Høgskulen på Vestlandet

TingOtek

Informasjonsteknologi

Institutt for data - og realfag

Skrevet av:
Jo-ari Utsi Sara (182043)
Fredrik Christensen (148916)

Veileder: Carsten Gunnar Helgesen
Innleveringsdato: 02.06.2020

Jeg bekrefter at arbeidet er selvstendig utarbeidet, og at referanser/kildehenvisninger til alle kilder som er brukt i arbeidet er oppgitt, jf. Forskrift om studium og eksamen ved Høgskulen på Vestlandet, § 10.

Institutt for data- og realfag

TingOtek

Rapportens tittel: TingOtek	02.06.2020
Forfattere: Jo-ari Utsi Sara og Fredrik Christensen	Antall sider med vedlegg: 36
Studieretning:	Antall disketter/CD-er: 0
Kontaktperson ved studieretning: Carsten Gunnar Helgesen	Gradering: Ingen
Merknad: Ingen	

Oppdragsgiver: Zahlhuus AS	Oppdragsgiver referanse: Ingen
Oppdragsgivers kontaktperson: Fredrik Salhus	Telefon: 90234694

Sammendrag: Denne rapporten beskriver utviklingsprosessen av TingOtek, som er et digitalt lånebibliotek. Ideen er basert på en tidligere app utviklet av Zalhuus AS som heter Pigify. TingOTek er en videre utbygging av Pigify, men med litt andre kriterier. Zahlhuus AS har gitt frie tøyler for utviklingen av prosjektet.

I rapporten vil prosessen av hvordan noen problemstillingen ble løst og evaluering av selve applikasjonen.

Summary: This report describes the development process of the digital general purpose library TingOtek. The idea is based on the previous software developed by Zalhuus AS called Pigify. TingOtek is an extension of Pigify, but with different goals. Zahlhuus AS has granted a liberal workspace for the development of this project.

In this report we are going to explore the process of problem solving and evaluation of the application itself.

Stikkord:

Ionic	REST API	Angular	Postgresql

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap

Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen

Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no
Hjemmeside: http://www.hvl.no

Forord

Denne rapporten blir å forklare utviklingsprosessen av Tingotek og hvordan prosessen førte frem til forskjellige løsninger. Tingotek er et digitalt lånebibliotek, inspirert av Finn.no og den tidligere versjonen Pigify, som er utviklet av Zahlhuus AS. Hele prosjektet ble gjennomført av Jo-ari Utsi Sara og Fredrik Christensen.

Vi ønsker å takke Fredrik Salhus for et spennende oppdrag som har gitt oss muligheten til å utvikle et verktøy for samfunnet. Vi skylder også en takk til Mark Trzopek for å ha vært med å utvikle logo og mye av grafikken til Tingotek. Det har vært interessant å få et innblikk i et prosjekt som det her. I tillegg vil vi takke veilederen Carsten Gunnar Helgesen fra HVL, for god oppfølging og tips til prosjektet. En så kunnskapsrik og erfaren veileder innenfor denne bransjen er sjeldent å finne i dag.

Innholdsfortegnelse

1 Innledning

1.1 Mål og motivasjon

Zahlhuus AS har med hjelp av studenter fra HiB utviklet Pigify, som gir brukere et personlig bibliotek for å låne ut ting og utstyr. En bruker kan få velge om å låne ut eller låne ting fra andre brukere helt gratis. Prosjektet går ut på å videreutvikle dette prosjektet til Pigify 2.0, altså TingOTek, slik at det skal bli enda bedre brukeropplevelse. Målet med dette prosjektet er å lage et digitalt lånebibliotek som kan brukes av kunder og eieren. Som et resultat av det så vil det føre til et mer miljøvennlig samfunn ved å unngå unødvendig kjøp av varer.

1.2 Kontekst

I 2018 publiserte National Geographic en artikkel angående mengden av søppel som blir kastet hvert år. Det ble estimert at 3.5 millioner tonn av plast og annet søppel blir kastet hver dag. Det er 10 ganger mer enn det som ble kastet for 100 år siden (Leahy, 2018). Verden har kommet inn i en sterk konsumer fase i planetens levetid, og det blir å bli verre hvis verden ikke bremser ned utviklingen. I fremtiden som kommer så må befolkningen prøve å komme med bedre løsninger enn det som eksisterer der ute. Verden må gå sammen og innovere de eksisterende løsningene for å stoppe den kraftige økningen av konsumerisme. Spørsmålet er hvordan kan vi bremse ned denne utviklingen, og hva er løsningen? Det er kanskje ikke mulig å redde verden med det første, men det kan starte med å bremse ned den negative utviklingen.

Som sagt så har Zahlhuus med hjelp av tidligere bachelorstudenter allerede skapt Pigify som skal hjelpe med å bremse konsumerisme, men oppgaven videre blir å videreutvikle prosjektet og gjøre det bedre og mer bærekraftig. Målet er ikke å tjene penger på dette, men å gi samfunnet et nyttig verktøy som kan forhåpentligvis hjelpe samfunnet den tunge byrden av over-konsumerisme.

1.3 Avgrensninger

Istedenfor å lage hele prosjektet fra bunnen av, så kan det videreutvikles det som allerede har blitt laget, problemet er at det kan ta litt tid å forstå oppsettet av et gammel prosjekt. Det er i tillegg utviklet ved hjelp av Ionic rammeverk, så tiden blir å gå på å lære det rammeverket som har blitt brukt.

Slik det er nå så er planen at applikasjonen skal ha mulighet å bruke vipps, det vil si at det må skapes et samarbeid med DNB for å lage en sikker løsning. Vipps API må bli brukt og det

kan hende det blir brukt mye tid på det. I tillegg må det sørges for at det er ingen sikkerhetshull som kan bli utnyttet.

Et siste problem er hvordan brukere skal få levere inn ting til TingOTeket. Det er en del av løsningen å belønne utlånere, men bare hvis selve låneren blir forsinket å levere tilbake tinget, det gjør at personen får et purregebyr som må betales. Det er vanskelig å si om det blir å tiltrekke seg utlånere, men for nå så er det løsningen.

1.4 Ressurser

Et av de største ressursene er det tidligere Pigify prosjektet. Kildekoden fra det eldre prosjektet blir å spille en viktig rolle, men det må gjøres endringer og legges til nye funksjonaliteter underveis. Ionic Framework sin dokumentasjon blir å spille en viktig rolle for utviklingen av applikasjonen, det blir også lagt vekt på kunne Angular sin dokumentasjon for videre utvikling. Når det kommer til Bank løsningen så blir Vipps API brukt som en ressurs, i tillegg blir kontaktpersonen fra DNB spille en viktig rolle.

2 Prosjektbeskrivelse

2.1 Praktisk bakgrunn

Zahlhuus AS består av en ansatt som har utviklet appen Pigify med hjelp av studenter. Pigify er et digitalt lånebibliotek som gir privatpersoner muligheten til å skape sine egne biblioteker og eventuelt låne fra andre biblioteker.

2.1.2 Prosjekteier

Fredrik Malmedal Salhus er eieren av Zalhuus AS og prosjektet Pigify. Salhus har jobbet som freelancer innenfor multimedia design i 15 år, og har i tillegg en del erfaring med utvikling av nettsider og apps.

2.1.3 Tidligere arbeid

I 2017 ble appen «Borrowing app» utviklet av tre bachelorstudenter på HiB med hjelp av Salhus. Navnet ble etter hvert byttet over til Pigify og prosjektet ble en suksess.

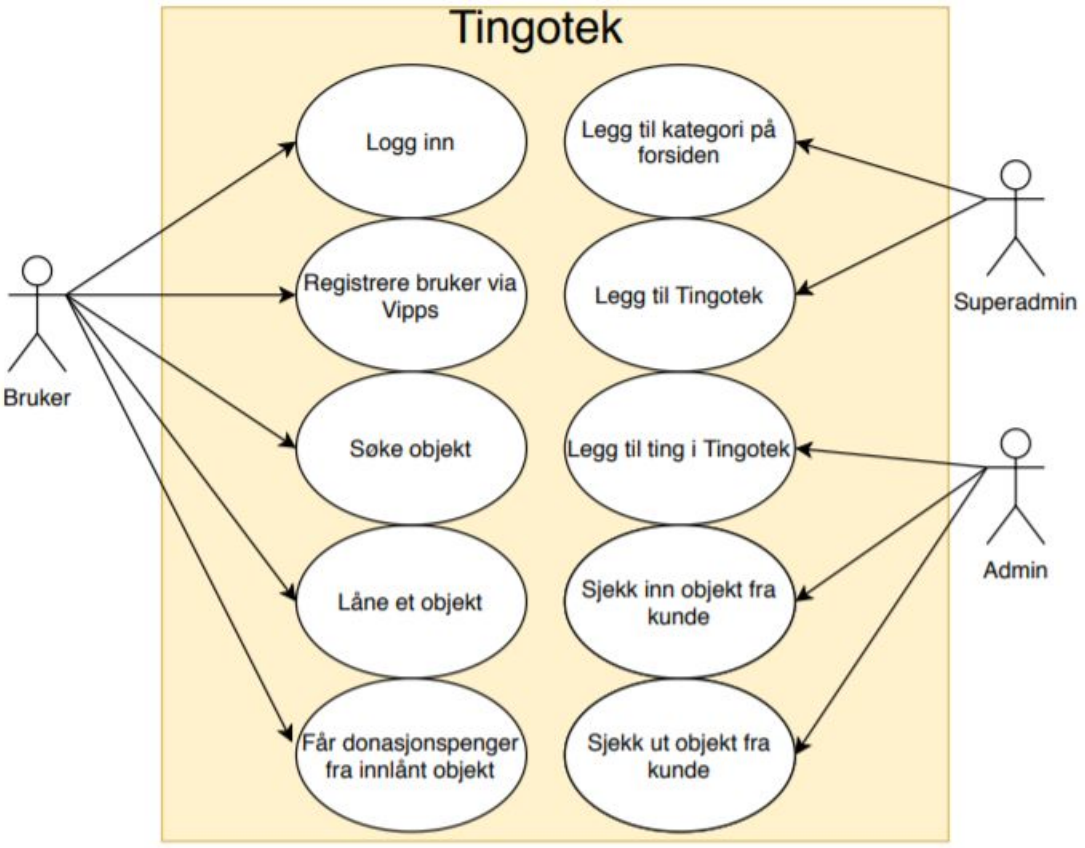
Det har til nå vært i bruk over to år av privatpersoner fra hele Norge, og i tillegg blitt oppdaget i land som USA. Det er altså ikke spørsmål om verdien til en slik løsning.

