SBANKEN'S
MINE KVITTERINGER

Bachelor, Dataingeniør

Institutt for data- og realfag

Fakultet for ingeniør- og naturvitskap

INNLEVERINGSDATO: 02.06.19

ANTALL ORD: 7327

JOCEF IVERSEN OG PEDER WIIG AALEN

Høgskulen på Vestlandet, avdeling Bergen, institutt for data- og realfag
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

Høgskulen på Vestlandet

Fakultet for ingeniør- og naturvitskap
Institutt for data- og realfag

TITTELSIDE FOR HOVEDPROSJEKT

Tittel:	Mine Kvitteringer
Dato:	02.06.19
Forfattere:	Peder Aalen
	Jocef Iversen
Studieretning:	Dataingeniør
Veileder:	Svein-Ivar Lillehaug
Oppdragsgiver:	Sbanken
Kontaktperson:	Andreas Øye
Nøkkelord:	ReactJS, C#, .NET Core
Antall sider:	35
Gradering:	Ingen

Høgskulen på Vestlandet, avdeling Bergen, institutt for data- og realfag
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 er skrevet av Jocef Iversen og Peder Wiig Aalen som hovedprosjekt i DAT190 ved HVL. Oppgaven gikk ut på å lage en applikasjon for Sbanken som lar en kunde ta bilde av en kvittering og ekstraherer ut verdifull informasjon for så å lagre det i en skytjeneste.

Vi vil takke Lars Nestås og Andreas Øye i Sbanken som tildelte oss oppgaven. Vi vil også takke Svein-Ivar Lillehaug for gode tilbakemeldinger under arbeidet med rapporten.

Innholdsfortegnelse

1 Introduksjon

1.1 Mål og motivasjon

Sbanken, tidligere Skandiabanken, holder til i Folke Bernadottes vei 38 i Fyllingsdalen. Banken åpnet i april 2000 som Norges første heldigitale bank (nettbank) uten filialer. Banken separerte seg fra sitt svenske eierkonsern høsten 2015 og skiftet navn til Sbanken høsten 2017. Hoved kontoret har hele tiden vært i Fyllingsdalen. I dag har selskapet over 400 medarbeidere og 400 tusen kunder og en forvaltningskapital på over 400 millioner kroner.

For Sbanken står innovasjon sentralt, og de ønsker å være den banken som tilbyr de beste og enkleste løsningene til sine kunder. Sbanken ønsker nå å erstatte fysiske kvitteringer med en digital utgave som gjør det enklere for kunden å holde orden på økonomien sin.

Målet for dette prosjektet er å utvikle denne løsningen, som digitaliserer kvitteringer og lagrer det som et bilag til en transaksjon. Ved å koble kundens kvittering opp mot en transaksjon, og ekstrahere ut den detaljerte informasjonen i kvittering, vil kunden ha mer verdifull data om transaksjonen enn hva som er tilgjengelig per i dag.

Banken har et mål om å gå over til skyløsninger, ved å bruke tjenestene Azure Cognitive Services og Azure Storage fra Microsoft. Gjennom denne plattformen vil det være mulig å både ekstrahere teksten fra en kvittering og lagre den i en database.

Det er i dag et høyt fokus på ny teknologi i banksektoren. Følgelig gjelder det å være innovativ og lage de beste og enkleste løsningene for kundene. Ved å ta i bruk teknologiene som Azure tilbyr, vil Sbanken ha et fortrinn når kunden skal velge bank. Det vil også gi banken god innsikt, kunnskap og lærdom om hvordan det er å ta i bruk Cloud-services.

1.2 Kontekst

Det skal utvikles en løsning som gir kundene i Sbanken en mulighet til å se kvitteringene sine og hvilken transaksjon de hører til. Kunden skal kunne laste opp et bilde av en kvittering og få informasjon som hva beløp, dato og brukersted var.

Bakgrunnen for dette produktet er at en kunde skal ha mulighet til å lagre sine papirkvitteringer digitalt. Ved å ha en digital lagringsplass for kvitteringer vil det bli enklere for en kunde å finne igjen kvitteringene sine kontra å lagre de fysisk.

1.3 Begrensninger

For å kunne tolke bildet som kunden tar av kvitteringen er vi avhengig av at kunden tar et klart bilde. Dersom bildet er av lav kvalitet vil OCR (Optical Character Recognition) gi feil tolkning av hva som står på kvitteringen. Vi er også avhengig av at Microsoft OCR gir tilbake tilfredsstillende resultat. Dette er et verktøy som vi ikke kan endre på selv.

Programvaren skal leve innenfor bankens interne systemer. Dette gir oss et sett med regler og systemer som vi må tilpasse oss til. Sikkerhetsprotokollen til Sbanken gir oss ikke mulighet til å jobbe med prosjektet via andre steder enn i banken.

Vi kan heller ikke vite om det er en faktisk kvittering det blir tatt bilde av. Dette kan løses gjennom en videreutvikling av bachelor-oppgaven.

1.4 Ressurser

Sbanken har en egen «wikipedia» hvor kunnskap og guider blir lagret. Her vil det bli opprettet en egen side for prosjektet, hvor veilederen og prosjektlederen vår kan legge ut informasjon og ressurser om prosjektet. På wikien er det mye tilgjengelige ressurser om nettbanken og andre verktøy. Dette vil komme til god hjelp for vårt prosjekt.

Ansatte i banken er veldig behjelpelig og stiller med råd når vi har behov for hjelp. De vil være en god ressurs for oss.

Ettersom vi skal bruke to tjenester fra Azure som ikke er så mye brukt i banken pr i dag vil vi bruke Microsoft sin egen dokumentasjon som en ressurs.

1.5 Organisering av rapporten

Kapittel 1:

Presentasjon av målet og motivasjonen for oppgaven. Hva er konteksten og om løsningen er relevant.

Kapittel 2:

Beskriver hvorfor prosjektet er viktig for eieren av prosjektet og beskriver tidligere arbeid som er relevant. Forklarer kravspesifikasjonen og ønsket løsning fra Sbanken, og punkterer samt litteratur som er blitt brukt.

Kapittel 3:

Oversikt over mulige fremgangsmetoder for å utvikle produktet, samt hva vi har valgt og hvorfor. Diskuterer også nødvendige ressurser, risiko og presenter hvordan resultatet blir evaluert i slutten av prosjektet.

Kapittel 4:

Presenterer prosjektets arkitekturdesign og utvikling av applikasjonen.

Kapittel 5:

Inneholder en evaluering av applikasjonen med resultat fra evalueringen.

Kapittel 6:

Diskuterer hvordan vi løste oppgaven og hvilke konsekvenser det fikk, samt en konklusjon og hvordan veien videre blir.

Kapittel 7:

Appendix med risikoliste og gannt-diagram.

Kapittel 8:

Referanser.

2 Prosjektbeskrivelse

2.1 Praktisk bakgrunn

Sbanken har lenge ønsket å ta i bruk tegngjenkjenningsteknologier for å gjøre bankopplevelsen for kundene enklere. Digital skanning av e-faktura begynner å bli dagligdags i bankmiljøet, mens en løsning for en oversikt over digitale kvittering er ikke det enda. Denne løsningen er noe de har ønsket å implementere, men de har til nå manglet ressurser for slik utvikling. Da vi fortalte dem at vi var interessert i å gjøre et bachelor prosjekt hos dem, ble ideen trukket frem.

2.1.1 Prosjektets eier

Sbanken står som prosjekteier av applikasjonen vi utvikler. Når applikasjonen er ferdig, skal den leve som en egen side i nettbanken.
Sbanken har en egen avdeling som heter Konseptutvikling hvor de jobber med å lage nye konsept som kan bidra til å gjøre banken bedre og enklere for kundene. Ved Konseptutvikling har de kommet opp med ideen å bruke optisk tegngjenkjenning til å ekstrahere tekst fra en kvittering, og lagre den som en digital utgave. Dette er en av flere ideer som de selv ikke har hatt ressurser for å prioritere. Derfor var det en god mulighet for Sbanken å tildele oss denne oppgaven med å utvikle produktet gjennom et bachelorprosjekt for dem. Ved å tilby kundene et produkt som leser av den viktigste informasjonen i kvitteringen, vil de ha et nytt interessant produkt som de kan tilby kunder. Dette vil opprettholde Sbankens image om å være en utfordrer med de mest fornøyde kundene.

2.1.2 Tidligere arbeid

Sbanken har et lignende prosjekt hvor de bruker Azure Cognitive Service for å lese e-faktura. Dette blir utviklet av Oscar Kvamme som jobber i banken. Vi har sett på muligheten for å utvikle videre på dette prosjektet, men det var benyttet en eldre versjon av OCR slik at det ikke var mulig for oss å bygge videre på det. Kvamme hadde derimot skrevet en implementasjon av et API (Application programming interface) for e-faktura, som vi kunne videreutvikle. Derfor var det ikke nødvendig for oss å opprette et nytt API i banken, for prosjektet.

Konseptutvikling utvikler et produkt for å kategorisere transaksjoner. Dette er en masteroppgave som skrives av Fredrik Absalonsen for Sbanken. Produktet kan brukes på vår løsning som ressurs for å finne brukerstedet til en transaksjon.

