Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Patrick Thorkildsen
Quan Tran

SkiltInfo

Utvikling av applikasjon innen veginformatikk

Bacheloroppgave i Dataingeniør
Veileder: Majid Rouhani

Mai 2020

NTNU
Kunnskap for en bedre verden

Patrick Thorkildsen
Quan Tran

SkiltInfo

Utvikling av applikasjon innen veginformatikk

Bacheloroppgave i Dataingeniør
Veileder: Majid Rouhani
Mai 2020

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Forord

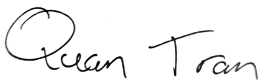
Oppgaven ble valgt etter at prosjektgruppen tok kontakt med Triona AS for å spørre om mulighet for å skrive bacheloroppgave for dem. Triona hadde flere oppgaveforslag, men vi gikk for denne siden den var mest relevant med tanke på gruppens kompetanse og tidligere erfaring.

Samarbeidet startet med Triona på kontorene deres, men ble etter hvert hemmet av tiltakene som ble innført i Norge den 12. mars 2020 som følge av utbruddet av Covid-19.

Vi vil gjerne takke veilederen vår ved NTNU, Majid Rouhani, for god veiledning og råd under prosjektet. Triona AS for god støtte og hjelp med både oppgaven og kontorplass, spesielt takk til veilederen vår og prosjektleder i Triona, Ragnhild Egtvedt, for god oppfølging. Vi vil også takke Aurora Wallenius Kjevik for hjelp og råd til design og utforming av wireframes i Adobe Illustrator.

19.05.2020, Trondheim

Patrick Thorkildsen



Quan Tran

Oppgavetekst

Oppgaveteksten vår ble utformet av Triona AS, og ble endret litt under oppstartsmøtet. Her presenteres både den opprinnelige og den reviderte oppgaveteksten.

Opprinnelig oppgavetekst:

Vi ønsker å få utviklet en applikasjon som skal benytte en eksisterende bildegjenkjenningsløsning til å hente ut objekttype og posisjon. Denne informasjonen skal så benyttes til å hente ut fullstendig informasjon om nettopp dette objektet fra NVDB (Nasjonale vegdatabanken hos Statens Vegvesen), og denne informasjonen presenteres for brukeren i et GUI.
Forventninger

- Løsningen skal være en webapplikasjon for nettbrett/mobil
- Løsningen skal kunne ta et bilde av et objekt som inkluderer objektets posisjon
- Bildet skal sendes til den eksisterende løsningen for å bli gjenkjent som et type objekt.
- Utifra informasjonen om objektets posisjon og type objekt skal komplette objektinformasjonen som hentes fra NVDB kunne vises i løsningen.

Under oppstartsmøtet og underveis i utviklingsprosessen ble det besluttet å endre oppgaveteksten. Produktet skulle være en mobilapplikasjon for Android og iOS istedet for en webapplikasjon, og løsningen skulle lages på en slik måte at det er mulig å implementere bildegjenkjenningsløsningen, uten å faktisk implementere den.

Revidert oppgavetekst:

Vi ønsker å få utviklet en applikasjon som skal ha mulighet til å benytte en eksisterende bildegjenkjenningsløsning til å hente ut objekttype og posisjon. Denne informasjonen skal så benyttes til å hente ut fullstendig informasjon om nettopp dette objektet fra NVDB (Nasjonale vegdatabanken hos Statens Vegvesen), og denne informasjonen presenteres for brukeren i et GUI.
Forventninger

- Løsningen skal være en mobilapplikasjon for Android og iOS
- Løsningen skal kunne ta et bilde av et objekt som inkluderer objektets posisjon
- Bildet skal kunne sendes til den eksisterende løsningen for å bli gjenkjent som et type objekt.
- Utifra informasjonen om objektets posisjon og type objekt skal komplette objektinformasjonen som hentes fra NVDB kunne vises i løsningen.

Grunnen til at bildegjenkjenningsmodulen ikke skulle tas i bruk enda er at prosjektgruppen ikke fikk tilgang til den av Triona AS.

Sammendrag

Oppgaven er å lage en mobilapplikasjonen for å finne informasjon fra Nasjonal Vegdatabank (NVDB) om veiskilt i nærheten. Applikasjonen skal kunne implementere en eksisterende bildegjenkjenningsmodul som bruker bildet som ble tatt gjennom applikasjonen til å gjenkjenne hva slags skilt det ble tatt bilde av.

Hensikten med denne applikasjonen er å forenkle og effektivisere hvordan arbeidere i veisektoren finner informasjon om skilt. Applikasjonen er delt inn i to deler, server og klient. Serveren er skrevet i Java og er forbindelsen mellom NVDB og klienten. Klienten er skrevet i JavaScript med React, og React Native i Expo CLI.

Oppgaven ble løst ved å bruke GPS-posisjonen til brukeren, samt skilt-typen brukeren ønsker å søke etter. Med dette gjøres det et søk i NVDB, og de aktuelle skiltene presenteres på et kart for brukeren slik at det riktige skiltet kan velges for å få mer detaljert informasjon. En annen viktig funksjonalitet ved produktet er at brukeren kan velge selv hva slags informasjon han ønsker å hente ut. Dette ble løst med mulighet for å sette filter på søket, brukeren kan enten velge blant de forhåndsdefinerte filtrene eller opprette og tilpasse sine egne med kun de feltene brukeren ønsker.

Bildegjenkjenningsmodulen er ikke implementert, men applikasjonen er utviklet med tanke på at modulen skal implementeres senere.

Det ble ikke tatt brukertester av den relevante brukergruppen, arbeidere i veisektoren, grunnet utbruddet av covid-19 og de nasjonale tiltakene som ble innført. I stedet ble det gjort brukertester med studenter og personer i samme husholdning som prosjektdeltakerne. Brukertestene viste at applikasjonen som ble utviklet, *SkiltInfo*, oppfylte problemstillingen. Selv om de gjennomførte testene ikke var optimale, var dette det eneste og beste alternativet med tanke på situasjonen.

Abstract

The task is to develop a mobile application to retrieve information from Nasjonal Vegdatabank (NVDB) about road signs located nearby. The application should have the possibility of implementing an existing image recognition module that uses the picture taken in the app to recognize the type of sign to search for.

The purpose of the application is to simplify and streamline how workers in the road sector find necessary information about road signs. The application is divided into a server and a client. The server is written in Java and is the connection between NVDB and the client. The client is written in JavaScript with React and React Native in the Expo CLI.

The task was solved by utilizing the GPS-position of the user and the id of the type of sign the user wishes to search for. Using this, a search is done in NVDB, and the possible signs are presented for the user in a map. The user can then choose the correct sign from the markers on the map to display more information about it. Another important functionality is to allow the user to choose what kind of information to retrieve. This was solved by letting the user apply filters to the search, the user can either choose between the predefined filters or create their own with only the necessary information.

The image recognition module is not implemented, but the application is developed with the module in mind, so that it can be implemented later.

User tests of the relevant user group, workers in the road sector, were not completed due to the outbreak of covid-19 and the national measures that were introduced to prevent spreading. As an alternative, user tests with students and people in the same household as the project participants were conducted. The user tests showed that the application that was developed, *SkiltInfo*, met the research question. Even though the completed tests were not optimal, this was the best and only alternative with the situation at the time.

Innhold

Forord i

Oppgavetekst ii

Sammendrag iii

Abstract iv

Innhold vii

Figurer viii

Akronymer, forkortelser og definisjoner ix

1 Introduksjon og relevans 1
- 1.1 Bakgrunn . . . 1
- 1.2 Problemstilling . . . 1
- 1.3 Struktur . . . 1

2 Teori 2
- 2.1 Effektivisering . . . 2
  - 2.1.1 Hvordan effektivisere? . . . 2
  - 2.1.2 Robotisk Prosessautomatisering (RPA) . . . 3
- 2.2 Maskinlæring og klassifisering . . . 3
  - 2.2.1 Maskinlæring . . . 3
  - 2.2.2 Klassifisering . . . 4
- 2.3 GNSS og koordinatsystemer . . . 4
- 2.4 Systemutvikling . . . 5
  - 2.4.1 VCS . . . 6
  - 2.4.2 Kontinuerlig Integrasjon . . . 6
  - 2.4.3 Menneske Maskin Interaksjon (MMI) . . . 6
  - 2.4.4 Universell Utforming . . . 6
  - 2.4.5 Testing og Test Driven Development . . . 7

3 Valg av Teknologi og Metode 8
- 3.1 Prosessautomatisering i *SkiltInfo* . . . 8
  - 3.1.1 Uten RPA . . . 9
  - 3.1.2 Med RPA . . . 10
- 3.2 Standarder . . . 11
  - 3.2.1 JSON . . . 11
  - 3.2.2 Hypertext Transfer Protocol (HTTP) . . . 11