2.1.4 Initielle krav

Kravene bygger videre på Pigify men med noen endringer. Det ble holdt et møte med prosjekteier hvor de forskjellige kravene ble diskutert, der snakkes det om hvilke utvidelser programvaren skal ha. Oppdragsgiver beskrev et konsept hvor bruker skal ha mulighet til å tjene penger på forsinkelsesgebyrer. Disse forsinkelsesgebyrene skal deles likt mellom systemet og brukeren som legger ut gjenstanden. Videre snakkes det om å forenkle innloggingsprosessen slik at man kan logge seg inn uten passord. Det kommer også opp gjennom samtale at identiteten til brukeren må kunne verifiseres i tilfelle det oppstår misbruk av tjenesten. I tillegg noterte vi disse punktene:

- I motsetning til Pigify så kan bare administrator lage et digitalt bibliotek
- Vipps purregebyr for forsinket levering
- Brukere kan låne et objekt fra en Tingotek i nærheten
- Brukere kan se hva som er ledig
- Brukere kan bli varslet når objektet blir ledig
- Lånehistorikk og vanlig data om brukere skal være tilgjengelig
- Kategori filter
- En fungerende applikasjon for IOS og Android, i tillegg til en web-versjon
- Timer på innleveringsfristen
- Timer på reservasjon

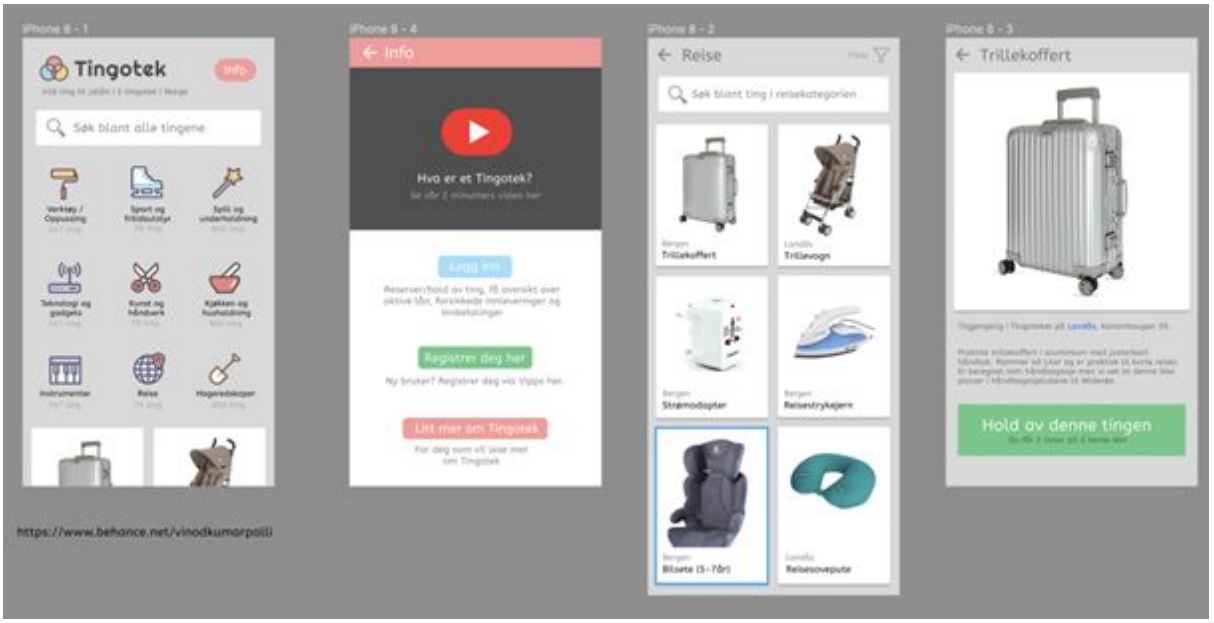
Figur 1 viser en brukstilfellediagram av hvordan systemet skal fungere. Helt enkelt forklart så skal superadmin ha like mange funksjonaliteter som en vanlig admin, men i tillegg til noen flere funksjonaliteter.



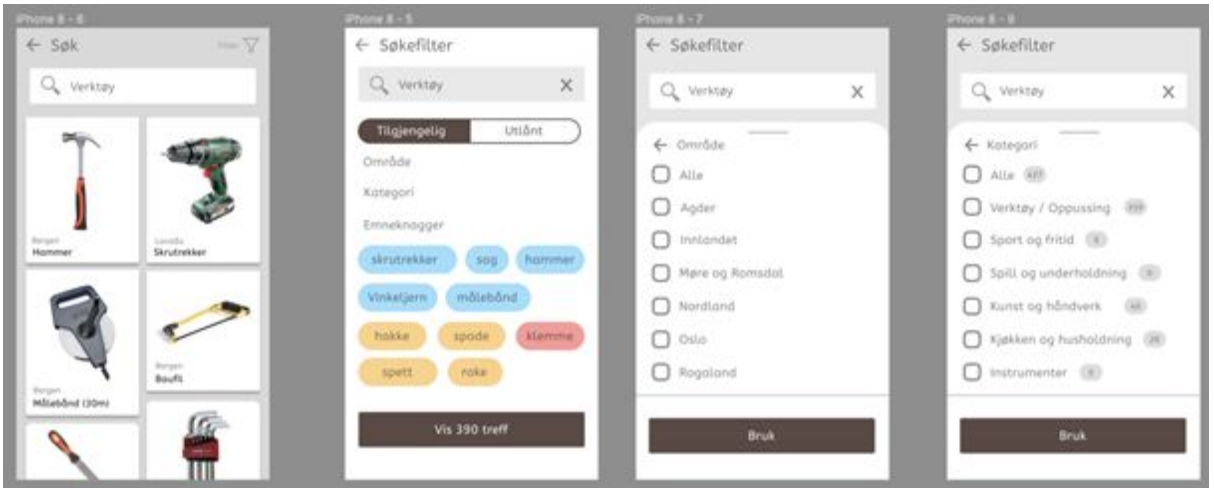
Figur 1 – Et eksempel på hvordan systemet skal fungere

2.1.5 Initiell løsnings-idé

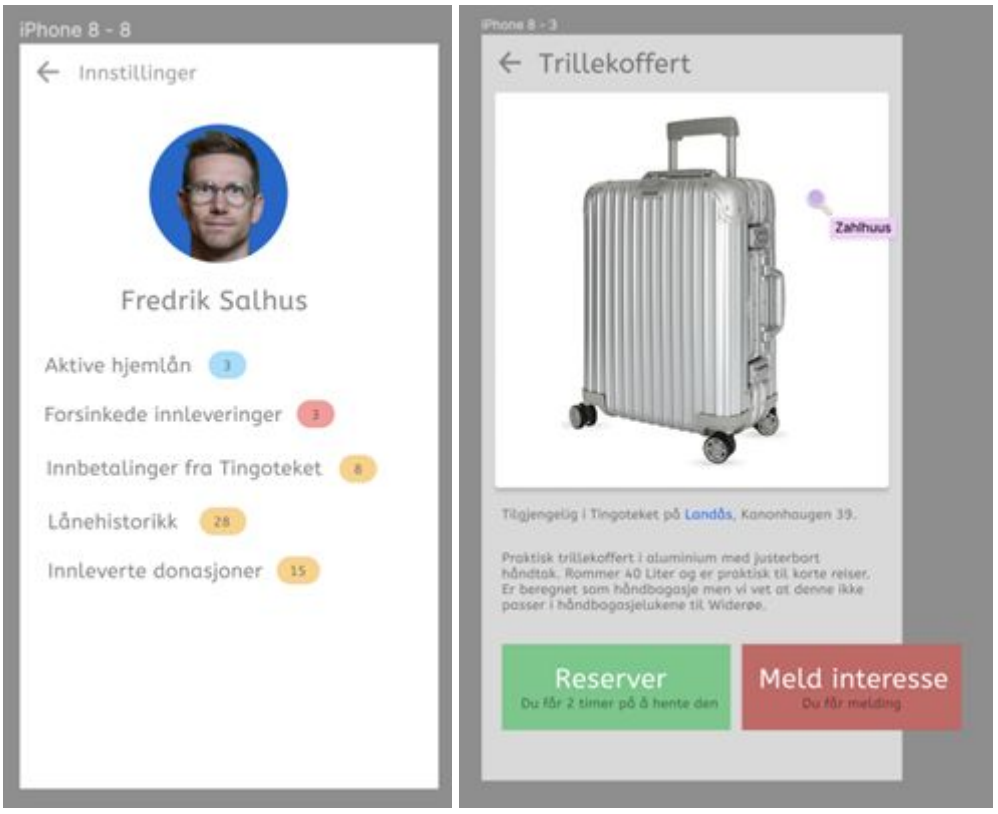
Målet er å ha en enkel designløsning slik at brukeren ikke føler seg overveldet av et komplisert UI. Brukeren skal i tillegg ha mulighet til å se utlåns objekter som kan være av interesse på hjemmesiden av appen. Slik det fungerer, så skal TingOTek skape et digitalt og fysisk bibliotek som blir styrt av Zahlhuus. Brukere kan stikke innom og levere ting og utstyr de har lyst til å låne bort, hvor de tingene da blir sortert og lagt ut på det digitale biblioteket. Brukere har da muligheten å legge av ting som de har lyst å låne og dermed hente det fra det fysiske biblioteket.