2.1.3 Innledende kravspesifikasjon

De første kravene er hentet fra kravspesifikasjonen gitt av konseptutvikling hos Sbanken.

Disse er formulert gjennom to brukerhistorier som forklarer hva som er ønsket:

- Som kunde skal jeg kunne ta bilde av en kvittering og laste den opp i nettbanken
- Som kunde skal jeg finne igjen kvitteringer jeg har tatt bilde av i nettbanken.

De to brukerhistoriene har flere akseptansekrav som må være oppfylt for at brukerhistorien skal være godkjent.

Som kunde skal jeg kunne ta bilde av en kvittering og laste den opp i nettbanken:

- Jeg skal kunne trykke på en lenke på siden "Mine kvitteringer" som lar meg ta/laste opp et bilde av en kvittering.
- Jeg skal kunne legge til kommentar før jeg laster opp bildet.
- Jeg skal få en tilbakemelding på at jeg har tatt bilde av kvittering og at feltene er tolket og kan gjøre endringer i tolket tekst om nødvendig.
- Jeg skal få en tilbakemelding når bildet ikke kan tolkes eller aksepteres og hvorfor den ikke gjør det.

Som kunde skal jeg finne igjen kvitteringer jeg har tatt bilde av i nettbanken:

- Som kunde skal jeg kunne finne igjen bilder av kvitteringer på transaksjoner jeg har betalt i nettbanken.
- Som kunde skal jeg kunne se kvitteringen tilkoblet transaksjonen i kontoutskriften.
- Jeg skal kunne trykke på "Slett kvittering" som permanent sletter bildet og teksten.
- Jeg skal kunne filtrere/sortere listen av kvitteringer.

2.1.4 Innledende løsning

Banken ønsker at vi skal la kunden ta bilde av en kvittering. Den skal lastes opp til en side i banken hvor vi skal hente ut informasjon ved hjelp av Microsoft Azure Cognitive Service. Videre skal vi få tilbake data fra Cognitive Service og klare å lese av sum, dato og hva som ble handlet. Dersom dataen vi henter ut er feil, skal kunden kunne endre dette selv. Når kunde godkjenner at disse verdiene stemmer, lagrer vi disse verdiene sammen med bildet som tilleggsdata (metadata). Så laster vi alt opp i Microsoft Azure Cloud Storage og dermed vil kvitteringen ligge tilgjengelig for kunden ved senere bruk. Kunden skal ha en oversikt over alle kvitteringene som er lagret.

2.2 Litteratur

Vi benytter oss av Microsoft Azure sin dokumentasjon for Cognitive services og Cloud Storage. Siden vi bruker .Net vil vi benytte oss av all dokumentasjon for .Net og C#.

Klientside-programmeringen blir gjort i JavaScript og React-biblioteket. Her bruker vi Mozilla Developer Network for JavaScript dokumentasjon. Facebook har laget dokumentasjon for React som vi benytter oss av. Dersom vi har veldig spesifikke utfordringer vil vi bruke Google, stackoverflow og artikler på nett.

3 PROSJEKTDESIGN

3.1 Mulige løsninger

3.1.1 Løsning 1

En måte å lage applikasjonen på er å hente inn OCR-resultatet i back-end, for så å sende det videre til kunden. Før kunden får presentert verdiene vil vi ekstrahere sum, dato og tekst i nettleseren. Dette kan vi gjøre ved å utvikle logikken for ekstrahering på klientsiden.

Når kunden får presentert verdiene som er ekstrahert kan hun godkjenne eller endre verdiene dersom vi presenterte feiltolket informasjon.

Kunden godkjenner verdiene og det blir sendt til backend hvor vi legger sammen dataen og bildet av kvitteringen i en blob. Denne lagres så i Azure Cloud Storage.

Dersom en kunde vil se alle kvitteringene går hun inn på siden “mine kvitteringer” og vi henter inn alle kvitteringene som er lagret for den kunden.

3.1.2 Løsning 2

I alternativ 1 gjør vi logikk på klientsiden. En alternativ løsning er å ha all logikk i back-end. Da vil vi hente ut informasjonen fra kvitteringen i back-end når vi får resultatet fra Cognitive Service. Deretter kan vi videresende den ekstraherte teksten til klientsiden hvor den blir presentert for kunde. Kunden kan da godkjenne eller endre resultatet derfra om noe er feil. På denne måten vil det rettes en forespørsel til serveren vår, en til OCR, en til klientsiden og så en tilbake til serveren.

3.1.3 Løsning 3

Det er mulig å lage en applikasjon som bruker maskinlæring for å trene opp applikasjonen til å finne de korrekte verdiene. For å få dette til må vi hente inn et stort antall kvitteringer og lage et treningssett. Vi må da legge inn korrekte verdier på alle kvitteringene i treningssettet og bruke en maskinlæringsalgoritme for å gjenkjenne verdiene ut fra bilde klienten tar av kvitteringen.

3.1.4 Løsning 4

En alternativ nedskalering vil være å kutte ut OCR tjenesten som gir oss teksten som står på et bilde. Da må kunden manuelt skrive inn verdiene som står på kvitteringen. Dette er et godt minste brukbart produkt siden det gir oss muligheten til å fokusere på å utvikle et godt

brukergrensesnitt, og deretter kan vi videreutvikle løsningen for å lagre å hente bildene kunden laster opp.

3.2.5 Vurdering av løsningsforslag

Det er fordeler og ulemper med de forskjellige løsningsforslagene. I denne seksjonen vil vi drøfte om de ulike.

I alternativ en vil ekstrahering av teksten fra kvitteringen skje på klientsiden etter å ha fått resultatet tilbake fra OCR. Ulempen med dette er at mye av logikken blir prosessert på klientsiden, dvs. at kundens maskin utfører logikken, og ikke webserveren. På den andre siden vil det ikke ha så stor betydning siden ekstraheringslogikken ikke er så stor, men etter hvert som prosjektet vokser kan det være en idé å gjøre dette på serversiden. I alternativ to vil prosesseringen utføres på backend av programvaren. Da vil forretningslaget være skjult for brukeren og kalkuleringsoperasjonene vil skje der. Dette vil gjøre at vi får en lavere kobling mellom front og backend.

Oppdragsgiver har uttrykt et ønske om alt skal skje automatisk slik at kunden ikke skal skrive inn noe manuelt. Det skal bare være et klikk og så er alt gjort. Følgelig har vi valgt å se bort fra alternativ fire. Denne løsningen ville vært en god base for å videreutvikle prosjektet senere, men trolig ville ikke en slik løsning fått positiv respons fra sluttkunden.

Dersom vi skulle brukt maskinlæring ville det vært nødvendig å samle inn hundrevis av kvitteringer, laget treningssett og testsett, og således utviklet et nettverk som kan prosessere informasjonen i settene. Dette ville i seg selv vært en egen bacheloroppgave. Vi mener Cognitive Service gir et godt resultat ut fra vår testing og tror det skal være godt nok til å kunne hente ut den informasjon vi har behov for, fra kvitteringene. Følgelig kan vi utelukke alternativ tre for nå.

Etterhvert som kunder tar i bruk produktet og vi får samlet inn data vil vi kunne gjøre dypere analyser av resultatene til applikasjonen vår. Resultatene kan hjelpe oss å finne nye mønstre i kvitteringer som vi ikke har sett på før.

Konklusjon:

Løsning to er et veldig godt alternativ og det stod mellom det og løsning en. Vi valgte å gå for alternativ en hvor vi gjør det meste av operasjonene i frontend. Dette er i hovedsak fordi vi er mest kjent med programmeringsspråket JavaScript med biblioteket React, og vi føler at dette vil gi et godt utgangspunkt for løsningen.

3.2 Spesifikasjon

I møte med Sbanken ble det uttrykt et ønske om å lagre dataene i skyen. Sbanken ville at vi skulle bruke Microsoft Azure Cloud til lagring.

Microsoft har et produkt for optisk tegngjenkjenning I Microsoft Azure Cognitive Services. Dette produktet ga veldig gode resultater når vi testet demoen, og dette blir også vårt valg når vi skal utvikle.

For å lage logikk i Backend bruker banken C# og .Net. For å gjøre det enkelt for oss når vi skal utvikle vil vi holde oss til Sbanken sine valg av programmeringsspråk.

Til klientsideprogrammering har Sbanken bestemt at all nyutvikling skal kodes i ReactJs.