3.2.3 Representational State Transfer (REST) . . . 11
3.3 Konvertering mellom lat/long og N/E . . . 11
3.4 Systemutviklingteknologi og -metode . . . 12
3.4.1 React og React Native . . . 12
3.4.2 Expo . . . 12
3.4.3 JavaScript Testing Framework (Jest) . . . 13
3.4.4 JUnit . . . 13
3.4.5 Java Code Coverage (JaCoCo) . . . 13
3.4.6 Git . . . 13
3.4.7 Travis-CI . . . 14
3.5 Agile metoder og Scrum . . . 14
3.6 Arbeids- og rollefordeling . . . 14

4 Resultater **15**
4.1 Vitenskapelige resultater . . . 15
4.1.1 Server . . . 15
4.1.2 Klient . . . 15
4.1.3 Design . . . 16
4.2 Ingeniørfaglige resultater . . . 21
4.2.1 Prosjektmål . . . 21
4.2.2 Brukertester . . . 23
4.2.3 Kodetester . . . 24
4.3 Administrative resultater . . . 24
4.3.1 Timebruk . . . 25
4.3.2 Fremdriftsplan (Gantt diagram) . . . 25
4.3.3 Utviklingsmetodikk . . . 25

5 Diskusjon **26**
5.1 Vitenskapelige resultater, diskusjon . . . 26
5.1.1 Server . . . 26
5.1.2 Klient . . . 27
5.1.3 Design . . . 27
5.2 Ingeniørfaglige resultater, diskusjon . . . 28
5.2.1 Prosjektmål . . . 28
5.2.2 Brukertester . . . 29
5.2.3 Kodetester . . . 29
5.3 Administrative resultater, diskusjon . . . 29
5.4 Helhetlig systemperspektiv . . . 30
5.5 Refleksjon over gruppearbeidet . . . 30

6 Konklusjon og videre arbeid **31**

6.1 Problemstilling 31
6.2 Videre arbeid 31

References **33**

7 Vedlegg A - Visjonsdokument **34**

8 Vedlegg B - Kravdokumentasjon **41**

9 Vedlegg C - Systemdokumentasjon **48**

Figurer

2 Mercator projeksjon 5
3 Oppdelingen av UTM soner 5
4 Informasjonsflyten uten bruk av RPA 9
5 Informasjonsflyten med bruk av RPA 10
6 Kameraskjermen i wireframe og sluttprodukt 16
7 Skjerm for visning av data i wireframe og sluttprodukt 17
8 Hjelpskjermen i wireframe og sluttprodukt 18
9 Instillingskjermen i wireframe og sluttprodukt 19
10 Filterskjermen i wireframe og sluttprodukt 20
11 Testdekningen i frontend 24
12 Testdekningen i Backend 24
13 Timebruk over prosjektperioden 25

Akronymer, forkortelser og definisjoner

- NVDB - Nasjonal vegdatabank, en database med informasjon om statlige, kommunale, private, fylkes- og skogsbilveger
- VCS - Versjonskontrollsystemer (version control systems), håndtering av kodeversjoner.
- CI - Kontinuerlig integrasjon (continuous integration), kjøring av testene på en virtuell maskin hver gang en ny versjon lastes opp.
- Kodebase - Lagringsplass for kode
- Refaktorere - Omstrukturering av kodebasen for å få den enklere å lese, endre, vedlikeholde og videreutvikle.
- Difi - Direktoratet for forvaltning og ikt, norsk direktorat for modernisering og omstilling av offentlig sektor. Lagt ned 31. desember 2019 og samlet under Digitaliseringsdirektoratet 1. januar 2020.
- Applikasjonsnivået - Forholdet seg direkte til, og utfører tjenester for applikasjoner.
- Forespørsel - En requesttil en server, kan være en GET, PUT, POST eller DELETE.
- Svar - Svaret fra en forespørsel, kalles også for respons.
- TLS - Transport Layer Security, en kryptografisk kommunikasjonsprotokoll.
- MMI - Menneske-Maskin-Interaksjon, studien av interaksjon mellom mennesket (brukeren) og datamaskiner.
- Frontend - Koden som lager det brukeren visuelt ser på skjermen, brukergrensesnittet.
- Backend - Koden som ligger nærmest hvor dataen er lagret, serverdelen av prosjektet.
- Rammeverk - Programvare med grunnleggende funksjoner som kan endres med brukerskrevet kode.
- Java - Objektorientert programmeringsspråk.
- Swift - Programmeringsspråk for Apple sine operativsystemer.
- Responsiv - Tilpasser seg skjermstørrelsen
- CLI - Kommandolinje grensesnitt (Command Line Interface)
- Android Studio - IDE for utvikling av Android applikasjoner.
- XCode - IDE for utvikling av Apple applikasjoner
- Barn - Et objekt eller en komponent som er inneholdt i et foreldreobjekt eller komponent

- Tilbakerulle (rollback) - Å gå tilbake til den forrige versjonen av koden.
- FAQ - Ofte Stilte Spørsmål (Frequently Asked Questions)
- Presentasjonslaget - Ligger over applikasjonslaget, og utfører tjenester for det.
- Mocking - Å etterligne en komponent eller modul med testing som formål
- State - Tilstanden til programmet med utgangspunkt i den informasjonen som er lagret til et gitt tidspunkt.

1 Introduksjon og relevans

1.1 Bakgrunn

Triona AS er et konsulentfirma med fokus på levering av IT-løsninger innen logistikk og infrastrukturrelatert virksomhet. De har sett et behov for å kunne effektivisere veiarbeid ved å gjøre det enklere å slå opp skilt i den Nasjonale vegdatabanken (NVDB). Dagens løsning med manuelle oppslag i databasen kan være tidkrevende, det ønskes derfor en mobil løsning som gjør det raskere å slå opp, samtidig som det er presist. Det innebærer utvikling av en mobilapplikasjon som er koblet opp mot en maskinlæringsmodell som gjenkjenner veiskilt fra et bilde tatt med applikasjonen. Oppgaven går ut på å utvikle denne applikasjonen uten bildegjenkjenningsmodulen implementert, men som er tilrettelagt for at den kan implementeres.

1.2 Problemstilling

Dette er et utviklingsprosjekt der det endelige målet er å effektivsere arbeidshverdagen til enkelte personer som arbeider i veisektoren. Problemstillingen er derfor formulert som følger:

> *Utvikling av en mobilapplikasjon for ansatte i veisektoren som effektiviserer og forenkler arbeidet med å hente nødvendig informasjon om skilt fra den nasjonale vegdatabanken.*

1.3 Struktur

Rapporten er delt opp i seks hovedkapitler. Kapittel 2 omfatter teorien som er nødvendig for å løse og/eller forstå oppgaven. I kapittel 3 forklares valg som er tatt underveis og under planlegging med utgangspunkt i det tekniske utviklingsarbeidet. Resultatene som prosjektgruppen har kommet frem til under arbeidet presenteres i kapittel 4. Dette kapittelet er tredelt; under *Vitenskapelige resultater* finnes resultatene knyttet til problemstillingen, *Ingeniørfaglige resultater* inneholder resultater knyttet til målene for prosjektet som er satt i visjonsdokumentet, og *Administrative resultater* er resultater knyttet til fremdriftsplanen.

I kapittel 5 diskuteres resultatene og hvorfor de ble som de ble. Her forklares også noen av valgene som ble tatt med utgangspunkt i resultatene, i tillegg til å diskutere noen av svakhetene rundt løsningen. Konklusjonen i kapittel 6 forklarer hva gruppen har kommet frem til med utgangspunkt i problemstillingen, med videre arbeid for løsningen.

2 Teori

I dette kapittelet dekkes det teoretiske grunnlaget for oppgaven. Først litt om effektivisering og Robotisk Prosessautomatisering, ettersom målet med applikasjonen er å effektivisere. Deretter litt generelt om maskinlæring, som brukes av bildegjenkjenningsmodulen som skal kunne implementeres. Dette er etterfulgt av litt teori om koordinatsystemer, og til slutt teori rundt systemutviklingsverktøy, -prinisipper og -metoder.

2.1 Effektivisering

Effektivisering handler om å finne fram til områder som kan løses rimeligere og smartere (Difi 2020). En måte å effektivisere gjennom teknologi og digitalisering er å automatisere. Målet med applikasjonen SkiltInfo er å effektivisere prosessen med å hente ut informasjon om veiskilt ved å automatisere det nødvendige oppslaget i NVDB. Digital effektivisering baserer seg ofte på robotisk prosessautomatisering, eller Robotic Process Automation (RPA), med bruk av maskinlæring eller kunstig intelligens.