Figur 2 – Et eksempel på hvordan noen av app sidene skal se ut



Figur 3 - Viser objekter til utlån og kategori filter



Figur 4 - Personlig info og objekt siden

3.1 Forslag til løsning

Det finnes mange anvendelige løsninger for å utvikle dette prosjektet. Noen av kravene til dette prosjektet er at det skal kjøres i nettlesere som en nettside og som en applikasjon på Android og iOS enheter. Det er tre løsninger som har hovedfokuset og de skal utforskes og diskuteres med hensyn til gruppen sin tilgjengelig tid og eksisterende kunnskap.

3.1.1 Alternativ løsning 1 – Nettside og applikasjon

Den første løsningen består av å utvikle applikasjonen og nettsiden hver for seg på hver plattform. Denne løsningen gir full kontroll over hvordan applikasjonen ser og føles ut på de forskjellige enhetene. Videre kan man benytte seg av de nyeste funksjonene på de plattformene som gjelder.

På en annen side er dette en løsning som kan gjøre vedlikehold av prosjektet svært tungvint, fordi prosjektet mest sannsynlig må deles opp i flere underprosjekter.

3.1.2 Alternativ løsning 2 – Bygge på forrige prosjekt

Den andre løsningen er å bygge videre på det gamle prosjektet "Pigify" i rammeverket Ionic. Ionic tilbyr kryss-nettleser kompatibilitet og muligheter for å konvertere prosjektet til kjørbar kode på flere mobile og stasjonære enheter. Dessuten er hele prosjektet samlet i et hierarki, dette medfører at vedlikehold av prosjektet kan gå enklere. Rammeverket tilbyr også en rekke forskjellige funksjoner som gjør enhets-avhengige funksjonaliteter (som for eksempel kamera tilgang) tilgjengelig under et felles grensesnitt. Igjen er dette noe som letter på vedlikeholdsarbeid.

Når det er sagt er det fortsatt noen utfordringer knyttet opp mot denne løsningen. Det å lære seg alle disse nye teknologiene kan være krevende, derfor har man tenkt seg nøye gjennom de mulighetene man har.

3.1.3 Alternativ løsning 3 – Et rammeverk fra bunnen av

Det tredje alternativet er å lage prosjektet fra bunnen av med Ionic rammeverk. Det tenkes å bruke Pigify kildekoden som et utgangspunkt og hente inspirasjon fra det, men vil i bunn og grunn bygge alt uten å kopiere koden. Som sagt så tilbyr Ionic mange funksjonaliteter som gjør det lettere å bygge applikasjonen, men mengden av ting man må lære seg kan bli et problem. Det positive med denne løsningen er at man kan få full kontroll på hele applikasjonen og slippe å lære seg logikken bak kildekoden til Pigify.

3.1.4 Diskusjon av alternativene

Løsningen for å utvikle dette prosjektet velges ut ifra noen kriterier, man tar til betraktning at det skal kjøres i flere nettlesere og på forskjellige enheter. Det blir også tatt med i betraktning at gruppen bare har grunnleggende kunnskap om design og utforming av moderne nettsider.

Når man ser på løsning nummer èn, har den noen fordeler, men i størst grad ulemper som er særdeles alvorlige. Fordelen med dette alternativet er at gruppen fra før av er kjent med java, javascript, html og css. Dette er noe som er med på å redusere tiden det tar for å komme i gang med utvikling. På en annen side er ikke alle i gruppen så kjent med iOS utvikling, og programvareutvikling på mobiltelefoner. Det argumenteres at det å lære seg et rammeverk for et språk gruppen allerede er kjent med, er lettere enn å lære seg en ny arkitektur, språk og utviklermiljø. Videre har ikke gruppen kapasitet til å utvikle alle de forskjellige applikasjonene for hver plattform og

nettleser. Det kan dessuten også være vanskelig å vedlikeholde så mange applikasjoner hvor kravet er at de skal se likt ut.

Med fordelene og ulempene til første alternativ i mente ser vi på løsning nummer to, nemlig Ionic-rammeverket. Rammeverket lar gruppen utvikle èn applikasjon som kan kjøres på alle nettlesere fra alle enheter. Det at rammeverket tillater gruppen å arbeide med et prosjekt gjør at de kan arbeide mer med utvikling enn feilsøking. Det er også en stor fordel at alt foregår i html, css og javascript som gjør at gruppen ikke trenger å forholde seg til nye språk og arkitekturer. Til slutt er det positivt at rammeverket tar seg av følelse og utseende for hver enhet automatisk. Da dette er sagt kom man frem til at gruppen har tatt et valg som vil spare dem en del tid på finjustering for hver enhet.

3.2 Valgt løsning

Det konkluderes at Ionic er løsningen som passer best, altså løsning nummer to. Det originale prosjektet er bygget på dette rammeverket og fungerer som en referanse til rammeverket. Rammeverket krever at det læres en del nye teknologier og konvensjoner, men det tenkes at det er å foretrekke fremfor de ulempene alternativ èn fører med seg.

Om kravene for prosjektet hadde vært annerledes kunne det ha vært mulig at løsning nummer èn hadde blitt valgt, men slik er det ikke. Derfor velges Ionic rammeverket i videreutviklingen av prosjektet, det vil vise seg om denne løsningen er den beste.

Det viser seg underveis at bruken av løsning nummer 2 ikke er ideell. Den har noen problemer som for eksempel at database systemet ikke støtter enkel håndtering av objekter med mange til mange relasjoner. Det fører til at prosjektet benytter seg av postgresql i stedet for firebase. Mer om dette kan leses i diskusjonen. Det konkluderes derfor at løsning nummer 3 er bedre egnet for prosjektet og vil brukes videre.

3.3 Valg av verktøy

For å finne ut av hvilke verktøy som skal brukes setter vi opp et møte med oppdragsgiver. På møtet blir det notert en liste med forskjellige teknologier det forrige prosjektet benyttet seg av:

- Ionic-rammeverket, for utseende, kryss-kompatibilitet og

enhetsfølelse (Ionic Framework, 2020).

- Typescript, en utvidelse av Javascript med typer (TypeScript - JavaScript that scales, 2020).
- HTML, en standard som nettlesere bruker for å vise nettsider (Html docs, 2020).
- SCSS, en utvidelse av CSS som støtter variabler (Sass: Syntax, 2020).
- Angular, et rammeverk som lar oss bruke MVC-arkitekturen på en enkel måte og som oppfordrer gjenbrukbar kode gjennom moduler (Angular docs, 2020).

- Firebase, som vertstjeneste (Firebase, 2020).
- Github som oppbevaringssted (How developers work, 2020).
- Google drive, som et felles område for å lagre relevant informasjon (Alle filene dine - tilgjengelige der du er, 2020).
- Firebase Firestore, som initiell database system (Firebase, 2020).
- Postgresql, som valgt database system (About, 2020).

For innlogging og betaling tenkes det å bruke BankID og Vipps, for innlogging og betaling av gebyrer henholdsvis (Vipps Invoice API, 2020). Det oppdages umiddelbart at Vipps bruker BankID bak kulissene, noe som gjør at direkte integrasjon med BankID kan ses bort fra. Vipps velges da for innlogging siden den er sikker nok på grunnlag av at det bruker BankID (Vipps Login API, 2020). Dette er gode nyheter siden det kun er ett API å forholde seg til i stedet for to da. Som man kan se på verktøylisten er Firebase Firestore den initielle database løsningen, den nåværende database løsningen er postgresql på grunn av problemer som kan sees i diskusjonen.

Videre diskuteres NFC teknologi og rollen den har ved å organisere gjenstander på de forskjellige tingotekene. Near Field Communication eller “NFC”, er en teknologi som innebærer at mobilen kan snakke med et NFC objekt uten Wi-fi, mobilnett eller andre alternativer. Ved å ta mobilen i nærheten av taggen så kan mobilen og taggen kommunisere med hverandre. Vi finner ofte den teknologien i dag, eksempel

betaling med mobiltelefon på butikker, og kredittkortet som kommuniserer med terminalen som da igjen kommuniserer med banken (Montegriffo. 2020). Det viser seg at iOS enheter ikke tillater adgang til NFC funksjoner i tidsperioden original-prosjektet ble utviklet. Dette er heldigvis ikke situasjonen lenger, så det menes at NFC funksjonaliteter på iOS enheter skal realiseres denne gangen. Dette gjøres ved å bruke Ionic rammeverket.

Til slutt diskuteres visjonen til oppdragsgiver, han konstaterer at dette er et lukket prosjekt i motsetning til original-prosjektet. I første versjon av prosjektet tillates det at enhver bruker kan opprette et bibliotek, mens den nye versjonen skal heller benytte seg av en mer kontrollert løsning. Den nye løsningen skal tillate brukere å opprette biblioteker via en etterspørsel som oppdragsgiver personlig kan bekrefte eller avkrefte. Videre skal også brukere få lov til å se innhold i tingotekene uten å logge seg inn, men da uten tilgang til videre funksjoner. Dette løses med å sette inn roller på forskjellige funksjoner, da har prosjekteier superadmin rolle, Tingotek ansatte har admin rolle, og brukere har bruker rolle.

3.4 Prosjektmetodikk

3.4.1 Utviklingsmetodikk

Vi følger en «agile» arbeidsmetode hvor man praktiserer filosofien om å angripe risiko tidlig, sørge for å ha kjørbar kode til enhver tid og til slutt unngå omfattende endringer mellom hver iterasjon (Kroll and Kruchten, 2003). Det gis stor oppmerksomhet på disse punktene fordi prosjektet har så mange alvorlige risiko. Dessuten testes koden jevnlig for å sikre kjørbar kode og minske tiden brukt på feilsøking. Den agile arbeidsmetoden har en del punkter som definerer den, punktene vi valgte å følge er blant annet: å vise kjørbare eksempler til prosjekteier, tilpasse oss endringer i kravene, ha jevnlige møter og kommunikasjon med hverandre, basere fremgang på kjørbar kode, bærekraftig arbeidsmengde, fokusere på å gjøre ting enkelt og velge de enkleste løsningene.