3.3 Valg av verktøy og programmeringsspråk

3.3.1 Front – og Backend

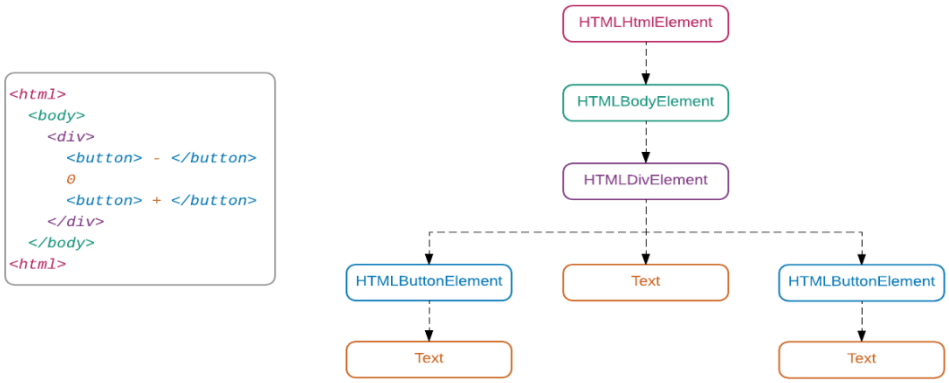
Hoveddelen av koden er skrevet i JavaScript med ReactJs. Dette er den delen av applikasjonen som ligger nærmest brukeren, koden du visuelt ser på skjermen - altså frontend.

Vi skriver et API i C# .NET Core. Dette er den delen av programvaren som ligger nærmest databasen Azure Storage, og Azure Cognitive Service. Dette er backenden vår. Den kommuniserer med Azure sine servicer og gjør disse tilgjengelig for frontenden vår. API'et vårt består av et sikkerhetslag og forskjellige operasjoner som å lagre og hente kvitteringer, samt tegngjenkjenningstjenesten fra Azure.

3.3.2 Kort om ReactJs

React er et Javascript bibliotek skrevet og vedlikeholdt av Facebook. React gjør det mulig for en utvikler og skrive komponentbasert kode. Å dele opp koden i komponenter gir utvikleren fleksibilitet og gjør det lettere å vedlikeholde. DRY – Dont Repeat Yourself, er et kjent prinsipp innen programmering. Ved å utvikle komponenter reduserer en duplisert kode.

React har mange fordeler og få ulemper. En av de største, om ikke den største, er at React tar en kopi av DOM'en (Document Object Model) til nettsiden og oppretter en virtuell kopi. I vanlig Javascript vil hele DOM'en bli lastet inn på nytt om en endrer på et element. Det React gjør er å sammenligne endringer i kopien mot den originale, og kun bytte ut det som er endret. En analogi er at det er lettere å flytte på møbler i en plantegning enn å flytte de fysisk. På denne måten har React en høy ytelse.



Figur 3.1 viser HTML DOM treet.

Se for deg at du skal endre på nederste node til venstre i HTML DOM treet. I vanlig Javascript blir alt rendert (skapt) på nytt, mens i React blir kun den aktuelle noden byttet ut. I større applikasjoner kan disse trærne bli gigantiske. Da er det virkelig et løft for ytelsen med React.

3.3.3 Liste av verktøy, rammeverk og programmeringsspråk

- Git og GitBucket – versjonshånderingsverktøy
- Asana – digitalt prosjektverktøy
- Visual Studio and Visual Studio Code
- Azure Cognitive Services
- Azure Cloud Storage
- MermaidJS
- Postman
- Jest
- JavaScript og ReactJs
- C# .Net

3.4 Prosjektets utviklingsmetode

3.4.1 Utviklingsmetode

For å få en god arbeidsflyt bruker vi Asana som verktøy for agil prosjekt administrasjon og Git som versjonshåndtering. Asana er et godt verktøy for å oversikt over forskjellige oppgaver og hvem som arbeider med hva. Git bruker vi som et verktøy for å dele felles kildekode og holde orden på riktig versjon av kode. Vi bruker GitBucket som en lagringskontainer for Git. Her kan vi hente ned den nyeste kildekoden og se på endringer som er blitt gjort.

En oppgave begynner med å bli opprettet og beskrevet i Asana. Da oppgaven er opprettet kan en tildele den en tidsfrist og en utvikler. Utvikleren vil da kunne se hvilke oppgaver hun er tildelt og kunne starte på disse. Dersom en oppgave er for stor kan den bli delt inn i flere delaktiviteter med en bredere beskrivelse. Dette blir gjort slik at en utvikler kan fokusere på en del av prosjektet av gangen.

Når utvikleren er ferdig med en oppgave laster (pusher) hun opp koden på GitBucket. Da kan en annen utvikler ta et såkalt code-review (kode-gjennomgang) av hva som er blitt gjennomført. Det er lett å gjøre feil og derfor er det smart å la andre programmerere sjekke ut ens kode. Når code-review er godkjent blir det foretatt en sammenfletting av kodesnutten og hovedkodebasen. Da vil dette bli den nye versjonen av koden.

Testing blir gjort etter at code-review er ferdig. Dette kan gjøre på forskjellige måter. Vi skriver både enhetstester og foretar manuell testing av koden for å sjekke at alt fungerer som det skal.

3.4.2 Prosjekt plan

Prosjektplan ligger som vedlegg

3.4.3 Risikohåndtering

Det finnes regler for hva Sbanken har lov å lagre av opplysninger. Juridisk avdeling i banken må finne ut hvilke data vi har lov å lagre i Azure Cloud. Lagring av kundenummer i skyen kan vi ikke gjøre før banken har tatt en intern risikovurdering. Dette kan ta en stund, og vi har behov for å vite hvilket bilde som tilhører en kunde. Derfor har vi valgt å generere en ny id for en kunde, og den kan da lagres i skyen. Dette er ekstra arbeid og er blant en av risikoene i den laveste klassen.

Designet for produktet er ikke 100% ferdig. Mot ferdigstillelse av prosjektet kan det være for liten tid til å utforme designet slik det er ønsket.

For å utvikle vår applikasjon vil vi gjøre bruk av enkelte nye teknologier som Sbanken enda ikke har tatt i vanlig bruk. I hovedsak Azure Cloud og OCR. For oss er det også mye kunnskap vi må tilegne oss på kort tid. Det er en risiko for at prosjektet er for stort for tiden vi har til bruk.

3.5 Evalueringsmetode

Måloppnående vil være at vi klarer å oppfylle alle akseptansekravene i brukerhistoriene, gitt i kapittel 2. Vi ble enige om å ha en fungerende pilot ferdig til 1.mai. Denne skal bli brukt og testet internt av ansatte i banken. I møte med intern veileder ble vi enige om at 90% korrekt lesing av data fra kvittering ville være akseptabelt for vår pilot. Det var også ønsket at tiden det tar å tolke teksten ikke skal ta for lang tid.

Vi kan dele kravene i funksjonelle krav og ikke-funksjonelle krav. Akseptansekravene er de funksjonelle kravene og ønsket om 90% korrekt tolkning og kort tid på tolkning er de ikke-funksjonelle kravene.

4 Design og utvikling

Dette kapittelet vil forklare designet og arkitekturen til prosjektoppgaven.

4.1 Design

Vi har valgt å dele designet av oppgaven opp i to deler. Det ene er designet av arkitekturen og den andre delen er det visuelle designet for klienten. Vi fikk selv bestemme arkitekturen for applikasjonen så lenge vi brukte de verktøyene som var bestemt av Sbanken. Som tidligere nevnt skulle vi bruke Microsoft Azure Cloud, Microsoft Azure Cognitive Services (OCR), ReactJs og .Net-rammeverket. I designet av arkitekturen skulle vi knytte alle disse verktøyene og programmene sammen til en applikasjon. Vi skulle lage en applikasjon som var mest mulig effektiv med tanke på lagring av data. En kunde skal også slippe å vente lenge når denne bruker applikasjonen. Bak designet ligger brukerhistoriene som tidligere er nevnt i oppgaven. Underveis i prosjektet ble vi også enige med intern veileder om å gå bort fra kravet om å koble kvitteringen til en transaksjon.