2.1.1 Hvordan effektivisere?

Difi har satt opp en 5-stegs metode for systematisk effektivisering som omfatter perioden fra å kartlegge behovet til vedlikehold og videreføring av det nye systemet. Fasene i metoden ser slik ut:

1. *Kartlegg nåsituasjonen*
   I denne fasen skal det utarbeides en diagnose av virksomheten. Her skal målene for effektiviseringen settes, samt at de ønskede gevinstene skal defineres. Denne fasen er det virksomhetens ledelse som har hovedansvaret for å gjennomføre.

2. *Priotiter prosjekter*
   Her bestemmer ledelsen hvilke prosjekter de ønsker å iverksette og jobbe videre med. I tillegg skal det utarbeides tiltak for gevinstrealisering og effektiviseringspotensialet skal kartlegges.

3. *Beskriv tiltak*
   I denne fasen skal det foreslås konkrete tiltak for å oppnå effektiviseringspotensialet. Prosjektdeltakerne foreslår tiltak, og ledelsen velger tiltakene de ønsker å gå videre med.

4. *Gjennomfør tiltakene*
   Her skal tiltakene gjennomføres for å realisere gevinsten.

5. *Veien videre*
   Dette punktet handler om å legge en plan for hvordan effektiviseringsprogrammet skal integreres i virksomhetsstyringen når prosjektet er avsluttet.

Om denne stegvise tilnærmingen til effektivisering følges sikrer man at arbeidet blir målrettet, godt planlagt og at alle de relevante partene er involvert i planleggingen.

2.1.2 Robotisk Prosessautomatisering (RPA)

Robotic Process Automation (RPA) handler om å automatisere arbeidsprosesser slik at vi mennesker kan gjøre annet, mer verdifullt arbeid. Ofte brukes maskinlæring eller kunstig intelligens for å utføre disse oppgavene. RPA følger bestemte regler og logikk for å gjennomføre repetitive oppgaver raskere enn mennesker kan utføre dem. En vellaget RPA løsning vil også eliminere risikoen for menneskelige feil (Pwc 2020).

RPA er en svært aktuell teknologi som er i stadig utvikling. En RPA løsning baserer seg på en eller flere av disse teknologiene (Rouse 2020):

- *Skjermskraping*
  Skjermskraping er en teknologi som vokste frem før internettets tid. Det handler om å hente ustrukturert data fra programvare for så å strukturere det og presentere det på en ryddig måte. I dag brukes teknologien hovedsaklig for å hente ut data fra web og vise det frem på presentasjonslaget.

- *Automatisert arbeidsflyt*
  Dette begrepet ble først brukt i forbindelse med fremveksten av produksjons- og fabrikkeringsindustrien på 1920-tallet. Siden 1990-tallet har begrepet blitt stadig mer populært for å beskrive programvare. Dette er programvare som for eksempel automatiserer arbeidet med å fylle inn data i skjemaer, det fører til økt effektivitet og presisjon.

- *Kunstig intelligens*
  Kunstig intelligens handler om programvare som kan gjennomføre oppgaver som vanligvis krever menneskelig innblanding og intelligens. Programvare som bruker kunstig intelligens kan for eksempel gjenkjenne objekter ut fra bilder, planlegge for fremtiden eller gjenkjenne uvanlige eller mistenkelige oppføringer.

Disse tre teknologiene er for seg selv store teknologiske gjennombrudd innenfor automatisering, men RPA handler om å kombinere deler av disse teknologiene til å lage programvare som kan ha svært stor innvirkning på hvordan arbeidsoppgaver blir utført.

2.2 Maskinlæring og klassifisering

2.2.1 Maskinlæring

Det finnes flere typer maskinlæring, men her tar vi kun for oss en metode kalt veiledet læring. I tradisjonell programmering programmerer man en funksjon $f(x)$ der input x gir resultatet y. Ved veiledet læring innenfor maskinlæring lærer datamaskinen seg funksjonen $f(x)$ selv, gjennom å bli "matet" med par av observasjoner (x', y'), kalt treningsdata, der y' er

det korrekte resultatet av inputen x'. Gjennom optimalisering med mange sett av treningsdata vil de interne variablene i funksjonen justeres slik at en vilkårlig input x' gir et resultat som er tilnærmet likt y': $f(x') \approx$ y'

2.2.2 Klassifisering

Klassifisering i maskinlæringssammenheng handler om å plassere en gitt input x' i riktig klasse. Maskinlæringsmodellen i SkiltInfo er en slik klassifiseringsalgoritme. Den tar et bilde som input, og gir en skilttype id som output. Inputen til algoritmen er altså en kontinuerlig variabel som består av alle mulige bilder, mens outputen er en diskret variabel som består av mengden av id-er til skilttyper som er registrert i NVDB. Selve modellen kan sees på som en funksjon $f(x') \approx y'$, der x' er bildet og y' er en skilttype id.

2.3 GNSS og koordinatsystemer

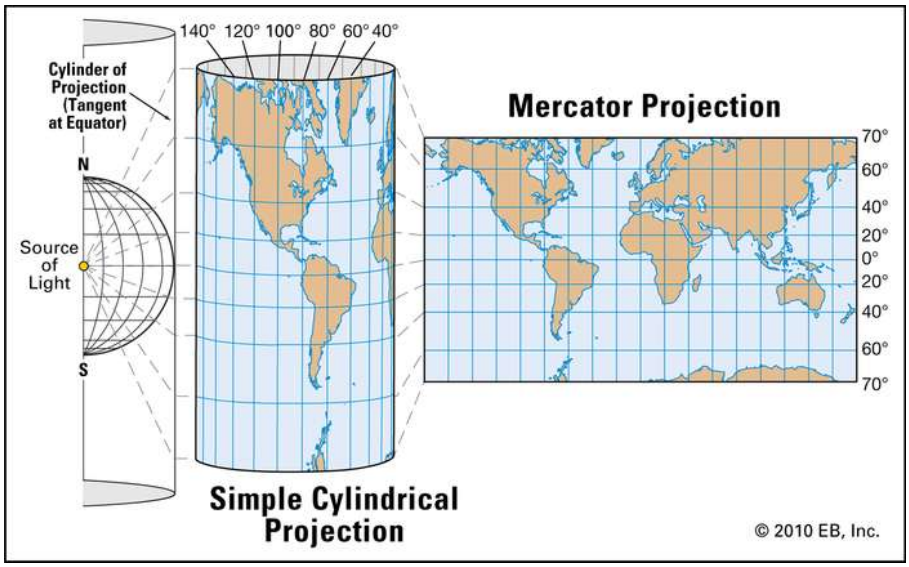
GNSS står for Global Navigation Satellite System og er fellesbetegnelsen for satellittnavigasjonssystemer. Det er fire globale systemer, amerikanske GPS, russiske GLONASS, kinesiske BeiDou og europeiske Galileo. Hvilke av systemene som brukes er avhengig av maskinvaren i telefonen. Det er mulig for telefoner å bruke flere GNSS. (Kartverket 2020)

GPS står for Global Positioning System og består av over 30 satellitter. Satellittene går i seks forskjellige baner, og derfor må det være minst 24 satellitter i bane rundt Jorden, slik at hvert punkt på Jorden skal bli sett av minst 4 satellitter til enhver tid.

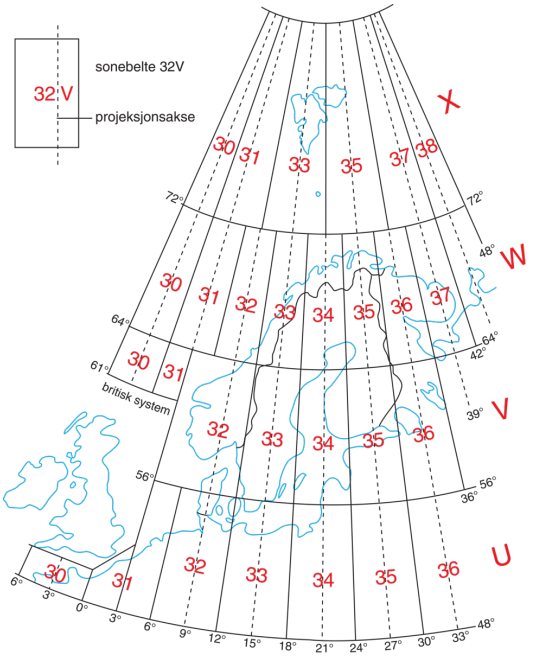
Det blir tatt i bruk 2 koordinatsystemer i dette prosjektet, WGS84 og EUREF 89. GNSS gir koordinater i WGS84 som er lengde- og breddegrader (latitude og longitude).