Utviklerne seg imellom har noen grunnregler og bestemte verktøy for å gjøre samarbeid og evaluering av kode enklere:

- Ordrike variabler i stedet for akronymer
- Teste koden jevnt ut til alle tider
- Forholde oss til Model-View-Controller mønsteret

- Arbeide på hver sin forgrening av skjellet-prosjektet på GitHub
- Flette arbeid ofte
- Bruke Visual Studio som utviklermiljø

3.4.2 Prosjektplan

Planleggingen av prosjektet startes ved å lage et aktivitetsdiagram. Den beskriver hvordan flyten i programmet skal gå, og funksjonene den trenger. Deretter blir det enighet om å innføre prioriteringer på et felles dokument, øverst på listen føres følgende punkt: forståelse for rammeverket, kjørbar original-prosjekt, oppsett av GitHub oppbevaringssted, lage utkast av Vipps innlogging, Vipps gebyr-system og iOS NFC funksjonalitet. En del av disse punktene kjøres parallelt (grunnet tidspress) slik som læring av Ionic-rammeverket, Angular-rammeverket, Firebase plattformen, Vipps API'et og NFC-teknologien.

Gruppens arbeid med denne planen gikk saktere på grunn av betydelige begrensninger, det viser seg at den er meget uoversiktlig. For å løse denne problemstillingen lages det et Gantt diagram hvor mål og milepæler føres ned. Disse punktene blir videre delt opp i tidsspalter og hvem i gruppen som har ansvar for hvert punkt. Til slutt har man en oversikt over kategoriserte punkter faller under fem forskjellige faser: spesifisering, design, programmering, integrering, evaluering og levering.

I spesifiseringsfasen har vi et møte med prosjekteier og faglig veileder. Det samles informasjon om hva prosjektet handler om, hvilken krav den har og en estimat av fremtidig tidsbruk på hvert krav.

I skissering fasen lages det brukstilfeller, diagrammer, mulige løsninger og tegninger.

I designfasen gås det mer i detaljene av modulene, hvilken moduler det kan tenkes at de vil trenge, flyt av programmet og en mer oversiktlig diagram av arkitekturen.

I programmeringsfasen programmeres de mest grunnleggende modulene, funksjoner og annet som de andre større modulene skal benytte seg av.

I integreringsfasen utvikles større komponenter som benytter seg av de grunnleggende modulene, slik som inlogging og andre krav.

I evalueringsfasen testes applikasjonen av en gruppe testere som prosjekteier har funnet.

Til slutt, i leverings fasen benyttes Firebase plattformen for å lansere applikasjonen og server, og Amazon Web Services for å laste opp database.

3.4.3 Risikovurdering

For å vurdere risiko tas listen med brukstilfeller som er gjennomgått med oppdragsgiver, og gjennomgås trinnvis. Hvert brukstilfelle blir sammenlignet med gruppens eksisterende og manglende ferdigheter, dette gjøres fordi man kan lage en liste over situasjoner som kan forekomme av dette. Situasjonene blir deretter ført inn i en risikomatrise hvor sannsynligheten drøftes for hvert av punktene, hvordan det kan utarte seg og prioriteringsgraden som vi regner ut. Prioriteringsgraden blir regnet ut ved å multiplisere sannsynlighetsgraden og alvorsgraden.

Etter noteringen av de forskjellige tilfellene, tenkes det frem forskjellige løsninger på hvordan disse situasjonene kan håndteres. Disse løsningene ligger i siste kolonne av risikomatrisen.

3.5 Evalueringsplan

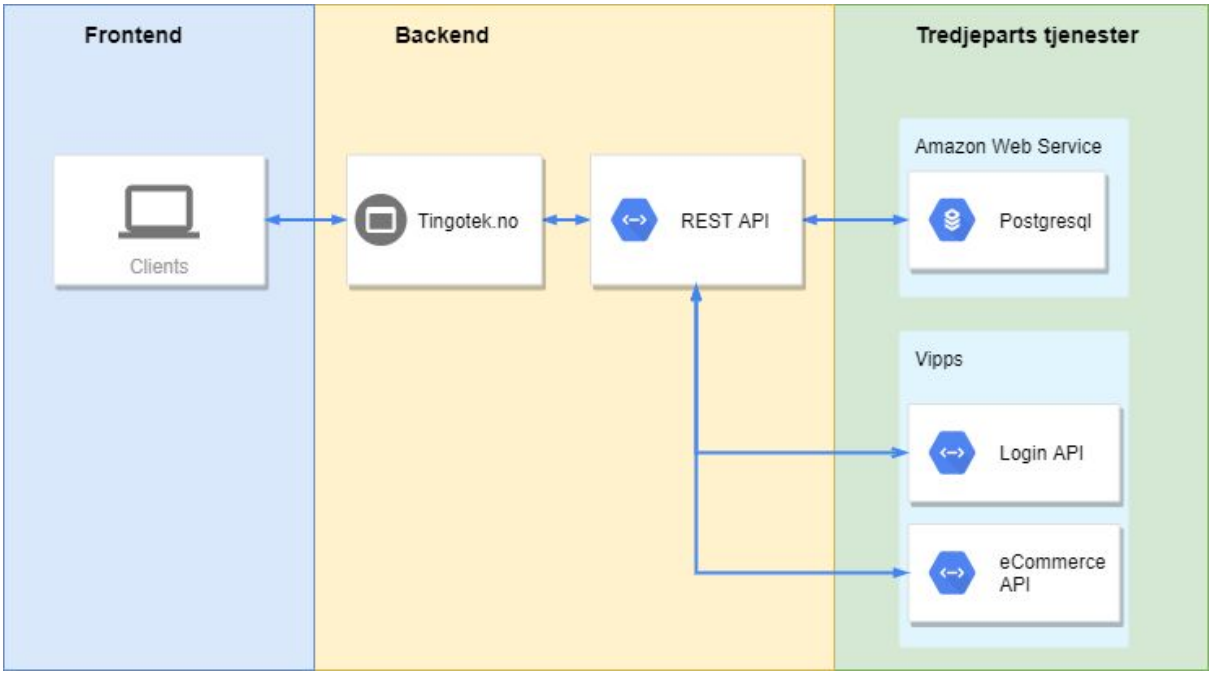
Oppdragsgiver tilbyr seg å holde styr på distribusjon av programvare til test-brukere. Disse brukerne består av tidligere Pigify brukere som har gitt tilbakemeldinger og er kjent med applikasjonen. Dette anses å være en god løsning fordi disse brukerne allerede er kjent med et lignende system og ganske sikkert vet hva de skal se etter. Det er også integrert unit-testing med et test-rammeverk kalt “Jasmine” (Jasmine documentation home, 2020), den inkluderer såkalte “mockups” som gjør testing av nettverksbaserte tjenester enklere. Til slutt ser vi på tilbakemeldinger fra oppdragsgiver og veileder for å styre prosjektet i riktig retning.

4 Detaljert design

4.1.1 UI

Et viktig punkt for denne applikasjonen er en forståelig UI som respondere godt for flere plattformer. Siden vi gjorde valget om å ikke lage mobilapplikasjoner, men bare en webside, så må vi fortsatt gjøre det like enkelt for mobilbrukere å få brukt nettsiden. Ionic Framework gir oss et godt verktøy for å nå disse målene, med knapper som lett kan skaleres basert på størrelsen til skjermen, gjør det enkelt å bygge websiden. Et av de tidligere problemene med

Pigify var at det kunne til tider være vanskelig å forstå hvordan applikasjonen blir brukt. Vi har valgt å endre på store deler av det prosjektet og gjøre det lettere for kundene å bruke.



Figur 5

4.1.2 Figma

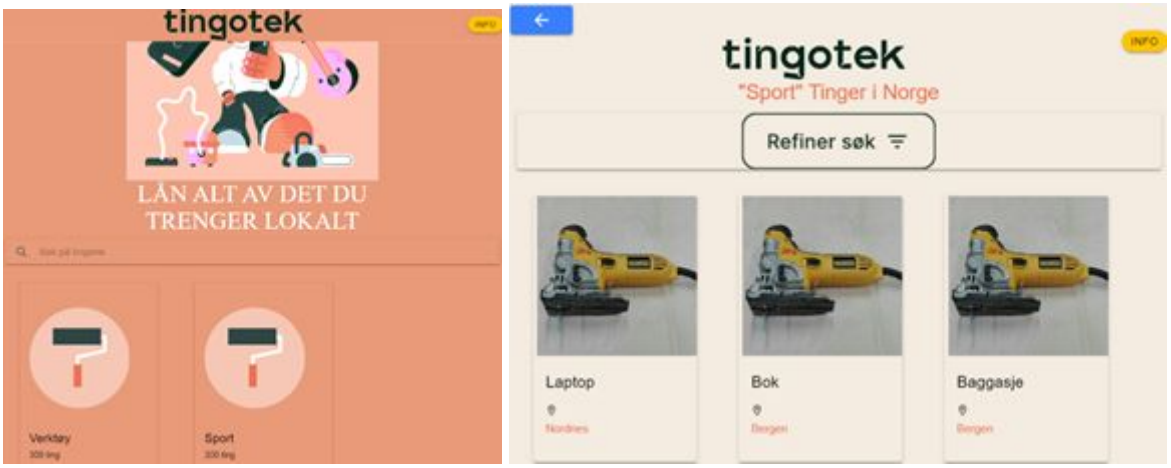
Et viktig verktøy som vi bruker, er Figma (Figma Design, 2020). Arbeidsgiveren har god kjennskap til det verktøyet og kan lett redigere hvordan applikasjonen skal se ut. Mesteparten av grafikken blir gjort av en tredjepart grafiker ansatt av Salhus. Han heter Mark Trzopek og er en dyktig grafiker. Salhus og Mark har samarbeidet for å lage et Figma prosjekt for å vise hvordan de vil at det skal se ut. Vi har hatt en grundig dialog om hva som er mulig, og hva som ikke er mulig. Veldig mye av elementene kan «kopieres» direkte over til prosjektet, slik som CSS koder for bakgrunn, font etc. Det er i tillegg veldig mye grafikk som er lagt til der som vi kan hente som SVG bildeformater. Her er et eksempel på hvordan Salhus og Mark har laget sidene.