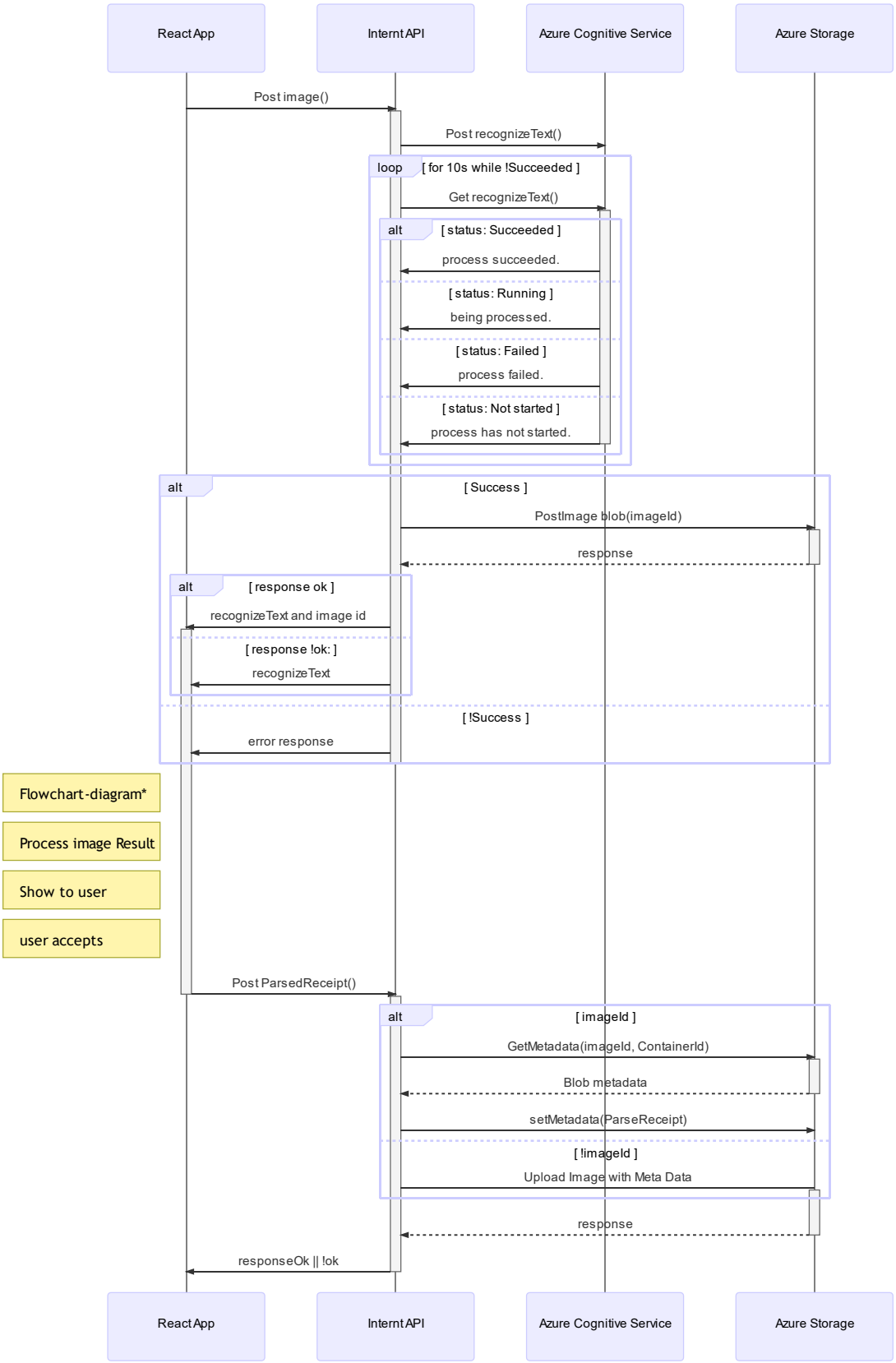
4.1.2 Arkitektur

Vi valgte å gå for løsning 1. Applikasjonen vår skal inneholde disse delene:

- Applikasjonsflyt
- Klientside (ReactJS)
- Tekstekstrahering (ReactJS)
- Backend-logikk (.Net/C#)
- Optisk tekstgjenkjenning (Microsoft Azure Cognitive Services)
- Skylagring (Microsoft Azure Blob Storage)

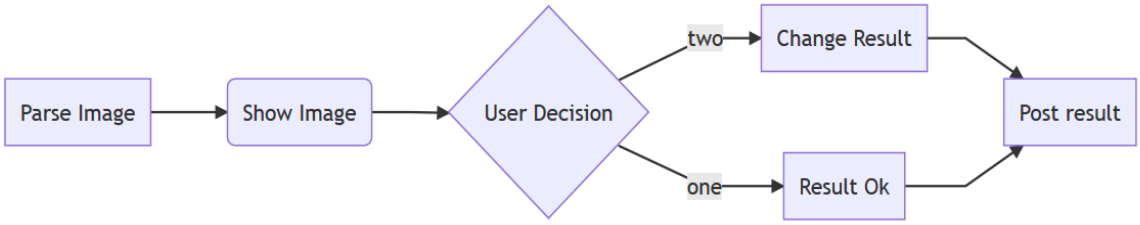
4.1.2.1 Applikasjonsflyt

Figur 4.1 viser flyten som skjer når et bilde blir tatt av brukeren. Det blir sendt en forespørsel til tjeneren, vårt (API). Tjeneren spør deretter OCR om teksten som står på bildet. OCR sender tilbake en lenke hvor bildet kan bli hentet etter at det er ferdig prosessert. Deretter sendes det en ny forespørsel fra API ’et til denne lenken, tjeneren svarer med en av fire forskjellige statuskoder basert på om teksten er klar.



Figur 4.1 Flyt når en bruker laster opp en kvittering.

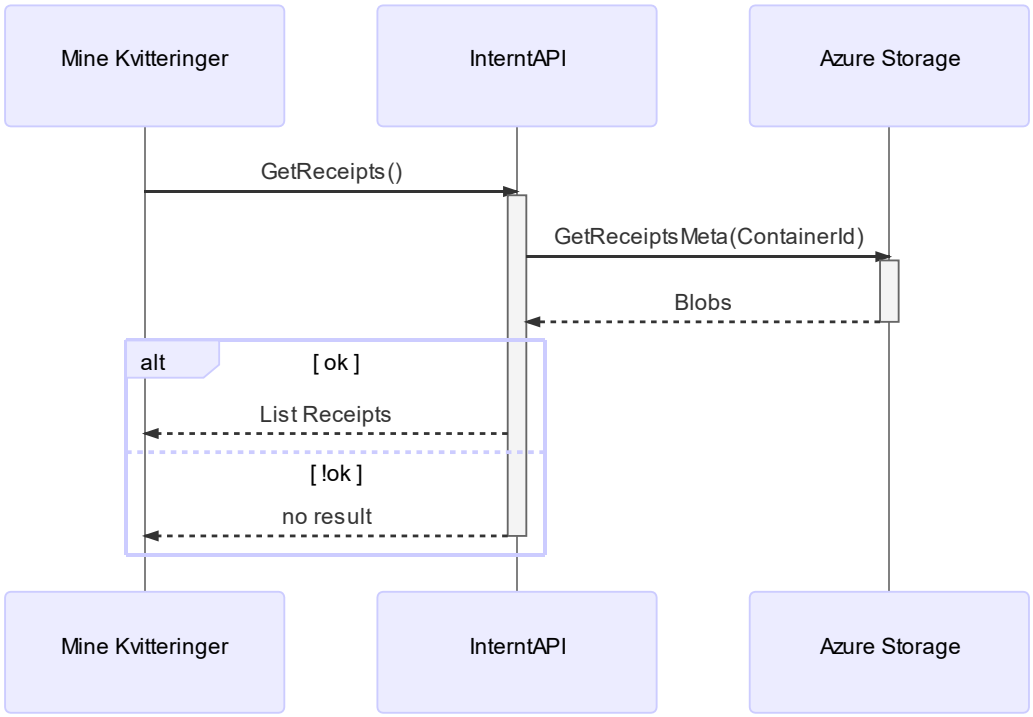
Videre sendes teksten tilbake fra API ’et til klientsiden hvor brukeren blir vist resultatet. Da har bruker mulighet til å endre data om dette viser seg å være feil. Ref. Flowchart diagram*.



Figur 4.2 Kunden blir presentert med teksten fra kvitteringen og kan velge å endre resultatet.

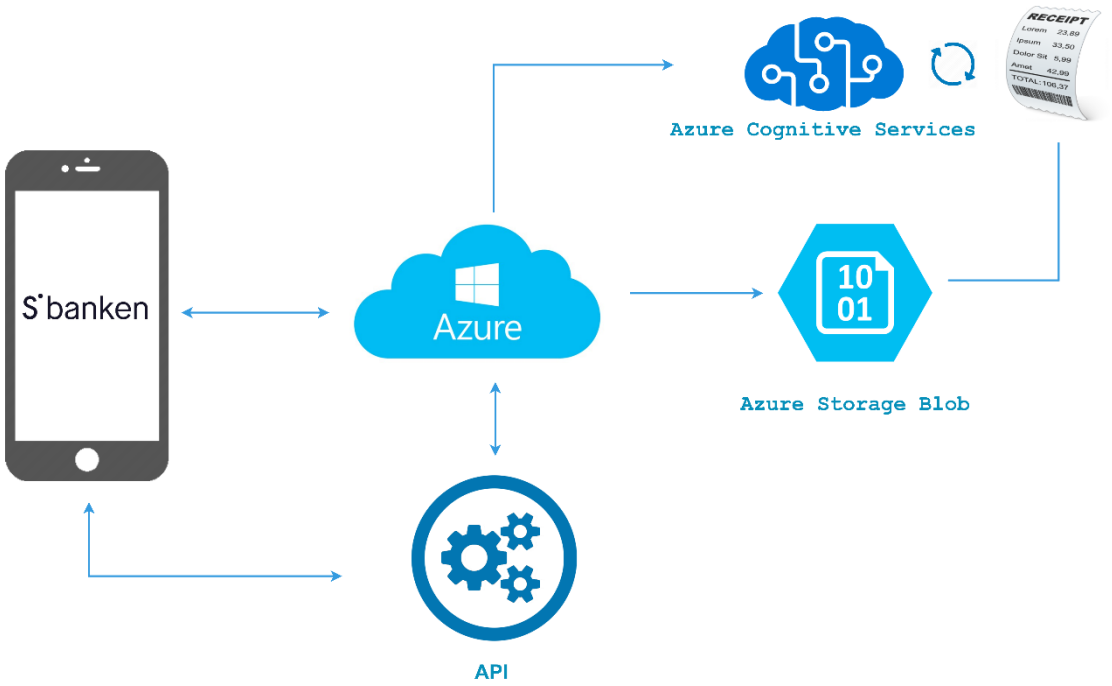
Når bruker har tatt sin beslutning, sendes det ny forespørsel til API’et som håndterer forespørselen ved å hente bildet fra skyen og legge til metadataen brukeren har godkjent.