EUREF 89 er den regionale referanserammen for Eurasia. Dette blir gjengitt som plankoordinater N for nord og E for øst. Disse gjelder for Mercator-projeksjonen UTM, Universal Transversal Mercator. I en Mercator-projeksjon er de vertikale meridianene parallelle og jevnt fordelt, mens de horisontale meridianene er fordelt lenger fra hverandre jo lenger unna ekvator en er. Dette gjør at landmasser lengre unna ekvator vil se større ut enn det de virkelig er. I UTM deles kloden inn i soner på 6 lengdegrader, nummerert fra 1 til 60, og 8 breddegrader, fra C til X. Den første bokstaven i den nordlige halvkule er N, så alt før N vil være i den sørlige halvkulen.

I Norge brukes det UTM-32, 33 og 35. Informasjonen som blir brukt i databasen er skrevet i enten WGS84 eller UTM-33, hvor UTM-33 er standarden.



Figur 2: Mercator projeksjon



Figur 3: Oppdelingen av UTM soner

2.4 Systemutvikling

I systemutviklingsprosjekter brukes det mange forskjellige prinsipper og teknologier. Noen av de som er tatt i bruk i denne utviklingsprosessen er forklart her.

2.4.1 VCS

Et versjonskontrollsystem (engelsk: version control system, VCS), er et hjelpemiddel i utviklingsprosessen som gjør det enklere for utviklere jobbe på samme prosjekt samtidig. Et VCS har kodebase der alle nåværende og tidligere versjoner av koden er lagret, i tillegg til alle endringer (soumya08 2020). Før en utvikler kan gjøre endringer på den felles kodebasen må han først gjøre endringene lokalt på sin maskin og forsikre seg om at det ikke er noen feil eller problemer med koden.

2.4.2 Kontinuerlig Integrasjon

Kontinuerlig Integrasjon kan implementeres om et versjonskontrollsystem brukes i utviklingsprosessen. Ved bruk av kontinuerlig integrasjon vil koden kjøres gjennom en såkalt "build" hver gang en ny versjon sjekkes inn til versjonskontrollsystemet. Under en slik build vil koden verifiseres og testes på en virtuell maskin med det/de programmeringsspråkene og operativsystemene som er spesifisert. Dette gjøres ofte, helst daglig (Fowler 2006) slik at koden kontinuerlig testes og verifiseres, og eventuelle feil oppdages tidlig. Om det skulle oppstå feil, er de mest sannsynlig små og kan fikses enkelt før koden lastes opp til hoved-kodebasen, der alt skal være gjennomtestet og kjørbart.

2.4.3 Menneske Maskin Interaksjon (MMI)

Menneske Maskin Interaksjon handler om å lage datasystemer som er effektive, enkle og tilfredsstillende å bruke. Det er et multidisiplinært fagfelt som henter mye fra blant annet datateknologi og kognitiv psykologi (UIO 2010). Kjernen av MMI er hvordan mennesker interagerer med datasystemene de bruker; vet man noe om dette kan man bruke denne kunnskapen til å lage systemene enda bedre (Dix mfl. 2003). Innenfor fagfeltet finnes det en rekke designprinsipper og retningslinjer, flere av disse er tatt hensyn til under utviklingen av applikasjonen.

2.4.4 Universell Utforming

Universell utforming handler om å planlegge og designe produkter, omgivelser, programmer og tjenester slik at det kan brukes av flest mulig mennesker, målet er å unngå diskriminering på grunn av nedsatt funksjonsevne (Lid 2020). I Norge er det lovpålagt at IKT systemer skal være universelt utformet om systemet oppfyller spesielle krav. Når det gjelder mobilapplikasjoner er de underlagt dette lovverket om den må laste ned oppdatert informasjon over internett for å fungere, applikasjonen regnes da som en nettapplikasjon. Det vil si at applikasjonen *SkiltInfo* er underlagt dette lovverket dersom den skal offentliggjøres. Det finnes flere krav og anbefalinger for universell utforming av mobile applikasjoner, disse finner man beskrevet i Retningslinjer for tilgjengelig webinnhold (WCAG) 2.0 (Difi 2019)

2.4.5 Testing og Test Driven Development

I boken *The Art of Software Testing* (Myers, Sandler og Badgett 2011) defineres testing av programvare slik: "Testing is the process of executing a program with the intent of finding errors". Altså er ikke hovedmålet med testing, ifølge denne boken, å vise at koden er feilfri, eller å vise at koden fungerer som den skal. Testing skal heller brukes som et verktøy for å finne så mange feil som mulig, så tidlig som mulig. Derfor er det mange som praktiserer en utviklingsmetode kalt Test Driven Development (TDD).

TDD er en utviklingsmetode der man utvikler programvare i svært korte iterasjoner. Prosessen krever at utvikleren skriver automatiske tester for det som skal implementeres, før koden som skal gi funksjonaliteten skrives (Janzen og Saiedian 2005). Alle testene som skrives vil gi feil i begynnelsen, utvikleren vet at riktig funksjonalitet er oppnådd når testen består, og det ikke går utover noen av de andre testene. TDD syklusen ser slik ut (Beck 2003):

1. *Skriv en test*
   I TDD starter all ny funksjonalitet med å skrive en test. Nøkkelen med å skrive testen før koden er at det tvinger utvikleren til å tenke over kravene som skal oppfylles før koden skrives.

2. *Kjør alle testene og sjekk at den nye testen feiler*
   Hvis testen feiler er man sikker på at testen ikke kan bestås uten å skrive ny kode, og utelukker muligheten for at det er noe feil med testen slik at den vil bestå uansett.

3. *Skriv kode*
   Her skal utvikleren skrive kode som gjør at testen bestås. Denne koden trenger ikke å være optimal eller elegant, det tas hånd om i steg 5.

4. *Kjør testene*
   Hvis alle testene bestås kan utvikleren være sikker på at den nye koden oppfyller kravene og ikke ødelegger noe av den eksisterende funksjonaliteten. Hvis noen av testene ikke bestås må den nye koden skrives om til de gjør det.

5. *Refaktorer koden*
   I dette steget skal det renses opp i den nye koden. Koden kan for eksempel flyttes fra der det var enkelt å plassere den når den bare skulle bestå en test, til det stedet i koden hvor den hører til. Et annet viktig punkt i TDD er å unngå duplisering av kode, det er i steg 5 det sørges for at dette ikke forekommer. I dette steget kan man også endre navn på objekter, klasser, variabler og metoder slik at de representerer sin egen hensikt eller funksjonalitet.

6. *Gjenta*
   Nå kan det ny funksjonalitet implementeres ved å skrive nye tester og gjenta stegene.

3 Valg av Teknologi og Metode

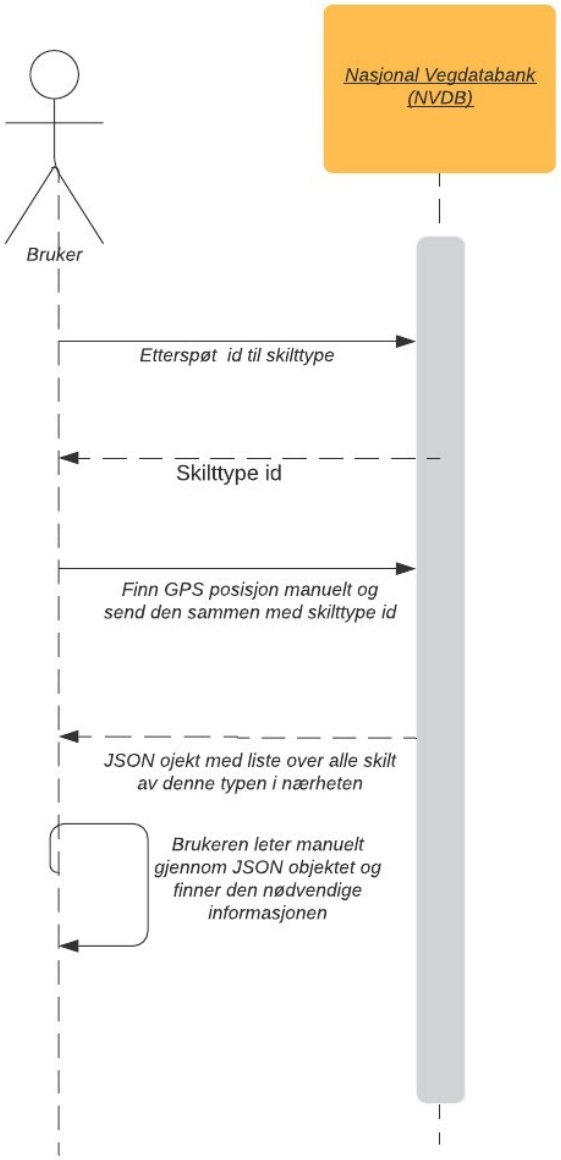
Dette kapittelet omhandler teknologiene gruppen har brukt for å utvikle applikasjonen og hvordan de fungerer. Utviklingsmetoden brukt under utvikling av applikasjonen forklares til slutt i kapittelet.