Figur 6 Eksempel på Figma format

4.1.3 Hjemmeside

Slik vi har gjort det til nå så kommer kunden inn i en hjemmeside på Tingotek.no. Det blir å vise de forskjellige kategoriene som kan velges, eller eventuelt søke opp globalt. Et av målene for Tingotek er at Salhus blir en såkalt «super admin» som kan lage nye Tingotek biblioteker, hvis det da eventuelt skulle åpnes opp flere av dem. Så hvis kunden trykker på en spesifikk kategori så viser den alle gjenstandene fra den spesifikke kategorien, men fra alle lokasjoner. Det blir mulig å filtrere et søk slik at kunden kan søke opp alle gjenstander fra en spesifikk lokasjon. Vi vil gjerne også nevne at vi har hentet litt inspirasjon fra Finn.no sin nettside, da vi føler de har fått til en veldig god løsning. Vi føler at hjemmesiden må være enkelt å forstå. Kunden skal ha mulighet til å se gjennom alle gjenstandene, men det er bare når kunden bestemmer å reservere en gjenstand at man må da logge på via vipps.

Her er et eksempel bilde av hjemmesiden og kategorisiden.



Figur 7 Hjemmeside

Slik det er nå så føler vi at vi har fått til en veldig enkel design som responderer bra på både mobil og standard nettside. Vi bruker Chrome sin developer tools for å teste skaleringen av websiden og i tillegg siden vi bruker SVG bildeformat på kategori-bildene så skalerer den veldig fint automatisk uten å måtte endre på noe kode.

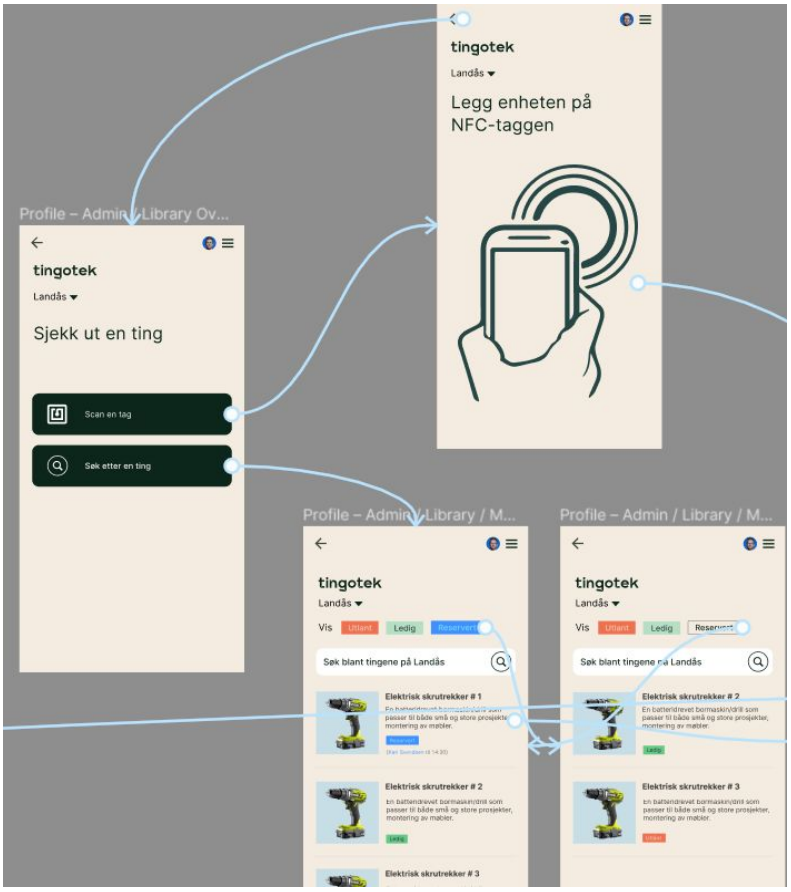
4.1.4 Admin

En kunde kan enkelt lage en ny bruker og bruke det for å låne gjenstander som finnes i Tingotek, men hvis en kunde skal ha tilgang til å bli en administrator for et eget Tingotek så må det bli godkjent av eieren. En administrator har muligheten til å legge inn nye gjenstander i Tingoteket, og godkjenne lån. Slik vi tenkte så skal en administrator essensielt bare være en standard bruker, men med litt ekstra funksjonaliteter. I eksempelbildet under, så har kunden muligheten til å se egen lånehistorikk og antall donasjoner. Det som skiller mellom brukeren og administrator er kunden kan ikke godkjenne lån eller låne gjenstander uten godkjenning. Slik vi løser det er at vi har laget en standard informasjonsside som gir funksjonaliteter til å se egen informasjon, men en administrator får noen ekstra knapper og sider som kan bli besøkt. Det blir løst ganske enkelt med at vi bruker en Angular «if» metode som sjekker om brukeren er en administrator. Hvis det stemmer så får personen tilgang til flere funksjoner.

Vi har i tillegg en ekstra bruker som er «super administrator», altså eieren av hele Tingoteket. Det blir det samme funksjoner som en bruker og administrator, men en ekstra funksjon som gjør at eieren kan lage nye kategorier og lage nye admin-brukere. Her er et eksempel bilde på hvordan det skal se ut:

Figur 8 Admin funksjonaliteter

En funksjonalitet som både admin og super admin kan gjøre er å "sjekke ut" og "sjekke inn" en ting på en Tingotek. Helt enkelt forklart så betyr det at hver gang det kommer en kunde som skal låne en ting så kan admin sjekke ut en gjenstand. Administrator kan velge med å søke opp den spesifikke tingen som er reservert og hente "id" koden, eller velge å skanne en NFC tag på gjenstanden. NFC skanningen gjør ting lettere og det fungerer så og si som et biblioteksystem hvor kunder kan sjekke ut bøker. På figur 9 vises et eksempel på hvordan "sjekk ut" fungerer.



Figur 9 "Sjekk ut" funksjonalitet

4.2 Ionic Framework

Det er enighet i gruppen om at det er viktig å gå litt innpå Ionic sine metoder. Slik som Ionic fungerer så er det da altså et rammeverk som gir enkelt tilgang til forskjellige funksjoner. Ting som knapper, moduler, routing, og enkel CSS koder blir godt forklart på dokumentasjonen til Ionic (Ionic Framework, 2020). Det kan tenkes slik at Ionic tilbyr en bedre måte å utvikle Front-end applikasjoner, det er fullt mulig å bruke standard javascript og html, men bruken av Ionic tillater en kortere utviklingsfase. Det er også viktig å nevne at Angular brukes i Ionic rammeverket, frem til nå har det ikke vært særlig behov for det, men det kan komme godt med senere i utviklingsfasen. I tillegg til Angular så er det også mulig å bruke React og Vue, men Angular er å foretrekke da det ble brukt i Pigify prosjektet.

4.3 Server og API

Serveren som støtter prosjekter er laget med nodejs, det er et Javascript kjøremotor som er designet for web utvikling (About Node.js, 2017). Sammen med nodejs brukes express rammeverket for å utvikle API'et. Express tillater enklere konstruksjon av API-endepunkter i form av "routes" som man kan videre spesifiseres etter hvilken type kall man ønsker å gjøre

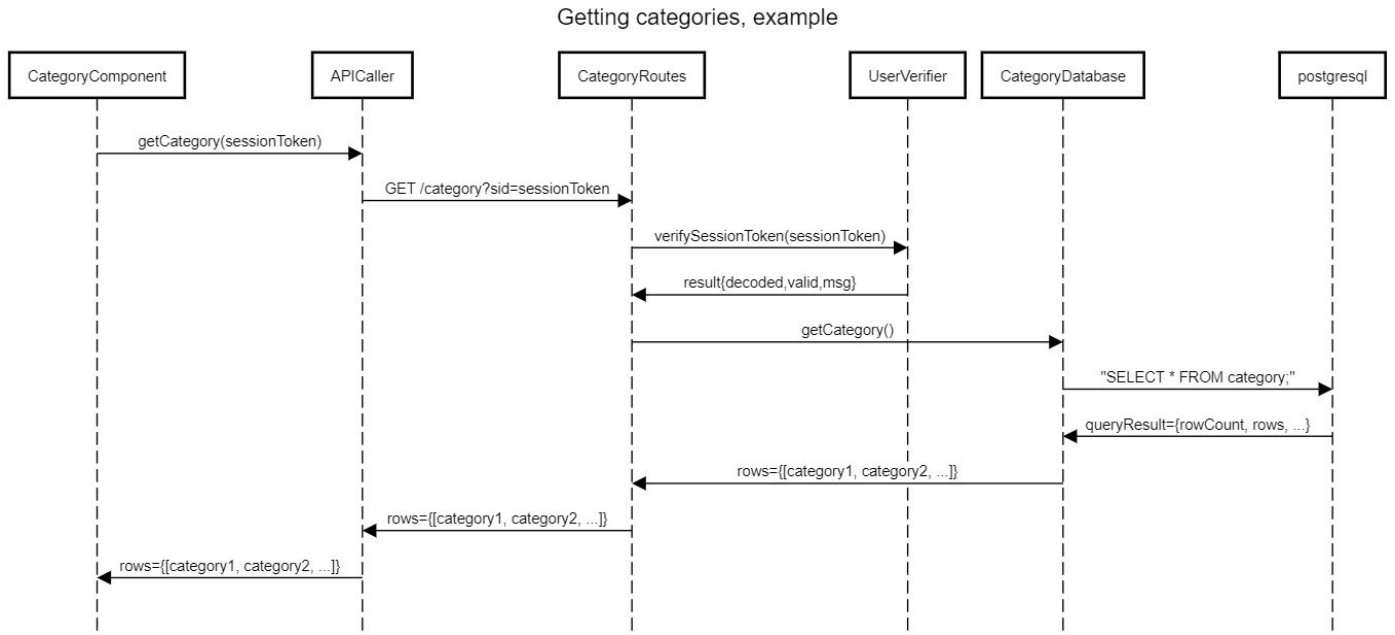
(Express routing, 2020). API'et følger REST arkitektur stilen, det vil si at den blant annet er tilstandsløs og fungerer isolert som egen modul (What is REST, 2018). Endepunktene i API'et beskyttes med tre lag, signering av brukeren sin sesjonsnøkkel ved innlogging, roller hvor sesjonsnøkkel sammenlignes med brukerroller i database og til slutt sjekke at kallet ble sendt fra domenet som brukes i prosjektet. Sesjonsnøkkelen lages ved å ta en identifiserende streng fra Vipps Login API sitt profil endepunkt og signerer den med JSON Web Token teknologi(JSON Web Token Introduction, 2020). På denne måten føles det at tilstrekkelig sikkerhet er oppnådd.