Deretter sendes en forespørsel fra API 'et til klientsiden med melding om at alt er klart, og brukeren sendes så tilbake til hovedsiden hvor alle kvitteringene vises.



Figur 4.3 Mine kvitteringer.

Figur 4.4 er en oversikt over arkitekturen til applikasjonen. Den viser en kommunikasjon mellom webtjeneren, vårt interne API og Azure tjenestene.



Figur 4.4 Mine kvitteringer arkitektur.

4.1.2.2 Klientside (ReactJS)

Klientsiden av applikasjonen hvor alt blir presentert er utviklet med JavaScript biblioteket ReactJS. Denne delen består av en visning av en hovedside og en visning av å ta bilde av en kvittering.

Fra hovedsiden har brukeren primært mulighet til å legge til, se, endre og slette en kvittering. Det skal også bli en mulighet til å søke og filtrere kvitteringer.

4.1.2.3 Optisk tekstgjenkjenning (Microsoft Azure Cognitive Services)

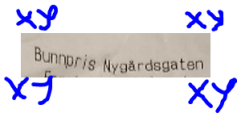
Vi bruker Azure sitt Optical Character Recognition (OCR) produkt for å lese av tekst fra bilde. Når kunden laster opp et bilde til banken vil det bli sendt videre til en Microsoft-server for å gjenkjenne tekst. Dette programmet sender tilbake et JSON objekt som gjør at vi har et dataobjekt hvor vi kan lete etter tekst vi er interessert i.

Figuren under viser JSON objektet fra OCR tjenesten.


```
"boundingBox": [
  1137,
  327,
  1666,
  360,
  1661,
  439,
  1132,
  406
],
"text": "Bunnpris Nygardsgaten",
"words": [
  {
    "boundingBox": [
      1145,
      331,
      1338,
      351,
      1334,
      419,
      1142,
      400
    ],
    "text": "Bunnpris"
```


Figur 4.5 Datarepresentasjon av resultatet sendt tilbake fra OCR.

OCR gjør en topp til bunn skanning av bildet og sender teksten tilbake som en tabell linje for linje. BoundingBox er på koordinat (XY, XY, XY, XY).



På denne måten er det en struktur i formatet og det gjør at lesing av dataen ikke er tilfeldig. Det er en gitt representasjon, slik at vi kan skrive algoritmer for å tolke teksten.

Den første versjonen vi prøvde ut av OCR ga oss tilfeldige inndelte soner av bildet, derfor kunne vi f.eks. (fra bildet over), få total og bank som en inndelt sone. Mot formodning kunne neste sone være en tilfeldig plass på kvitteringen, mens forventningen var 73.60. Dette ga oss en uforutsigbar struktur i dataen og derfor er det vanskelig å skille ut beløpet av varene.

Oppgaven ville vært desto mer utfordrende dersom vi hadde brukt versjon en av OCR. Azure har forøvrig oppdatert presisjonsresultatet fra tjenesten i versjon to. Det gjør at brukeren kan ta et bilde med lav oppløsning eller mindre lys og fortsatt få et akseptabelt resultat tilbake.

4.1.2.4 Tekstekstrahering (ReactJS)

Å hente ut tekst fra kvitteringen er enkelt fordi OCR gjør jobben for oss. Det vanskelige er å vite hva teksten representerer. Mønstre i kvitteringen viktig, men format og layout på kvitteringene er forskjellige og dette er det utsteder som bestemmer. Hvordan de vil presenterer sine kvitteringer er helt opp til dem.

Hva vi fant var at datoer og adresser ofte er presentert på likt format, men anomalier forekommer. Noen selskaper viser datoer representert med måned som navn og andre viser år først osv. Utsteders navn forekommer på ulike steder og er ikke alltid representert øverst på kvitteringen. I denne seksjonen vil vi forklare hvilke metoder vi har valgt for å utføre prosessen.

Teksten vi får fra OCR-tjenesten blir gått gjennom av en ekstraheringsalgoritmene som er blitt utviklet i klientsiden av applikasjonen.

For å ekstrahere dato og adresse bestemte vi oss for å konkatenere tabellen i JSON-format til en sammensatt streng av teksten. På den måten fikk vi en fullstendig setning og derved kunne vi implementere en tilstandsmaskin. Denne innehar regler for å lete etter tekst en er interessert i. f.eks. kan en si at en leter etter en tekst med formatet dd.mm.yyyy. Dette gjøres ved å utvikle en regex (tilstandsmaskin). Regex er en funksjon som generer output etter spesifiserte krav. Videre blir output validert og formatert, slik at brukeren får en leselig representasjon av dataen.

Brukersted er et felt vi ville ha med for å se hvilken butikk kvitteringen er utstedt fra. Kvitteringer som vi har sett på har inneholdt et organisasjonsnummer. Organisasjonsnummeret kan søkes opp i Brønnøysundregisteret, slik at vi kan få informasjon om brukerstedet derfra. Registeret har et åpent API som vi kan bruke for å hente dataen fra. Organisasjonsnummer har en matematisk egenskap, hvor det første sifferet alltid begynner på ni, er ni sifre langt, og det siste sifferet kan kalkuleres basert på de åtte foregående sifrene. Følgelig kan vi si med sikkerhet hvorvidt det eksisterer et organisasjonsnummer i en kvittering eller ikke.

En kvitterings sum (totalbeløp) illustreres på varierende måter, og den står ikke alltid på samme sted. Etter vår undersøkelse ser vi at summen ofte er skrevet med følgende ord: «Total», «Sum», «Totalt», «Sum x antall varer», «Å betale, beløp», «NOK», etterfølgt av beløpet i kroner. Dette er nøkkelord vi har med i algoritmen for å lete etter summen. For å

være mer sikker på at resultatet er riktig legges det inn ekstra parameterverdier for å justere algoritmen.

4.1.2.4 Backend-logikk (.Net/C#)

Dette er koden som binder hele programmet sammen og snakker med alle programmene vi lager. Her skal logikken koble sammen klient, sky og OCR. Dette gjøres ved å lage et API som kommuniserer med de ulike tjenestene. I banken skal API lages i .NET CORE. .NET CORE blir skrevet i programmeringsspråket C#.

Vi bruker autorisasjonsbevis, såkalte tokens, for verifisering av en tilkobling er pålitelig, at en forespørsel er fra en innlogget kunde.

JWT (JSON Web Tokens), er metoden vi bruker for å utføre sikker kommunikasjon mellom to parter. Det brukes et digitalt sertifikat (PKI) for å verifisere at en bruker er hvem den er. Dette gjøres ved at en entitet har et nøkkelpar bestående av en offentlig og privat nøkkel. Kun de to nøklene sammen kan brukes til å kryptere og dekode meldinger. Matematikken blir gjort i et kryptografisk system som f.eks. RSA (Rivest–Shamir–Adleman), som utfører denne prosessen.

API'et består av forskjellige endepunkter som brukeren kan utføre forespørsler til og få tilbake informasjon fra. Endepunktene er:

- Hente alle kvitteringer til en kunde
- Lagre bilde til Azure Storage
- Hent bilde fra Azure Storage
- Slett bilde fra Azure Storage
- Endre bilde fra Azure Storage
- Hent tekst til bilde fra Azure Cognitive Service

4.1.2.5 Skylagring (Microsoft Azure Blob Storage)

Vi lagrer bildene av kvittering sammen med godkjent ekstrahert tekst som blobber i en container for hver kunde.

Da vi fikk tildelt oppgaven fikk vi et krav om å bruke Microsoft Azure sine skytjenester for lagringen av data. Siden vi skulle lagre et bilde så var det mest naturlig å lagre det i en Blob Storage. Vanlige databaser som kunne lagrer tekst vil bruke en vanlig SQL-database. Når det gjelder en fil så er det vanligste å benytte seg av Blob Storage. Dette er fordi at blob storage er et filsystem, og det går raskere å lese fra filsystemet enn fra databasen. Vi beholder størrelsen på bildet ved å ikke kode bildet til base64, og lagre det i en sql-database. Denne kodingen fører til en overhead ved at størrelsen øker.

Formel for base64 av en input på n bytes: $x = 4\left(\frac{1}{3}n\right)$

Størrelsen av en input på «123456» i ASCII hvor hver karakter er en byte, er seks bytes.