3.1 Prosessautomatisering i *SkiltInfo*

RPA i applikasjonen handler om å bruke maskinlæring og bildegjenkjenning til å gjenkjenne riktig type skilt, og deretter bruke dette, sammen med enhetens posisjon til å slå opp mulige skilt. Vi skal her se på prosessen med å finne frem til et tilfeldig skilt man ser langs veien både med og uten bruk av RPA.

3.1.1 Uten RPA

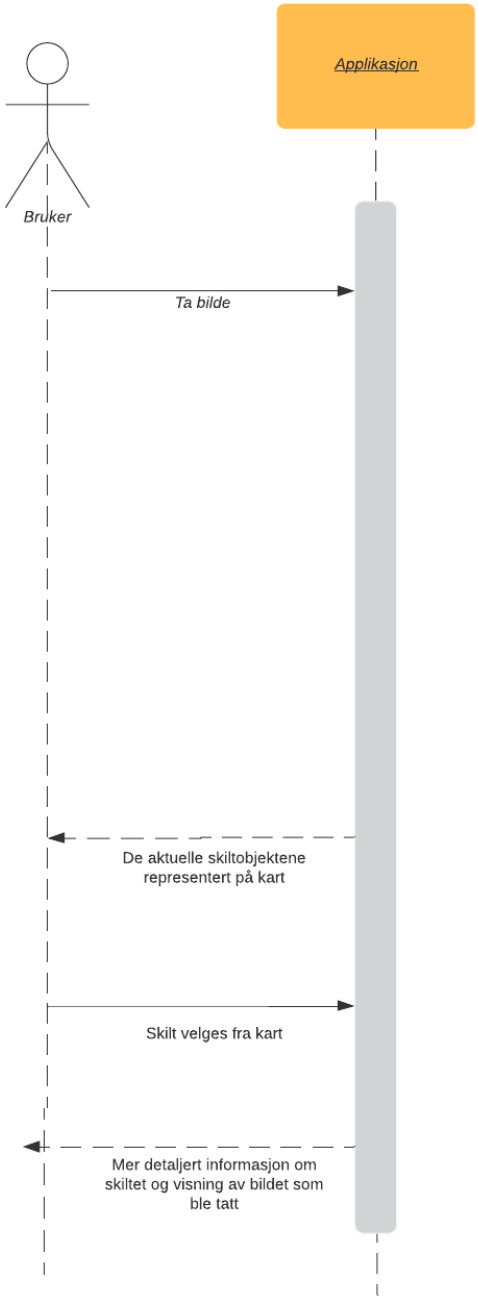
Dette er den mest primitive måten å gjøre det på, og slik man måtte gjort det om man ikke hadde noen eksisterende applikasjoner tilgjengelig. Man må her gjøre to manuelle kall til databasen, for deretter å lese gjennom resultatene og finne den informasjonen man er ute etter.



Figur 4: Informasjonsflyten uten bruk av RPA

3.1.2 Med RPA

Dette er løsningen i applikasjonen, her trenger brukeren kun å ta et bilde av skiltet, og velge et av de aktuelle skiltene fra listen i brukergrensesnittet. GPS posisjon og skilttype finnes automatisk, og oppslagene i databasen gjøres også automatisk. Utvidet versjon av dette sekvensdiagrammet finnes i vedlegget *kravdokumentasjon*



Figur 5: Informasjonsflyten med bruk av RPA

3.2 Standarder

3.2.1 JSON

JSON står for JavaScript Object Notation og er et dataformat som brukes for det meste i webapplikasjoner. Det er lett for både mennesker og maskin å lese og skrive, og ble standardisert i 2017.

3.2.2 Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol er en kommunikasjonsprotokoll på applikasjonsnivået som har vært i bruk av World-Wide web global information initiative siden 1990. I tillegg til at det er mulig å forespørre ren informasjon ved bruk av HTTP, finnes det "headers" som gjør det mulig å si noe om hensikten med forespørselen (Fielding mfl. 1999). Disse "headerene" kan for eksempel si noe om hvordan forbindelsen skal håndteres, autentisering, caching eller cookies. HTTP er en tilstandsløs protokoll, det vil si at hver forespørsel som sendes kan forstås av serveren helt uavhengig av statusen til klienten eller forespørsler som er sendt tidligere.

Ulempen med HTTP er at data som sendes ikke blir kryptert, slik som i HTTPS der forespørslene og svarene krypteres med TLS. Valget vårt falt fremdeles på HTTP fordi *SkiltInfo* ikke håndterer konfidensiell informasjon eller sensitive opplysninger, det vil derfor ikke ha konsekvenser om en eventuell angriper skulle plukke opp det som sendes eller mottas av applikasjonen. I tillegg er HTTP en del enklere å implementere.

3.2.3 Representational State Transfer (REST)

Representational State Transfer er en standard for kommunikasjon mellom datasystemer på web. Systemer som oppfyller denne standarden sies ofte å være RESTful. Slike systemer kjennetegnes ved at de er tilstandsløse; klienten og serveren kjører uavhengige av hverandre. Det vil si at klienten og serveren ikke trenger å vite om hverandre, og hvis koden i en av dem endres er det ikke nødvendig å endre koden i den andre også (CodeAcademy 2019). Hvem som helst kan lage applikasjoner som kommuniserer med de samme REST endepunktene i NVDB. Klientene vil da sende de samme forespørslene og motta de samme svarene, det er måten forespørslene utformes på og hvordan svaret håndteres i klienten som skiller dem fra hverandre

3.3 Konvertering mellom lat/long og N/E

I WGS84 avhenger lengdegraden av breddegraden, så avstanden mellom to lengdegrader vil ikke være det samme i forskjellige breddegrader. Dette gjør at avstand mellom to punkter må regnes ut ettersom verdiene er oppgitt i grader. Hver grad er delt opp i 60 minutter som igjen er delt opp i 60 sekunder. Et eksempel på en grad er 12°34°56°. I UTM er 1 verdi 1 meter, og sentrum av hver sone er sann nord. I applikasjonen er både konvertering fra N/E til

bredde- og lengdegrader, og motsatt, nødvendig. Dette er fordi posisjon i NVDB lagres som N/E, men koordinatene som brukes av kartet i applikasjonen og koordinatene som returneres ved bruk av GPS er bredde- og lengdegrader. Konverteringen i backend tar Java biblioteket proj4j seg av. I frontend gjøres konverteringen med JavaScript funksjoner som er hentet fra et konverteringsverktøy skrevet av Charles L. Taylor (Taylor 2017)

3.4 Systemutviklingteknologi og -metode

3.4.1 React og React Native

React.js er et JavaScript frontend rammeverk for å lage brukergrensesnitt. For seg selv er React.js et plattform uavhengig rammeverk, det kan altså, i teorien, anvendes til alle plattformer. Det er mest brukt til å lage web applikasjoner; da brukes React til å bygge opp selve brukergrensesnittet, og ReactDOM brukes for å ta hånd om de web-spesifikke tingene som gjengivelse av komponentene i nettleseren.

React Native er et separat bibliotek som er bygget på React.js og gir utvikleren tilgang til en del spesielle React komponenter som kan kompileres til "native" komponenter til Android og iOS. Det vil si at React Native koden vil kompileres til Java kode for Android eller Swift for iOS når applikasjonen kjøres. Det gjør det mulig å utvikle raske og responsive apper som ser bra ut til begge plattformer.

3.4.2 Expo

Når et nytt React Native prosjekt opprettes, må det velges om det skal bruke Expo CLI eller React Native CLI. Expo er et rammeverk til React Native applikasjoner som forenkler utviklingsprosessen på flere måter. For eksempel trenger ikke utvikleren å bruke Android Studio eller XCode for å publisere applikasjonen, Expo gir også tilgang til mange av enhetens funksjoner, for eksempel kamera, sensorer, stedstjenester og varslinger. Ulempen med å bruke Expo er at en har litt mindre frihet når det kommer til å utvikle plattform-spesifikk funksjonalitet, men for de fleste applikasjoner er ikke dette noe problem.

React Native CLI utvikles av React Native Teamet og gir utvikleren et "rent" utviklingsmiljø med mange valgmuligheter og en mer komplisert utviklingsprosess. Om det er behov for mye fleksibilitet og det skal utvikles en komplisert applikasjon som må tilpasses mye til begge plattformer burde dette brukes.