4.3.1 Vipps Login

Ved innlogging får klienten en sesjonsnøkkel som er signert med en privat nøkkel. Denne nøkkelen er som tidligere nevnt hentet fra Vipps sitt Login API. Sesjonsnøkkelen blir videre brukt i applikasjonen for å identifisere brukeren og bestemme hvilken handlinger brukeren tillates. Innloggingen via Vipps skjer gjennom oAuth2 protokollen med OpenID laget. Prosessen kan beskrives gjennom noen steg:

1. Brukeren trykker på Vipps login knappen
2. Brukeren videresendes til Vipps hvor videresendings URL'en inneholder informasjon om applikasjonen sin legitimasjon, data den ønsker å hente ut og videresendings URL som Vipps skal sende brukeren tilbake til.
3. Brukeren får mulighet til å akseptere eller avslå delingen av informasjon.
4. Brukeren sendes tilbake til opprinnelsessted hvor URL'en nå enten inneholder en autorisering nøkkel eller ikke, basert på samtykket gitt av brukeren.
5. Autorisering nøkkelen sendes til API'et hvor autorisering nøkkelen byttes med Vipps for en aksess nøkkel.
6. Til slutt kan aksess nøkkelen brukes for å hente ut brukerinformasjon på Vipps sitt profil endepunkt.

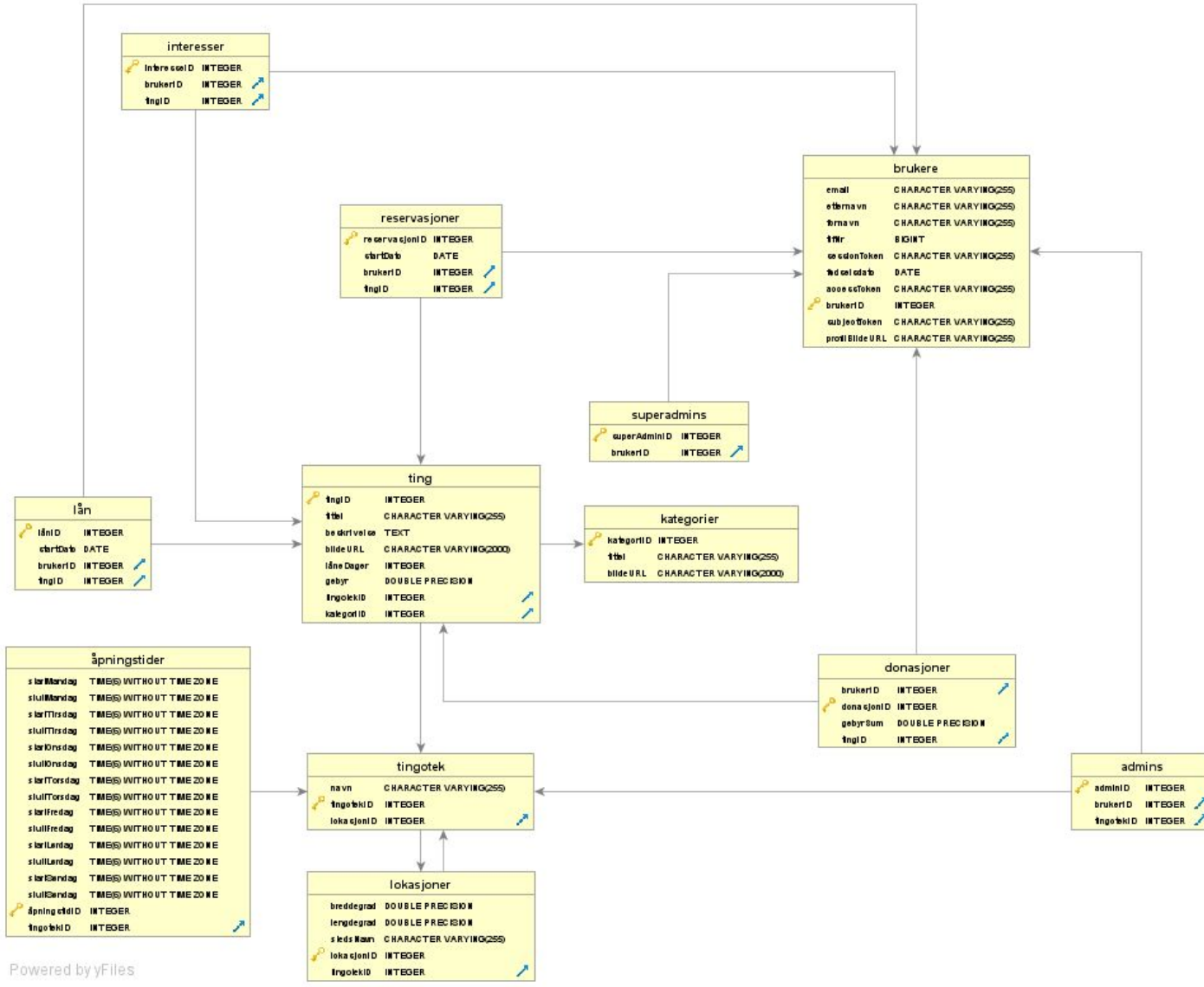
Grunnen til at nøkkel vekslingen utføres via API'et er at denne informasjonen er sensitiv, om en annen bruker hadde fått tak i denne kunne de i teorien ha hentet ut informasjon om denne brukeren. Ved å bruke API'et avdekkes aldri aksess nøkkelen for brukeren, og kan derfor regnes som sikret. Når informasjonen om brukeren til slutt er skaffet lagres den på databasen for fremtidig bruk. Følgende sekvensdiagram viser den generelle prosessen ved kommunikasjon mellom klient, server og database/vipps:



Figur 10

4.3.2 Database

Databasen er laget med postgresql, et database system som er særdeles nyttig i situasjoner hvor det er mange objekter med assosiasjoner til hverandre. Postgresql bruker SQL språket for å sende og hente informasjon i form av spørringer. For å benytte seg av disse spørringene i server miljøet nodejs brukes et bibliotek kalt node-postgres. Dette biblioteket er nyttig for å holde styr på databasetilkoblinger og spørringer. Databasemodulene i prosjektet er designet etter tabellene i databasen, men bruker en felles singleton modul som holder styr på et begrenset antall mulige tilkoblinger. Dette gjør at spørringer går raskere fordi flere brukere kan benytte seg av en kobling som allerede eksisterer i stedet for å opprette en ny tilkobling for hver spørring. Dette fører også til at flere klienter kan behandles samtidig i stedet for at de håndteres på en “First-In-First-Out”-måte. Hver modul inneholder spørringer som API’et bruker når den skal hente eller sette inn informasjon. Databasen kan visualiseres med dette relasjonsdiagrammet:



Figur 11

Databasetilgang utføres via API'et, den har en mengde endepunkter som følger REST arkitekturen. Den går ut på at api-kallene skal være tilstandsløs, det vil si at den ikke skal huske noe fra forrige api-kall og dermed kunne brukes om igjen flere ganger uten bivirkninger. Endepunktene følger bruksområdene i prosjektet, den er ikke direkte basert på database strukturen, samt inneholder den et endepunkt for oAuth2 autorisering med vipps og henting av profil data. De fleste endepunktene i API'et støtter GET, PUT, POST og DELETE-kall. GET-kall i et REST api skal hente data, PUT-kall skal oppdatere data, POST-kall skal sette inn nye data og DELETE-kall skal slette data (What is REST, 2018). Man refererer til data som skal behandles av API'et ved å oppgi relevant ID som også ligger i databasen.

5 Evaluering

I dette prosjektet settes det stor oppmerksomhet på tilbakemeldinger og løsninger på hvordan oppgavene skal utføres. Dialogen med eieren er god og man har private møter der man finner ut av de beste løsningene ved å holde samtaler. I de forskjellige delene av prosjektet finnes det forskjellige evalueringsmetoder, herunder kan man se nærmere på det.

5.1 Evalueringsmetode

For å evaluere designet i prosjektet har man hatt tett oppfølging med prosjekteier. Det nevntes tidligere at prosjektet får hjelp av en tredjeparts grafiker som tar seg av design og grafikk av prosjektet, samt hvilken side som går til hva. Det holdes møter ofte hvor prosjekteier får se på fremgangen av prosjektet, der kan det gis tilbakemeldinger. Disse tilbakemeldingene er svært verdifulle fordi det hjelper med å styre prosjektet i riktig retning, det dannes også en felles forståelse av hva som er mulig å lage. For å teste integrasjonen har man en felles Github- og Figma-bruker som alle har tilgang til, hvor eieren kan hoppe inn og vurdere. Figma er et designverktøy der man kan definere overgang fra en side til en annen, samt hvordan siden skal se ut. Siden eieren ikke har så god kjennskap til programmering så kan han ikke evaluere kode, men det finnes noen løsninger til det.