Konvertering til base64 vil da bli $4\left(\frac{6}{3}\right) = 8\ bytes$

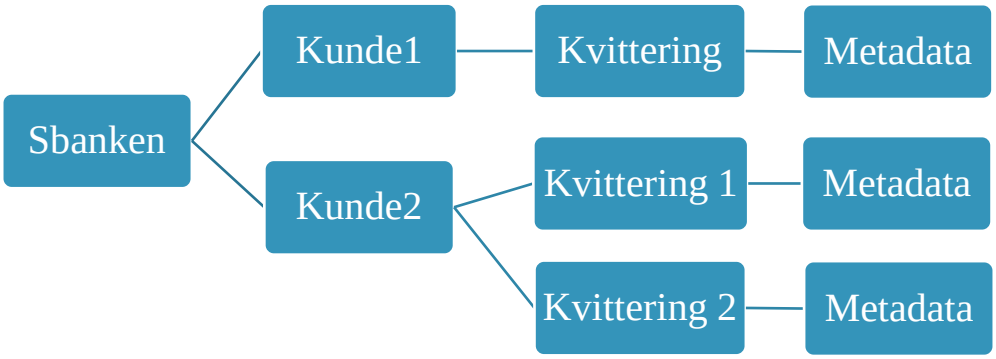
Forholdet mellom output og input bytes er 4:3 (33% overhead).

For små filer er ikke dette så betydelig. Den største fordelen er raskere utveksling av data, og vi kan cache bildet slik at det ikke vil være nødvendig å laste det inn på nytt. Dette er mulig siden metadataen til bilde gir oss en lenke som viser bilde, denne blir da lagret av nettleseren og blir ikke lastet på nytt.

4.1.2.5.1 lagringsstruktur

Strukturen til skyløsning er som følgende: Sbanken har en overordnet konto, og denne kontoen har flere kontainere. En kontainer tilhører en kunde og en kontainer kan ha flere kvitteringer. En kvittering kan ha flere metadatafelt.

Strukturen er representert på følgende måte:



Figur 4.6 Azure Storage arkitektur.

Kontaineren til en kunde er laget med en unik kryptografisk hash av kunden sitt personnummer. Hashing er en matematisk algoritme som gjør om data av tilfeldig størrelse til en streng med ordnet størrelse. Det er en enveis funksjon som betyr at en ikke kan dekode en tekst som er hashet. Vi har valgt å hashe personnumrene fordi vi av sikkerhetsmessige årsaker ikke har lov til å lagre personnumre i skyen.

Hver kunde har en kontainer med en unik id slik at vi kan endre eller slette dem etter kundens ønske. Kontaineren inneholder kvitteringene til kunden. Id'en blir laget ved å hashe klokkeslettet til datamaskinen. Til hver kvittering lagrer vi metadata som gir nøkkelinformasjon om kvitteringen. Nøkkelinformasjonen i vårt tilfelle er sum og dato. Når vi skal laste inn kvitteringen senere blir det lett å hente sum og dato også til bilde, siden de er lagret som metadata.

4.2.1 Klientdesign

Vi fikk på forhånd en mal på hvordan designet av siden skal se ut. Det er vanlig at en designer i Sbanken lager dette. Når vi utviklet løsningen var det bare en generell skisse av forsiden for mine kvitteringer. I Sbanken ligger det et bibliotek med ferdiglagde komponenter av knapper, former, farger og figurer. Disse benyttet vi så lenge det fantes passende i komponentbiblioteket. Vi måtte også ta hensyn til at det skal fungere på alle skjermer. Dersom bredden på skjermen blir mindre vil vi også presentere siden på en måte som er tilpasset skjermstørrelsen bedre. Dette kan gjøres ved å bruke en stiltype som heter CSS Flexbox. Denne stiltypen kan brukes til å flytte elementer på siden under hverandre ved smal skjerm og ved siden av hverandre ved bredere skjerm.

Klienten vår er bygget opp av komponenter, knapper og ikoner osv. Disse henter vi fra komponentbiblioteket til Sbanken, slik at vi holder på designet til nettbanken. De øvrige

utvikler vi selv. Dersom det er en komponent som kan brukes flere steder legger vi den til i komponentbiblioteket til banken. Gjennom å utvikle komponentbasert kode oppnår vi at applikasjonen vil føles mer dynamisk og raskere. Dette er mulig fordi React oppdaterer kun de komponentene som endres seg. Hver komponent har sitt eget lokale grensesnitt og variabler, og de har ansvar for seg selv. I objektorientert programmering kalles dette «Single responsibility principle». Komponentene i applikasjonen vår oppnår lav kobling mellom objektene, og på den måten garderer vi oss mot å innføre feil i programmet.

5 Evaluering

Evalueringen av produktet vil skje på forskjellige måter. Vi skal først evaluere det i forhold til brukerhistoriene og i hvilken grad disse kan oppfylles. Det vil også bli evaluert av intern veileder og eier av produktet i banken. Vi må også ta hensyn til kvaliteten på koden og lage tester.

5.1 Evalueringsmetode

Her beskriver vi de forskjellige metodene for å evaluere produktet.

5.1.1 Evaluering basert på brukerhistorie

Applikasjonen Mine Kvitteringer skal evalueres etter brukerhistoriene definert i kapittel 2. For at applikasjonen skal være tilstrekkelig må delpunktene i disse to brukerhistoriene oppfylles. Da vil produktet legges inn i betabanken slik at betakunder kan teste det. Etter samtaler med intern veileder ble kobling mot transaksjon tatt vekk fra kravet om piloten vi skulle levere. Følgende punkter i brukerhistoriene skulle implementeres i applikasjonen:

Som kunde skal jeg kunne ta bilde av en kvittering og laste den opp i nettbanken:

- Jeg skal kunne trykke på en lenke på siden "Mine kvitteringer" som lar meg ta/laste opp et bilde av en kvittering.
- Jeg skal kunne legge til kommentar før jeg laster opp bildet.
- Jeg skal få en tilbakemelding på at jeg har tatt bilde av kvittering og at feltene er tolket og kan gjøre endringer i tolket tekst om nødvendig.
- Jeg skal få en tilbakemelding når bildet ikke kan tolkes eller aksepteres og hvorfor den ikke gjør det.

Som kunde skal jeg finne igjen kvitteringer jeg har tatt bilde av i nettbanken:

- Som kunde skal jeg kunne finne igjen bilder av kvitteringer på transaksjoner jeg har betalt i nettbanken.
- Som kunde skal jeg kunne se kvitteringen tilkoblet transaksjonen i kontoutskriften.
- Jeg skal kunne trykke på "Slett kvittering" som permanent sletter bildet og teksten.
- Jeg skal kunne filtrere/sortere listen av kvitteringer.

5.1.2 Evaluering gjort av intern veileder og produkteier

Applikasjonen skal vises gjennom en demo i banken. Der vil vi gå gjennom brukstilfellene og lage en plan for videre arbeid. Denne demoen vil presenteres for intern veileder og produktansvarlig for applikasjonen.

5.1.3 Evaluering av kode

Under utvikling har vi kontinuerlig drevet med manuell testing for å sjekke at vi får ønsket funksjon. Dersom vi har lagt til funksjoner eller endret på koden, har vi alltid lastet opp forskjellige kvitteringer for å bekrefte at programmet fungerer. Vi har også laget tester i et javascript-bibliotek som heter Jest. Ved å bruke Jest har vi kunnet teste om tekstekstraheringen klarer å hente ut informasjon vi sender til testen på forhånd. Det betyr at vi skriver inn en dato og en totalsum som testen skal prøve å finne. Dersom testene klarer å hente akkurat den informasjonen vi har gitt kan vi anta at den fungerer tilstrekkelig.

Vi har også hatt et ansvar for å opprettholde god kvalitet på koden. Det betyr at vi kontinuerlig skal refaktorere kode når vi ser at vi kan skrive det om enklere, eller lage nye metoder som kan brukes flere steder. For å sjekke om kvaliteten på koden er tilstrekkelig går vi gjennom den med en mer erfaren utvikler i banken.

5.1.4 Evaluering med logging

Sbanken ønsket at verdiene fra OCR skulle være korrekte i minst 90% (0,9) av tilfellene hvor kunden laster opp bildet. Vår løsning for å tallfeste dette er å lage en loggemetode som sjekker hvor ofte kunden endrer på datoen eller summen. På denne måten kan vi sjekke hva som gjorde at den aktuelle kvitteringen ikke ble korrekt lest. Dette er også ett av tilfellene i risikolisten i kapittel 9.1. Når noe blir logget kan vi raskt se hvilken kunde det var, og tilhørende kvittering. Følgelig kan vi se hvorfor feilen oppstår. Det kan være forskjellige årsaker, men et dårlig bilde vil mest sannsynlig ikke bli tolket korrekt. Det kan også være at algoritmen vår for å finne verdiene ikke er gode nok. Når vi deler antall endringer på antall opplastninger vil vi få en prosentandel som sier hvor ofte det oppstår feil verdi. Dersom dette tallet blir under 0,1 vil vi vite om vi overholder målet om 90% (0,9) korrekt tekstekstrahering.

5.2 Evalueringsresultat

Her går vi gjennom evalueringsresultatene basert på de forskjellige evalueringene nevnt i kapittel 5.1.

5.2.1 Evalueringsresultat brukerhistorie

Vi vil her gå gjennom hvert delpunkt i de to brukerhistoriene og konkludere med om de er fullført eller trenger videre arbeid.