Vårt valg falt på Expo fordi vi ikke har behov for mulighetene som React Native CLI tilbyr. I tillegg er Expo enklere å bruke, noe som er praktisk for oss da vi ikke hadde utviklet applikasjoner i React Native tidligere.

3.4.3 JavaScript Testing Framework (Jest)

Jest er et rammeverk for testing av applikasjoner laget med JavaScript. Det er også tilpasset flere populære JavaScript rammeverk, blant annet React Native. I SkiltInfo brukes Jest for å teste frontend med snapshot tester og enhetstester. Snapshot tester er tester som lager output filer av komponenter hver gang testen kjøres, og sammenligner med output filen som ble generert sist testen ble kjørt. Hvis output filene er forskjellige vil testen feile. Hvis komponenten endres med vilje kan output filen enkelt oppdateres. Enhetstestene som kjøres er enkle tester som sjekker at komponentene har "riktig" antall barn.

3.4.4 JUnit

JUnit er et åpen-kildekode enhetstesting bibliotek til Java. Enhetstesting brukes for å verifisere små deler av koden og sjekke at de fungerer, ofte funksjoner eller metoder. I SkiltInfo brukes JUnit for testene på backend delen.

3.4.5 Java Code Coverage (JaCoCo)

JaCoCo blir brukt for å generere en rapport om dekningsgraden i koden for Java-prosjekter. JaCoCo analyserer testene for å sjekke dekningsgraden i koden. Rapporten blir laget i binærformat i filen jacoco.exec, som igjen kan konverteres til mer lettleselige formater som html og xml.

JaCoCo måler linjedekning, grendekning og kompleksitet. Linjedekning er hvor mange av linjene i koden dekkes av testene. Grendekning er hvor mange grener som er testet. Disse grenene er if/else og switch, hvor det blir sjekket om testene tester for disse utsagnene.

Kompleksiteten sier hvor mange runder med tester som må kjøres for å få full kodedekning. Eksempelvis vil en kode uten if/else og switcher ha en kompleksitet på 1, ettersom det ikke er noen grener som blir utelatt ved å kjøre testen en gang.

Dekningsgraden i koden presenteres i prosent.

3.4.6 Git

Git er det det klart mest brukte versjonskontroll systemet i dag. Git er et DVCS, Distributed Version Control System, det vil si at hver utviklers kopi av koden også er sin egen kodebase der full historikk av alle endringer lagres. Det sikrer at programvaren alltid kan tilbakerulles til et hvilket som helst tidspunkt.

I tillegg er Git godt integrert i IDEene som er brukt i dette prosjektet, noe som forenkler bruken.

3.4.7 Travis-CI

Travis CI er en kontinuerlig integrasjons tjeneste som kan brukes for prosjekter som ligger på GitHub. Travis CI konfigureres ved å legge til en fil i mappetreet som heter .travis.yml. I denne filen spesifiseres språket koden er skrevet i, hva slags operativsystem(er) koden skal testes på, og eventuelle andre scripts som utvikleren vil ha kjørt når koden skal testes. For eksempel, for å teste java koden i SkiltInfo må java filene kompileres først. Det gjøres med et script. Når en "build" er fullført med Travis CI vil utvikleren som lastet opp koden få en epost om builden bestod eller feilet slik at det kan fikses før det jobbes videre.

3.5 Agile metoder og Scrum

I motsetning til tradisjonelle metoder som vannfallsmetoden, der hele prosjektet planlegges i sin helhet før det praktiske arbeidet starter, er agile metoder tilpasset til å anvendes ved prosjekter der kravene endres mye underveis (Cohen, Lindvall og Costa 2004).

Mange agile metoder, eksempelvis Scrum, er basert på iterative prosesser. Iterative prosesser består av flere mindre prosjekter som kalles iterasjoner (Systemutviklingsprosesser, hentet 27.02.2020). Utviklingsprosjekter som bruker Scrum som metode kaller iterasjonene for sprinter, der hver sprint består av flere delmål, ofte kalt brukerhistorier. Hver sprint har sin egen planleggings, utviklings og testfase, dette gir rom for å tilpasse produktet underveis til det brukeren trenger.

3.6 Arbeids- og rollefordeling

Gruppen har jobbet sammen ved flere anledninger gjennom studieløpet, både på utviklingsprosjekter og annet arbeid. Vi valgte derfor å jobbe sammen under prosjektet fordi vi kjenner hverandre godt, både faglig og personlig.

Patrick Thorkildsen fikk hovedansvaret for utviklingen av frontend på grunn av litt tidligere kunnskap med frontendrammeverket *React Native*. Quan Tran hadde ansvar for utvikling av backend i Java på grunn av gode kunnskaper med dette fra tidligere prosjekter. Selv om hovedansvar var delt inn jobbet begge med alle deler av prosjektet. For eksempel ble utviklingen av backend ferdig tidligere enn forventet, og begge forkuserte da på frontendutvikling.

4 Resultater

Her presenteres resultatene med bakgrunn i problemstillingen i kapittel 1.2. Kapittelet er tredelt; de vitenskapelige resultatene tar utgangspunkt i sluttproduktet, de ingeniørfaglige resultatene omhandler mål og krav satt til prosjektet, i tillegg til resultater fra tester gjort underveis. De administrative resultatene tar utgangspunkt i fremdriftsplanen og timelisten, og resultater fra utviklingsmetodikken.

4.1 Vitenskapelige resultater

I dette delkapittelet vil de vitenskapelige resultatene presenteres. De er basert på sluttresultatet av produktet, som innebærer serveren, applikasjonen og applikasjonens design. Resultatene presenteres med utgangspunkt i problemstillingen i kapittel 1.2

Produktet deles inn i to deler; server og klient. Her tar vi for oss disse separat.

4.1.1 Server

Serveren er skrevet i java med biblioteket httpserver av Oracle. Serveren mottar en forespørsel om alle skilt av en spesifikk type i et gitt område fra klienten, og sender en ny forespørsel med parametrene fra klientens forespørsel til NVDB. Svaret fra NVDB videresendes fra serveren til klienten.

4.1.2 Klient

Klienten er skrevet i javascript med React og react-native. Klienten sender en forespørsel til serveren med disse parametrene når et bilde blir tatt:

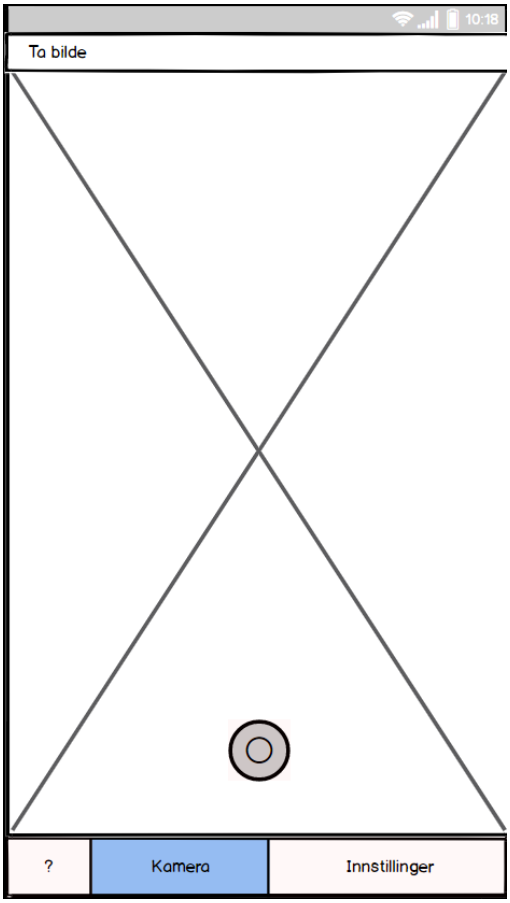
- Skilt-id: Id til skilttypen brukeren vil få svar om. Eksempler på skilttyper er forkjørsveg og gangfelt.
- GPS-lokasjon: Posisjonen brukeren er i når bildet blir tatt i latitude og longitude verdier.
- Radius - Avstanden fra brukeren til sidene på boksen som avgrenser området brukeren vil ha svar om i meter.

Klienten mottar svaret fra serveren og leser gjennom alle skiltene det fikk tilsendt og finner posisjonen til skiltene. Disse skiltene blir så plassert på et kart slik at brukeren kan se og velge hvilket skilt de vil ha mer informasjon om.

For lagring av innstillinger og filtre er ASyncStorage brukt. Innstillinger er hva slags type veiskilt det skal søkes på, radius på søket og hvilket filter som er aktiv. Filtre bestemmer hva slags informasjon om det valgte skiltet skal vises.

