Brukeren velger selv å gi tilgang til informasjon.

5. **Tredjeparter**
   Opplysninger som oppbevares deles ikke med tredjeparter. Unntak fra dette er ved å gi tilgang til andre tjenester som vil kreve tilgang til informasjon som kan inkludere hvor ofte og når man benytter applikasjonen. Data som budsjett og transaksjoner vil tredjeparter ikke ha tilgang på.

6. **Sikkerhet**

Brukerens opplysninger er lagret sikkert på våre servere. Kommunikasjon mellom applikasjonen og servere vil være kryptert. Brukere må autentisere seg for å få tilgang til sin informasjon.

J Running and deploying

running and deploying

Budgetapplication frontend.

how to run the project

- install node.js from package manager or from https://nodejs.org
- test node with **$ node -v** in terminal and npm with **$ npm -v**
- install cordova with **$(sudo if unix) npm install -g cordova**
- install ionic **$(sudo if unix) npm install -g ionic**
- install gulp **$(sudo if unix) npm install -g gulp**
- run **$ npm install** inside the project root folder to get all dependencies

run the application in a browser:

$ ionic serve

if application does not work properly run **$ gulp jsConcat** and **$ gulp sass**

add platforms to run on:

$ ionic platform add iOS

$ ionic platform add android

to run on iOS emulator (OS X computer):

$ ionic build iOS

$ ionic emulate iOS

to run on android device:

$ ionic run android

to generate a apk for android:

$ cordova build --release android

Budgetapplication Backend

- install node.js from package manager or from https://nodejs.org
- test node with **$ node -v** in terminal and npm with **$ npm -v**
- install forever with **$(sudo if unix) npm install -g forever**
- install gulp **$(sudo if unix) npm install -g gulp**
- run **$ npm install** inside the project root folder to get all dependencies

to start the server run **$ forever start server.js** in the project root

to stop the server run **$ forever stop server.js** in the project root