Brukercase 1: Som kunde skal jeg kunne ta bilde av en kvittering og laste den opp I nettbanken

- Kunden kan på siden Mine Kvitteringer trykke på en knapp som lar de laste opp et bilde.
- Kunden kan legge til en tekst om ønskelig, denne kan også endres.
- Kunden får opp et resultat med tolket tekst. Her kan kunden også endre felt som er feil.
- Kunden kan ikke få tilbakemelding hvis bildet ikke kan tolkes og hvorfor. Dette blir videre arbeid.

Brukercase 2: Som kunde skal jeg finne igjen kvitteringer jeg har tatt bilde av I nettbanken

- Kunden kan finne bilde av kvitteringen, men den er ikke koblet til en transaksjon. Kobling til transaksjon ble tatt vekk som et krav.
- Kunden kan slette en kvittering.
- Kunden får opp en liste over alle kvitteringer, men kan ikke filtrere kvitteringer. Dette går til videre arbeid.

Som tidligere nevnt er de fleste delkrav oppfylt, men noe er ikke implementert og vil gå til videre arbeid i applikasjonen.

5.2.2 Evaluering gjort av intern veileder og produkteier

Vi presenterte det vi hadde utviklet for vår interne veileder 6. mai. Han var veldig fornøyd med resultatet. Det skal jobbes videre med denne applikasjonen, og da vil de resterende kravene bli implementert. Planen er at vi skal presentere en demo for intern veileder og produktets eier i juni 2019. Det viktigste for dem nå var at vi har en fungerende pilot av applikasjonen som et fundament for videre utvikling.

5.2.3 Evaluering av kode

Vi gikk gjennom koden med forskjellige utviklere i banken. Disse jobber innen forskjellige områder som frontend og backend. For frontend fikk vi tilbakemelding på å bruke komponenter som eksisterte i banken. Når vi gikk gjennom backend-koden ble vi gjort oppmerksom på å lage metoder for kode som ble brukt flere steder i programmet. Vi har refaktorert koden, men dette er noe som vil skje hele tiden under utviklingen.

5.2.4 Evaluering med logging

Loggingen blir en veldig viktig del av programmet, men er for øyeblikket ikke implementert. Dette må være med i programmet før vi legger det ut i betabanken for å kunne tallfeste feil i tekstekstraheringen. Vi er blitt enige med intern veileder om at dette skal utføres når vi jobber videre med programmet i sommer.

6 Diskusjon, konklusjon og videre arbeid

I dette delkapittelet vil vi forklare konsekvensene av fremgangsmetoden vi har valgt. Hvordan valgene vi har gjort innvirket på resultatet og forbedringer som kan gjøres dersom vi skulle utført prosjektet på nytt.

6.1 Valg av fremgangsmetode og innvirkning på resultat

Vi valgte å utvikle en løsning som gjorde at ekstraheringen av kvitteringsresultatet ble gjort i frontend. Det positive med dette valget var at vi var komfortable med ReactJS og hadde en stor egeninteresse for å ta i bruk språket. Applikasjonen ville nok ha vært mer effektiv om vi hadde gjort ekstraheringen i backend. Årsaken til dette er at en da ville spart seg for ekstra sending av data fra backend til frontend. Vi begynte tidlig i semesteret å tilegne oss kunnskap innen de teknologiene vi skulle bruke. Dette ga oss et fortrinn under analyse av oppgaven, samt at vi også relativt raskt fikk på plass fungerende kode som kunne brukes under integreringen mot Sbanken sitt IT-system. Ved prosjekt start begynnelsen av mars opplevde vi store utfordringer med å integrere teknologiene inn i nettsiden til banken. Vi trodde denne delen av prosjektet skulle ta en uke men det tok ca. tre uker før implementeringen var ferdig. Følgelig gikk mye tid med til noe som ikke direkte handlet om applikasjonen vi skulle lage. Vi sakket akterut i forhold til opprinnelig Gannt-diagram, og vi ble bekymret for hvorvidt vi skulle klare å overholde fremdriftsplanen. Ved å ha kontinuerlige møter med intern veileder var det positivt at vi fikk en rask avklaring for å redusere arbeidsmengden gjennom å utelate koblingen mot transaksjoner.

Vi er veldig tilfreds med måten vi har valgt å løse lagringen av kvitteringer og tilhørende data på. Sbanken hadde selv ikke brukt denne teknologien og vi fikk velge selv hvordan vi skulle løse det. Etter samtaler med sikkerhetskompetente ressurser i banken om hva som var lov å lagre, kom vi opp med en god løsning slik at vi unngikk at juridisk avdeling måtte godkjenne noe for oss. Lagring av kvitteringer med metadata i en Blob Storage viste seg å være et godt valg. Blob Storage hadde mange metoder som gjorde det raskt for oss å få innhentet den informasjonen vi hadde lagret. Vi fikk svært gode resultater på tiden det tok å utføre operasjoner i Blob Storage, og vi ble også overrasket over hvor billig det var å bruke tjenestene til Microsoft, noe som er positivt for banken siden de har fremtidsplaner om å ha hele IT-systemet sitt i Azure.

Vårt kanskje viktigste valg for prosjektet var å gjøre bruk av OCR-tjenesten. Dersom OCR ikke hadde vært god nok til å tolke teksten fra bildene ville det mest sannsynlig ikke vært mulig å lage denne applikasjonen. Det viste seg også i begynnelsen at vi benyttet oss av en eldre versjon som ga oss svært dårlige tolkninger. Da vi analyserte hvordan vi skulle hente ut riktig tekst ved bruk av denne versjonen, ble løsningene vi diskuterte fort veldig kompliserte. Problemene med denne gamle versjonen var at teksten ikke ble representert fra topp til bunn. Her vurderte vi å lage algoritmer som tok hensyn til den geometriske aksen til kvitteringen for å hente ut verdiene. Heldigvis fant vi ut at det var en nyere versjon vi kunne benytte og denne versjonen gjenkjente teksten nærmest ordrett og listet teksten fra topp til bunn. Resultatet vi fikk tilbake gjorde det mulig for oss å lage mindre kompliserte algoritmer for ekstrahering av data. Vi utviklet også tester tidlig i utviklingsfasen som verifiserte at algoritmene fungerte veldig bra. Dette gjorde oss tryggere på at OCR og algoritmene fungerte. Det var positivt å se at regex fungerte for å hente ut data, da dette er en veldig hurtig måte å gå gjennom teksten på.

Vi er veldig fornøyd med løsningen vi har utviklet, spesielt hadde vi en god arkitektur i forkant som gjorde at vi var veldig forberedt på hva vi skulle gjøre da vi begynte å kode. Det første vi gjorde var å lage en flyt i hele programmet for å få alle delene til å fungere sammen, for så å forbedre hver del i arkitekturen. Responsen vi har fått fra intern veileder har vært overveldende positiv og vi tolker det som at produktet er godt gjennomført i forhold til hva de forventet.

Dersom vi skulle begynt prosjektet på nytt ville vi ha som tidligere nevnt gjort all logikken i backend, og da kun presentert data for kunden. Vi ville ha satt av mer tid til å sette opp miljøet i banken da det viste seg å være en tidkrevende prosess. Det kunne også vært en fordel å tilegne seg mer kunnskap om teknologiene vi skulle bruke før vi begynte å lage designet for arkitekturen.

6.2 Konklusjon og videre arbeid

Applikasjonen som vi har utviklet tilfredsstiller de fleste kravene som var bestemt ved prosjektets oppstart. Sbanken er svært fornøyd med det vi har utviklet, og de mener Mine Kvitteringer er noe som kan bygges videre på og forbedres over tid. Målet var å ha et fungerende produkt som kunne testes av ansatte i banken først. For å kunne teste de ikke-funksjonelle kravene er det gunstig at det blir gjort av flere, men resultatene vi har sett når vi har testet det lokalt er veldig gode. Applikasjonen klarer som oftest å hente ut summen og