4.1.3 Design

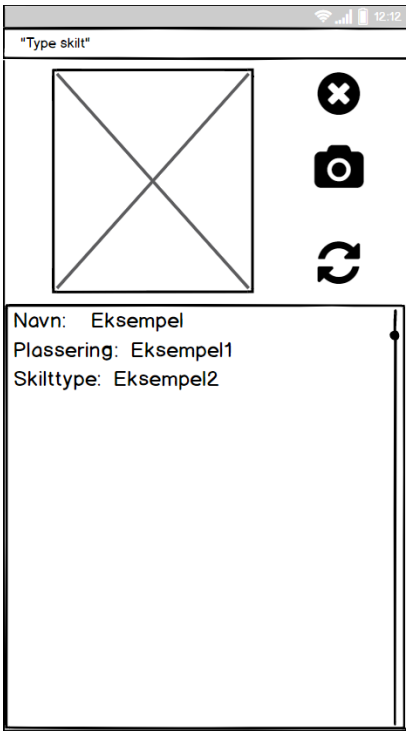
Designet av applikasjonen startet med en wireframe som fungerte som utgangspunkt for utviklingen av design. Senere i prosjektet ble det laget noen mer detaljerte designforslag for å få et mer profesjonelt design. Her skal vi vise hvordan noen av sidene i applikasjonen så ut i planleggingen, og hvordan de så ut til slutt. De første wireframene ble laget med Balsamiq, og de mer detaljerte designforslagene ble laget i Adobe Illustrator.



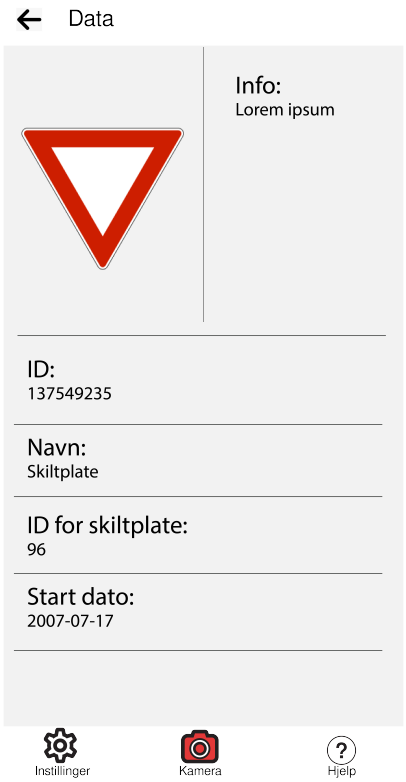
(a) Kameraskjermen i første wireframe

(b) Kameraskjermen i sluttprodukt

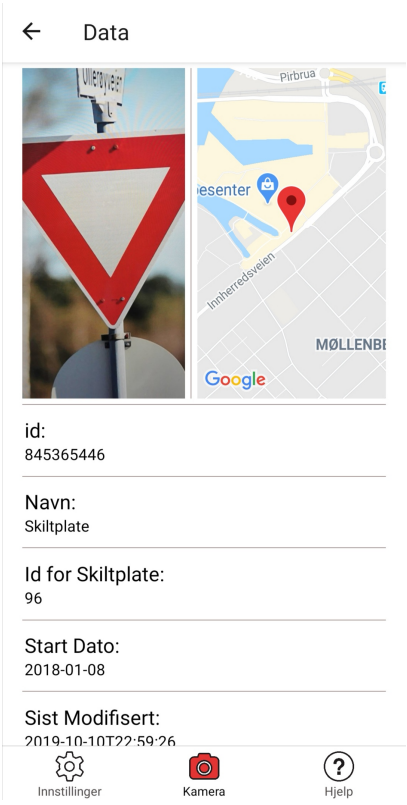
Figur 6: Kameraskjermen i wireframe og sluttprodukt



(a) Skjerm for visning av data i første wireframe

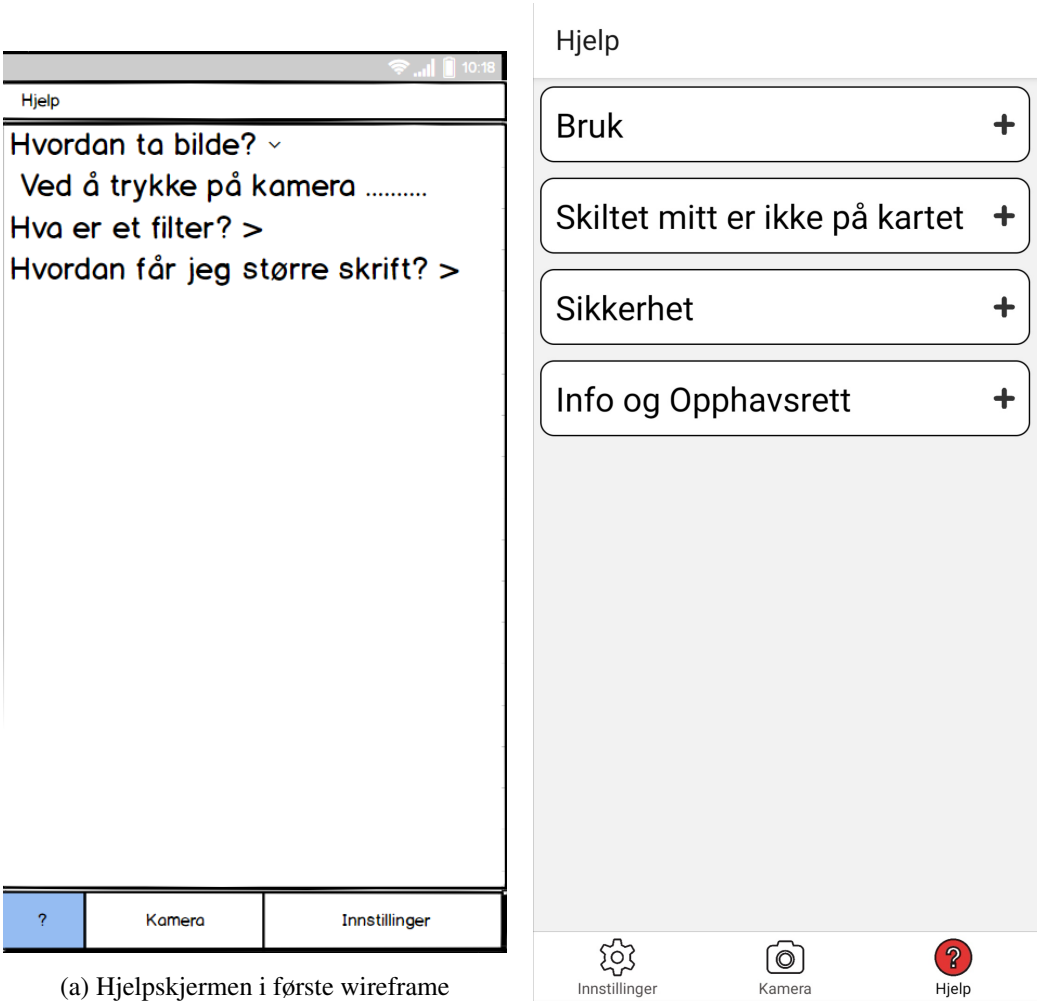


(b) Skjerm for visning av data i andre wireframe



(c) Skjerm for visning av data i sluttprodukt

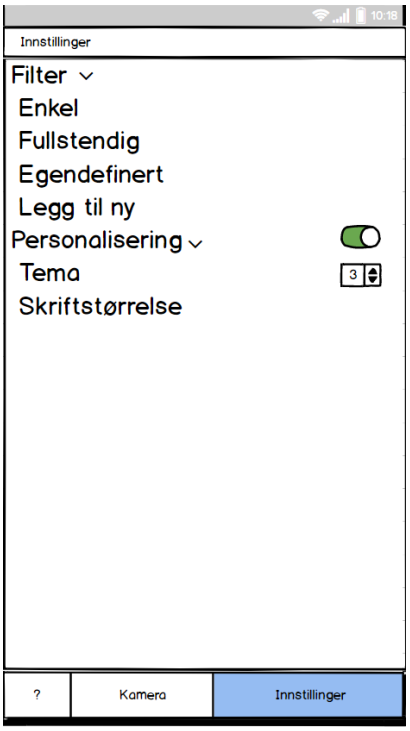
Figur 7: Skjerm for visning av data i wireframe og sluttprodukt