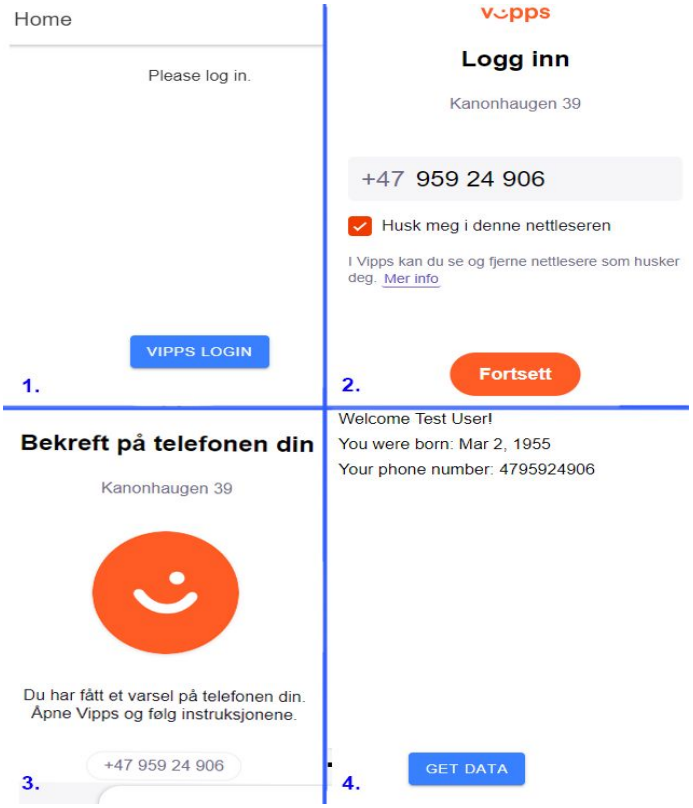
For å evaluere vipps innloggingsmodulen brukes det en enkel nettside som man har laget. Denne nettsiden demonstrerer dette ved å vise brukerinformasjon som hentes fra Vipps sitt API på skjermen. Det menes at dette er en god nok evaluering for denne modulen, fordi det viser at modulen åpner en kobling med Vipps sitt profil endepunkt. Denne koblingen er videre basert på aksess nøkkelen som hentes i nøkkelvekslingsprosessen mellom prosjektet sitt API og Vipps sitt Login API. Videre blir evaluering av mindre funksjoner gjort i form av kall som sendes med Postman verktøyet. Der testes for eksempel videresending og tilbakesending av brukeren, nøkkelveksling mellom prosjektet sitt API og Vipps sitt Login API, og til slutt profilhenting med aksess nøkkelen.

For å evaluere prosjektet sitt API brukes Postman også. Postman er et api-teste plattform for å lage og sende api-kall, man kan også konfigurere det til å bruke verdier og skripter for å automatisere en del prosesser (The Collaboration Platform for API Development, 2020). Postman er et veldig nyttig verktøy i Vipps integreringen siden man får et oversyn av dataen Vipps responderer med. Da kan man bruke denne informasjonen til å forutse hvilken data man kommer til å få. Man får også informasjon om feilmeldinger som man kan sjekke opp i Vipps sitt Login API dokumentasjon side. Postman kan også lagre api-kallene slik at man kan bruke dem om igjen. Den ble brukt for å evaluere prosjektet sitt API ved å sende forskjellige kall til den. Resultatet man fikk ut ifra kallene blir sammenlignet med antatt resultat, og deretter justeres API'et til man får de resultatene man ønsker. Gruppen ser på dette som

integrasjonstesting, siden disse kallene benytter seg av flere database-kall og interne funksjoner for å konstruere et svar.

5.2 Evalueringsresultat

Etter finjustering av prosjektet sitt API ble det observert at dataen man fikk tilbake samsvarte med forventningene. API'et ble derfor sett på som evaluert og klar for bruk. Vippslogin modulen ble sett på som evaluert etter at den gav oss de resultatene vi forventet å se, dette kan observeres med den enkle nettsiden som ble laget på figur 12.



Figur 12

6 Diskusjon

Konsekvensene av å velge Ionic rammeverket og de verktøyene som er tidligere nevnt er forskjellige. For eksempel kan oppbyggingen av loginmodulen gjøre fremtidig skalering av prosjektet vanskeligere. Loginmodulen er bygget rundt Vipps sin innloggingstjeneste og oAuth2 protokollen, men spesielt knyttet opp til Vipps. Det tenkes at denne modulen kan skaleres om man omstrukturerer deler av innloggingsmodulen. Grunnen til at denne modulen kan medføre skaleringsproblemer er at den bruker en del hardkodede variabler, dessuten baserer Vipps seg på norske brukere. Dette resulterer i at hvis prosjekteier i fremtiden ønsker å utvide prosjektet til utlandet, må loginmodulen endres en del. På en

annen side har Vipps innloggingen gitt applikasjonen en viss troverdighet, noe som kan muligens redusere forsøk på tyveri i fremtiden. Dessuten gjør denne løsningen prosjekteier sikker på at brukerne er ekte. Dette er viktig fordi utlåning av ting kan komme med noen risiko, som at brukeren kan velge å ikke levere gjenstanden tilbake. Hvis dette skjer kan det være en fordel å ha kontaktinformasjonen til brukeren slik at forsinkelsesgebyret har en mottaker.

Integreringen mot Vipps krever at man sender skreddersydde HTTP-kall som oppfyller oAuth2 spesifikasjonen (The OAuth 2.0 Authorization Framework, 2012). Her bruker vi et bibliotek kalt Axios. Axios er et promise basert HTTP klient som kan brukes i både nettleseren og på nodejs. Axios blir brukt for å konstruere oAuth2 kall som sendes til Vipps. Den virker utmerket for å kommunisere med Vipps.

Firebase viste seg å ikke egne seg som database system for prosjektet vårt. Vi oppdaget at den ikke var velegnet for situasjoner hvor det er mange objekter med mange-til-mange sammenhenger mellom seg. Den baserer seg på en trestruktur av data, noe som gjør at uthenting av data blir meget komplisert. For å hente ut en bit med data fra en "grein" må man hente ut hele "treet". Derfor velges postgresql i stedet, et databasesystem gruppen er godt kjent med og som støtter objekter med mange relasjoner. Konsekvensen av dette er at database koden må byttes ut.

Siden Ionic rammeverket tilbyr tilgang til enhetspesifiserte funksjoner har man spart mye tid. I stedet for å finne biblioteker finnes alt man trenger i ett rammeverk. Cordova plugins gjorde arbeidet med enhetsspesifiserte funksjoner mye enklere og kjappere. Noen ting kunne ha blitt gjort annerledes, for eksempel kunne oAuth2 kallene ha blitt konstruert med axios i begynnelsen i stedet for å prøve å integrere de bibliotekene som ble funnet. Det hadde også spart en del tid om postgresql ble brukt i begynnelsen i stedet for firebase.

Et bemerkelsesverdig problem oppstod under utviklingen av prosjektet sitt API. Javascript sin dynamiske tolking av datatyper gjør henting av data fra database og vipps ganske uoversiktlig. Etter å ha hentet data i form av json må man ha kontroll på hvilke attributter den har, dette kunne ha blitt fikset med et interface i Typescript. For øyeblikket blir JSON og kommentarer brukt i et forsøk på å dokumentere antatte retur verdier.

Ionic

Et av konsekvensene for å bruke Ionic var at det var ukjent terreng, men etter å ha sett hvor enkelt det er å lage en enkel applikasjon så ble det bestemt for å bruke det. Det har gått litt tid på å lære seg alle funksjonene som Ionic tilbyr, og i tillegg få teste dem, men det gikk greit etter en stund med læring og feiling. Et av problemene som oppsto var at Pigify ble bygget på en eldre versjon av Ionic (altså Ionic 2). Det ble gitt et par forsøk på å oppdatere prosjektet til ionic 5, men måtte til slutt gi opp. Den enkleste løsningen var å lage alt fra begynnelsen, men bruke Pigify som en guide videre i utviklingsfasen. En annen positiv fordel

var at man ikke lenger trengte å utvikle for mobiler, men bare en webside, som gjorde det enklere.

Noe som kunne ha bli gjort annerledes var å bruke mindre tid på å forstå koden til Pigify. Som sagt så ble det utviklet av tidligere bachelorstudenter, og det var veldig lite kommentarer for å forholde seg til. Den tiden kunne blitt brukt på å lage en egen løsning, istedenfor å bygge videre på Pigify prosjektet. Det kan nesten sies at det er to forskjellige applikasjoner.

7 Konklusjon og videre arbeid

Akkurat nå nærmer vi oss målet. Prosjektet har kommet langt og det har hatt stort utbytte av å bruke de forskjellige teknologiene. Vi har nå erfaring med moderne javascript rammeverk, REST api-utvikling og hybrid-app utvikling. Det finnes fremdeles noen moduler som må fylles ut, spesielt gebyr systemet og noen endepunkt i API'et. Vipps invoice API'et viser seg å være den største risikoen for øyeblikket, men alt annet er mer eller mindre på plass. Gebyrsystemet skal designes ut fra invoice API'et, gebyret i seg selv skal splittes mellom donør og tingoteket. Prosjekteier vil også at splittingen av gebyret skal kunne styres av administrator. Hvis ikke invoice API'et tillater oppretting og splitting av gebyr kan det hende at man må opprette et depositum system i stedet. Prosjektet har også som et mål å implementere scanning av nfc tags, dette skal brukes på tingene i tingoteket. Tanken er at man skal registrere et produkt i systemet, printe det på en en nfc tag og så klistre den på tingen. Senere skal nfc taggen brukes for utlåning og retur av tingen. Tanken bak implementasjonen er at nfc id'en skal settes som identifikator med gjeldende ting i databasen. Testing av nfc modulen kan bli en utfordring siden vi mangler nfc tags, med det kan sikkert håndteres med mockups. For vipps innloggingen prøvde man først å finne et bibliotek for å gjøre oAuth2 prosessen enklere. Dette førte med seg en del problemer og tap av tid, til slutt ble Axios brukt i stedet. Tanken med å bruke et bibliotek var å koble den inn i nøkkel vekslingsbiten av oAuth2 protokollen. Problemene vi møtte på var at bibliotekene enten krevde å bli brukt gjennom hele prosessen, eller at de ikke tillot nok konfigurasjon. Dette førte til at man konstruerte oAuth2 kall med Axios biblioteket, noe som fungerer greit men som tar litt tid.