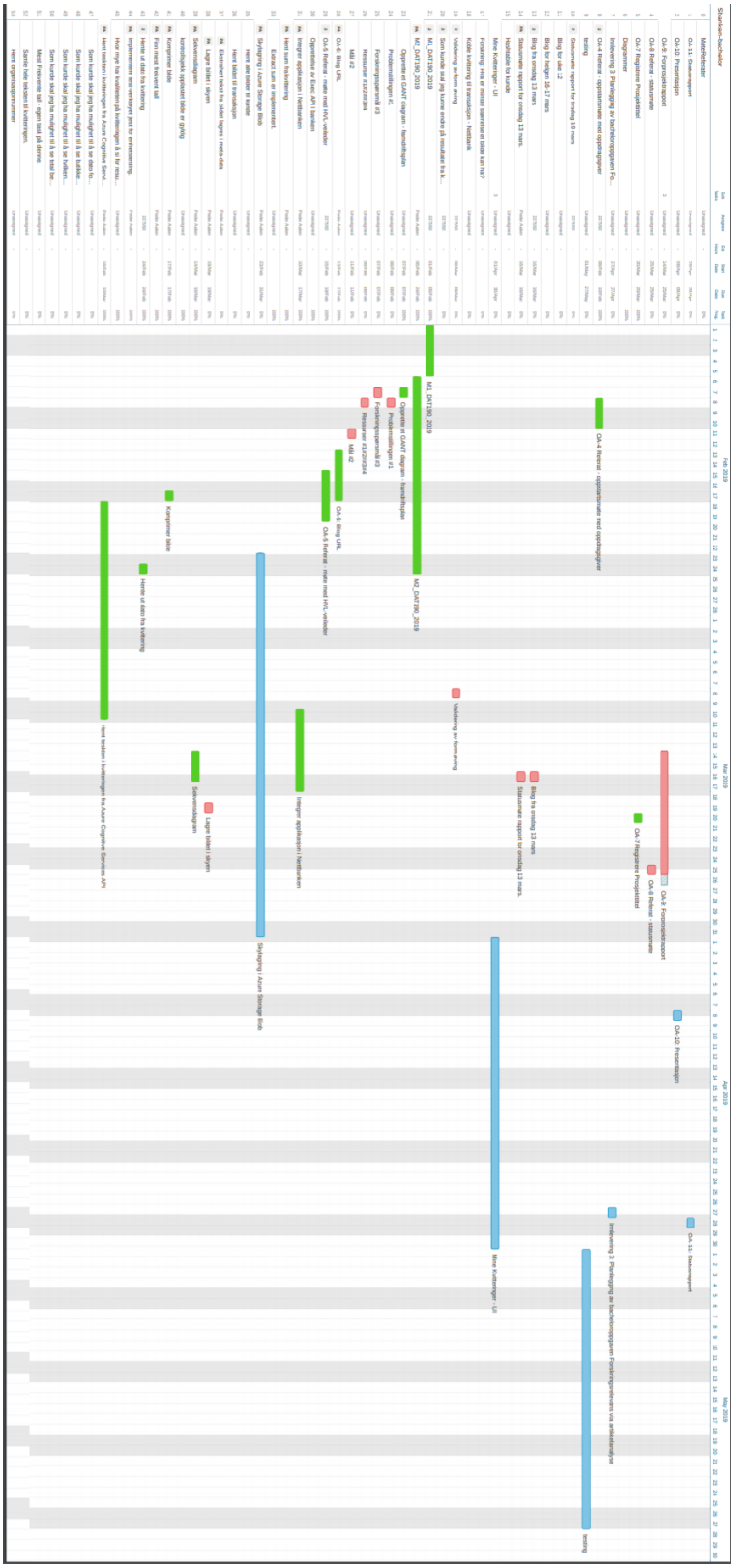
datoen. Dette må selvsagt bli tallfestet i fremtiden når mange skal bruke applikasjonen internt. Det er ikke planlagt hvem som skal jobbe videre med applikasjonen når vi leverer denne fra oss. Ønsket var å ha en pilot på riktig server i banken, men utviklingen har tatt lengere tid enn hva vi trodde. Ifølge Gannt-diagrammet skulle piloten vært ute 1. mai. Her var det en del uforutsette hindringer som gjorde overføringen til ny server utfordrende.

7 Appendiks

7.1 Risikoliste

Risikobeskrivelse	**Sannsynlighet**	**Konsekvens**	**Tiltak**
Kunde laster opp skadelig programvare	Liten	Kritisk	Legge begrensninger på filtyper kunde kan laste opp
Kvittering blir ikke lest korrekt	Middels	Middels	Gi kunde god informasjon om hvordan ta bilde, bruke riktig versjon
Andre enn kunden får tilgang til kundes data	Liten	Kritisk	Legge inn sjekk mot riktig kunde til riktig container
For liten tid til å ferdigstille applikasjon	Middels	Middels	Ukentlige statusmøter, god oppfølgning med intern veileder

7.2 Gantt-diagram



8 Referanser

stackoverflow. (2019). *Storing Images : DB or File System* -. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/1234202/storing-images-db-or-file-system [Accessed 26 May 2019].

Docs.microsoft.com. (2019). *Choosing a data storage technology*. [online] Available at: https://docs.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/data-storage#azure-storage-blobs [Accessed 4 Mar. 2019].

stackoverflow. (2019). *Storing image in database directly or as base64 data?*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/9722603/storing-image-in-database-directly-or-as-base64-data [Accessed 10 Feb. 2019].

Docs.microsoft.com. (2019). *Deciding when to use Azure Blobs, Azure Files, or Azure Disks*. [online] Available at: https://docs.microsoft.com/en-us/azure/storage/common/storage-decide-blobs-files-disks [Accessed 10 Mar. 2019].

Gray, J. (2019). *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem*. [online] Microsoft Research. Available at: https://www.microsoft.com/en-us/research/publication/to-blob-or-not-to-blob-large-object-storage-in-a-database-or-a-filesystem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fapps%2Fpubs%2Fdefault.aspx%3Fid%3D64525 [Accessed 3 Mar. 2019].

stackoverflow. (2019). *Storing Images in DB*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/3748/storing-images-in-db-yea-or-nay [Accessed 6 Feb. 2019].

stackoverflow. (2019). *Storing images - SQL DB vs Azure Blob storage*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/39650052/storing-images-sql-db-vs-azure-blob-storage [Accessed 6 Feb. 2019].

stackexchange. (2019). *Store images in the database, or in files with a database link?*. [online] Software Engineering Stack Exchange. Available at: https://softwareengineering.stackexchange.com/questions/105191/store-images-in-the-database-or-in-files-with-a-database-link [Accessed 1 Mar. 2019].

string, K., Becerril, L. and Moyal, H. (2019). *Know file size with a base64 string*. [online] Software Engineering Stack Exchange. Available at:

https://softwareengineering.stackexchange.com/questions/288670/know-file-size-with-a-base64-string [Accessed 7 Feb. 2019].

Docs.microsoft.com. (2019). Deciding when to use Azure Blobs, Azure Files, or Azure Disks. [online] Available at: https://docs.microsoft.com/en-us/azure/storage/common/storage-decide-blobs-files-disks [Accessed 13 Mar. 2019].

(smoothly), R., Ott, S., Nhat, T., Yakubov, D. and c, E. (2019). Resize image with javascript canvas (smoothly). [online] Stack Overflow. Available at: https://stackoverflow.com/questions/19262141/resize-image-with-javascript-canvas-smoothly [Accessed 15 Mar. 2019].

Mathematics Stack Exchange. (2019). How to get the aspect ratio of an image?. [online] Available at: https://math.stackexchange.com/questions/180804/how-to-get-the-aspect-ratio-of-an-image [Accessed 17 Mar. 2019].

Docs.microsoft.com. (2019). Azure Storage Documentation - Tutorials, API Reference. [online] Available at: https://docs.microsoft.com/en-us/azure/storage/ [Accessed 11 Apr. 2019].

Docs.microsoft.com. (2019). *Blob Containers (Azure Storage Resource Provider)*. [online] Available at: https://docs.microsoft.com/en-us/rest/api/storagerp/blobcontainers [Accessed 26 Apr. 2019].

Docs.microsoft.com. (2019). *Introduction to Blob storage - Object storage in Azure*. [online] Available at: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction [Accessed 26 Mar. 2019].

Docs.microsoft.com. (2019). *Introduction to Blob storage - Object storage in Azure*. [online] Available at: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-overview [Accessed 26 Mar. 2019].

En.wikipedia.org. (2019). *Base64*. [online] Available at: https://en.wikipedia.org/wiki/Base64 [Accessed 16 May 2019].

Stack Overflow. (2019). *Base64 length calculation?*. [online] Available at: https://stackoverflow.com/questions/13378815/base64-length-calculation [Accessed 14 May 2019].

Jwt.io. (2019). *JWT.IO - JSON Web Tokens Introduction*. [online] Available at: https://jwt.io/introduction/ [Accessed 26 May 2019].

Westus.dev.cognitive.microsoft.com. (2019). *Microsoft Cognitive Services*. [online] Available at: https://westus.dev.cognitive.microsoft.com/docs/services/5adf991815e1060e6355ad44/operations/587f2c6a154055056008f200 [Accessed 26 April 2019].

Westus.dev.cognitive.microsoft.com. (2019). *Microsoft Cognitive Services*. [online] Available at: https://westus.dev.cognitive.microsoft.com/docs/services/5adf991815e1060e6355ad44/operations/587f2cf1154055056008f201 [Accessed 26 Apr. 2019].

Docs.microsoft.com. (2019). *Understanding Block Blobs, Append Blobs, and Page Blobs*. [online] Available at: https://docs.microsoft.com/en-us/rest/api/storageservices/understanding-block-blobs--append-blobs--and-page-blobs [Accessed 26 Apr. 2019].

Docs.microsoft.com. (2019). *Azure Quickstart - Create a blob in object storage with the Azure portal.* [online] Available at: https://docs.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-portal [Accessed 26 Apr. 2019].

Reactjs.org. (2019). *Introducing Hooks – React*. [online] Available at: https://reactjs.org/docs/hooks-intro.html [Accessed 26 Jan. 2019].

CSS-Tricks. (2019). *A Complete Guide to Flexbox | CSS-Tricks*. [online] Available at: https://css-tricks.com/snippets/css/a-guide-to-flexbox/ [Accessed 17 Mar. 2019].