I front-end delen så er nesten alt ferdig. Det har ikke vært så store problemer med å lage sidene ut fra Figma diagrammet. Den største utfordringen fremover blir å sette sammen Frontend og Backend, som er de to delene prosjektet er delt opp i. Det blir interessant å se hvor fort man klarer å få alt til å fungere, men målet er å bli ferdig så fort som mulig slik at systemet kan bli brukertestet. Det finnes noen andre krav som mangler i prosjektet slik som

kamera funksjonalitet og NFC lesing, men vi forventer ikke å se store problemer i implementeringen av dem.

8 Referanser

[1] Ionic, *Ionic Framework*, 2020. Hentet fra:

https://ionicframework.com/docs. Lastet ned 20.05.2020

[2] Angular, *Angular docs*, 2020. Hentet fra:

https://angular.io/docs. Lastet ned 20.05.2020

[3] Figma: *Figma design*, 2020. Hentet fra:

https://www.figma.com/design/. Lastet ned 20.05.2020

[4] Internet Engineering Task Force (IETF), *The OAuth 2.0 Authorization Framework,* 2012. Hentet fra:

https://tools.ietf.org/html/rfc6749. Lastet ned 20.05.2020

[5] Devdocs, Javascript docs, 2020. Hentet fra:

https://devdocs.io/javascript/. Lastet ned 20.05.2020

[6] Devdocs, Html docs, 2020. Hentet fra:

https://devdocs.io/html/. Lastet ned 20.05.2020

[7] OpenJS Foundation, *About Node.js,* 2020. Hentet fra:

https://nodejs.org/en/about/. Lastet ned: 20.05.2020.

[8] OpenJS Foundation, *Hello world example,* 2020. Hentet fra:

https://expressjs.com/en/starter/hello-world.html. Lastet ned: 20.05.2020.

[9] RESTfulAPI.net, *What is REST*, 2018. Hentet fra:

https://restfulapi.net/. Lastet ned: 20.05.2020.

[10] The PostgreSQL Global Development Group, *About*, 2020. Hentet fra: https://www.postgresql.org/about/. Lastet ned: 20.05.2020.

[11] Postman Inc, *The Collaboration Platform for API Development*, 2020. Hentet fra: https://www.postman.com/. Lastet ned: 20.05.2020.

[12] Facebook Inc, *Delightful JavaScript Testing*, 2020. Hentet fra: https://jestjs.io/. Lastet ned: 20.05.2020.

[13] Microsoft, *TypeScript - JavaScript that scales,* 2020.Hentet fra: https://www.typescriptlang.org/. Lastet ned: 20.05.2020.

[14] Auth0, *JSON Web Token Introduction*, 2020. Hentet fra: https://jwt.io/introduction/. Lastet ned: 20.05.2020.

[15] Vipps AS, *Vipps Login API*, 2020. Hentet fra: https://www.vipps.no/developers-documentation/login/documentation. Lastet ned: 20.05.2020.

[16] Vipps AS, Vipps *Invoice API*, 2020. Hentet fra: https://www.vipps.no/developers-documentation/invoice/introduction/. Lastet ned: 20.05.2020.

[17] Google, *Firebase,* 2020. Hentet fra: https://firebase.google.com/. Lastet ned: 20.05.2020.

[18] Microsoft, *Visual Studio Code - Code Editing. Redefined*, 2020. Hentet fra: https://code.visualstudio.com/. Lastet ned: 20.05.2020.

[19] Sass, *Sass: Syntax*, 2020. Hentet fra: https://sass-lang.com/documentation/syntax. Lastet ned: 27.05.2020.

[20] GitHub, Inc., *How developers work,* 2020. Hentet fra: https://github.com/features. Laster ned: 27.05.2020.

[21] Google, *Alle filene dine - tilgjengelige der du er,* 2020. Hentet fra:

https://www.google.com/intl/no_ALL/drive/. Lastet ned: 27.05.2020.

[22] StrongLoop, IBM and other express.js contributors, *Express routing,* 2017. Hentet fra:

https://expressjs.com/en/guide/routing.html. Lastet ned: 28.05.2020.

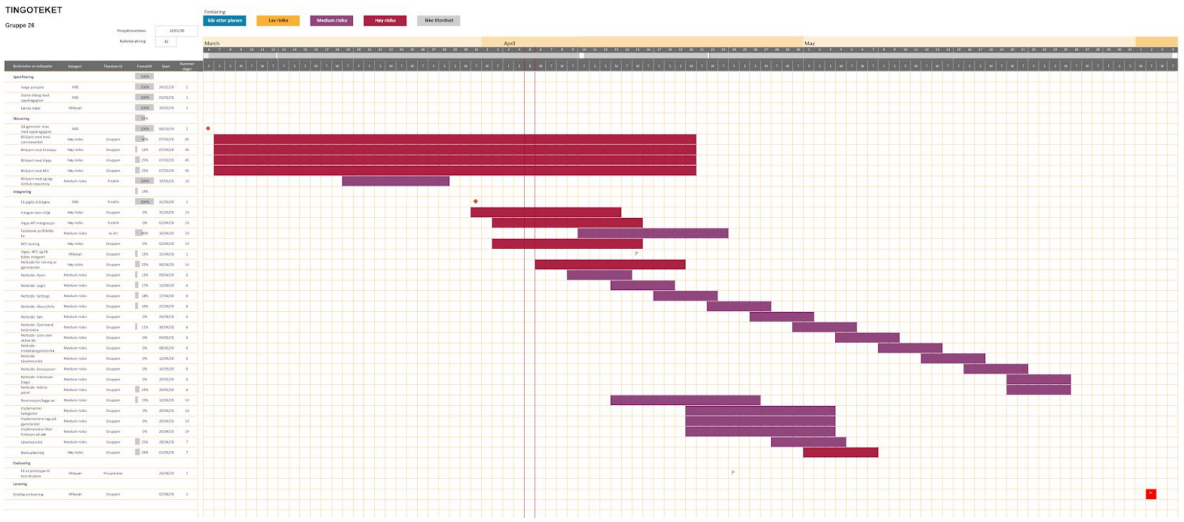
[23] Kroll, P. and Kruchten, P., 2003. Rational Unified Process Made Easy: A Practitioner's Guide To The RUP, The. Addison-Wesley Professional, pp.50-53.

[24] Leahy, S., 2018. [online] https://www.nationalgeographic.com/news/2018/05/zero-waste-families-plastic-culture/. Available at: <https://www.nationalgeographic.com/news/2018/05/zero-waste-families-plastic-culture/> [Accessed 28 May 2020].

[24] Montegriffo, N., 2020. *What Is NFC And Why Should I Use It? | Androidpit*. [online] AndroidPIT. Available at: <https://www.androidpit.com/what-is-nfc> [Accessed 1 June 2020].

[25] Jasmine.github.io. 2020. *Jasmine Documentation Home*. [online] Available at: <https://jasmine.github.io/pages/docs_home.html> [Accessed 1 June 2020].

9 Appendix



Risiko	Sannsynlighet	Innvirkningsgrad	Alvorlighet	Håndtering
Utilstrekkelig kunnskap om GitHub.	4	3	12	Vi må lære oss GitHub slik at vi enklere kan få overført prosjektet på maskinene våre, og for å enklere kunne samarbeide på prosjektet.
Prosjektet vil ikke kjøre på maskinene våre.	4	5	20	Vi må så fort som mulig få prosjektet til å kjøre på maskinene våre.

Utilstrekkelig kunnskap om Ionic Rammeverket.	5	4	20	Vi må sette oss inn i rammeverket slik at vi kan bygge videre på prosjektet.
Utilstrekkelig kunnskap om Vipps API'et.	5	4	25	Ansvarlige må lære seg Vipps API'et.
Utilstrekkelig kunnskap om NFC teknologi.	5	5	25	Ansvarlige for NFC-implementasjon må lære seg om teknologien.
Utilstrekkelig kunnskap om Firebase.	5	5	25	Vi må lære oss om Firebase plattformen og hvordan den skal brukes.
At vi ikke vet hvilke bibliotek vi skal bruke for unit-testing av programmet.	5	3	15	Vi må bli enig om bibliotek(er) som skal brukes for unit-testing.
At vi ikke får brukertestet prosjektet.	2	5	10	Vi kan spørre prosjekt-eier om hvordan dette kan løses, eventuelt spørre venner/familie om hvordan brukeropplevelsen er.

Uklar arkitektur i prosjektet.	2	3	6	Vi ser på hvilken arkitektur som allerede er brukt i original-prosjekt et, hvis ikke det fungerer kan utviklerne ha et møte og bli enig om en.
Uklart hvilke funksjoner som skal være med i prosjektet.	4	4	16	Ha jevne møter med oppdragsgiver og lage relevante diagrammer for å oppdage «gjemte funksjoner».
Utilstrekkelig kunnskap om versjoneringsverktøy et GIT.	4	4	16	Om det skulle oppstå problemer burde vi kunne bruke GIT for å gå tilbake, vi må bare lære oss å bruke det.