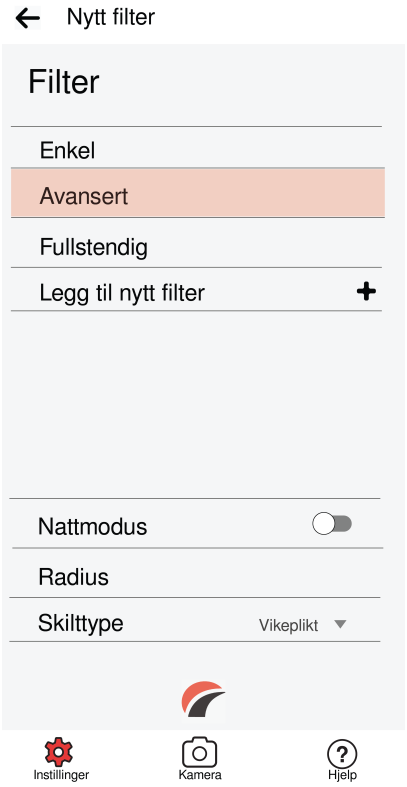
(a) Hjelpskjermen i første wireframe

(b) Hjelpskjermen i sluttprodukt

Figur 8: Hjelpskjermen i wireframe og sluttprodukt



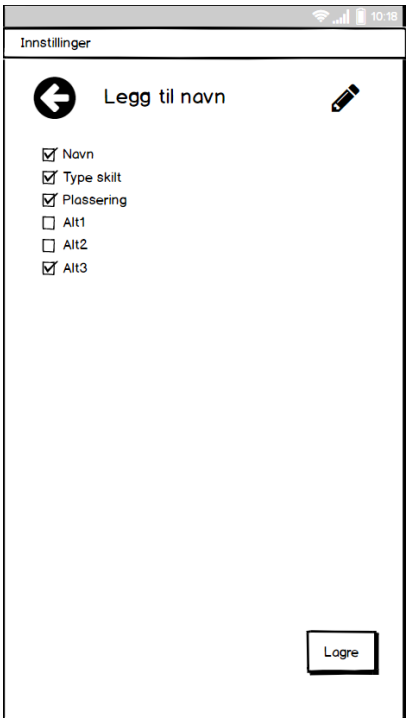
(a) Instillingskjermen i første wireframe



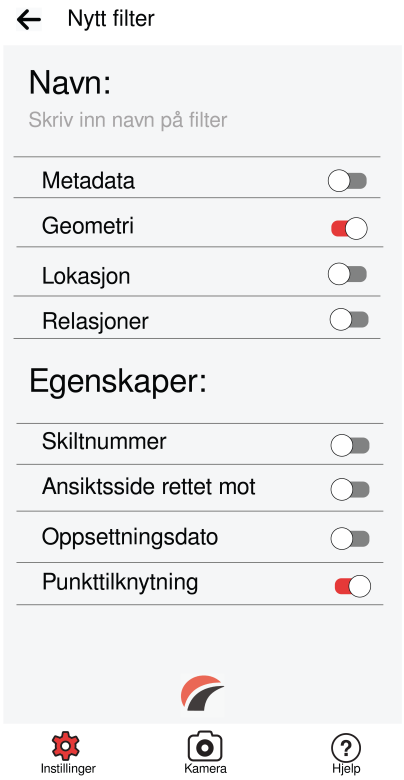
(b) Innstillingskjermen i andre wireframe

(c) Innstillingskjermen i sluttprodukt

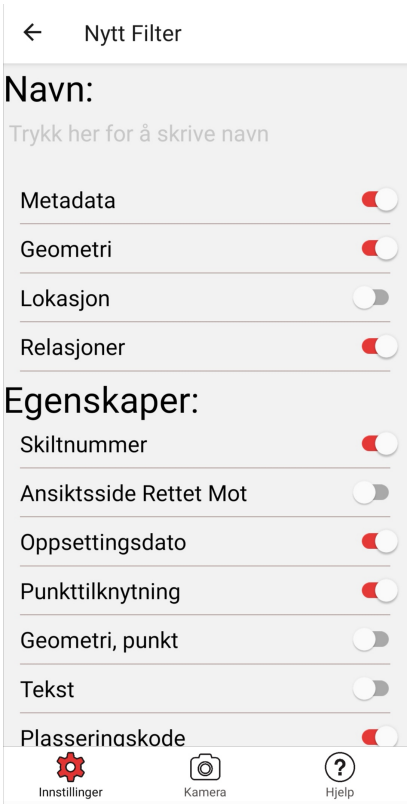
Figur 9: Instillingskjermen i wireframe og sluttprodukt



(a) Filterskjermen i første wireframe



(b) Filterskjermen i andre wireframe



(c) Filterskjermen i sluttprodukt

Figur 10: Filterskjermen i wireframe og sluttprodukt

4.2 Ingeniørfaglige resultater

De ingeniørfaglige resultatene for prosjektet er basert på målene som ble satt i starten og underveis i prosjektet, samt testene av koden. Målene er hentet fra visjonsdokumentet (referanse her) under kapitlene om funksjonelle- og ikke-funksjonelle egenskaper.

4.2.1 Prosjektmål

Målene beskrives med utgangspunkt i begrunnelse for målet, hvordan det var tiltenkt ved starten av prosjektet, og status på målet ved innlevering av prosjektet. De er sortert etter hvordan de har vært prioritert underveis.

1 **Mulighet for å integrere bildegjenkjenningsmodul i applikasjonen som bruker bildet som er tatt**
Dette er kjernefunksjonaliteten av produktet. Bildet skal først sendes til en bildegjenkjenningsmodul som returnerer hva slags skilt det er tatt bilde av. Deretter skal dette, sammen med GPS posisjon brukes til å gjøre oppslag i NVDB og finne skiltet det er tatt bilde av.

   Ved innleveringstidspunktet er ikke bildegjenkjenningsmodulen implementert grunnet at vi ikke har fått tilgang til den av Triona AS. Likevel har vi fått informasjon om hvordan den fungerer, og har simulert modulen slik at typen skilt som "gjenkjennes" er forhåndsbestemt. Det er også lagt inn funksjonalitet for å velge hvilken skilt som skal "gjenkjennes" i innstillingene for applikasjonen. Det er lagt til rette for at bildegjenkjenningsmodulen skal kunne implementeres når som helst uten å måtte endre for mye kode. I tillegg lagres bildet som tas midlertidig på enheten, som gjør det mulig å sende bildet videre til modulen.

2 **Kart over skilt som kan velges om det er flere skilt av samme type i nærheten**
GPS-lokasjonen til mobiler har en viss feilmargin. Feilmarginen avhenger av hvor mange GPS-satellitter som er tilgjengelige og hvor man befinner seg, om man står under noe eller i et tett bebygd område vil dette påvirke gps-signalets nøyaktighet. En annen grunn til at dette er viktig er at skilt av samme type ofte befinner seg i nærheten av hverandre. For eksempel vil fotgjengerovergang skilt som regel være plassert i par; et på hver side av veien. Dette gjør det nødvendig å ha en brukervennlig måte å kunne velge skilt på når det returneres flere skilt fra NVDB.

   Vi startet med å implementere muligheten for å velge skilt med en liste over alle skiltene som ble returnert, men dette var lite brukervennlig. Etter innspill fra oppdragsgiver besluttet vi å implementere et kart i stedet, der skiltene kommer opp som markører på kartet som kan velges for å se mer detaljert informasjon.

3 **Filtrering av informasjonen som skal vises med både egendefinerte og forhåndsdefinerte filtre**

Dataen som ligger i NVDB om hvert enkelt skilt er svært omfattende, og all informasjonen er ikke like interessant for alle brukere av appen. Det er derfor behov for at informasjonen som hentes ut om skiltene er filtrert etter det brukeren ønsker.

Dette ble løst ved å la brukeren opprette så mange egne filtre han/hun ønsker, der de velger hva slags informasjon som skal vises når det søkes med hvert filter. Det ble også lagt til tre forhåndsdefinerte filtre, kalt "Enkel", "Avansert", og "Fullstendig". Førstnevnte henter kun ut den viktigste informasjonen, "Avansert" henter ut mer, og "Fullstendig" henter ut all tilgjengelig informasjon som er lagret om skiltet.

4 Applikasjonen skal fungere til både Android og iOS

Som nevnt i kapittel 3.4.1 er React Native et rammeverk som gjør det mulig å skrive kode som kan kompileres til både Java og Swift, og dermed kan kjøres på både Android og iOS. I praksis er det fremdeles sjeldent man kan lage et helt system uten å gjøre noen plattform spesifikke justeringer. Det er fordi Android og iOS ikke alltid støtter de samme funksjonene og bibliotekene.

Ved innleveringstidspunktet er applikasjonen ikke testet på iOS

5 Lokal lagring av innstillinger

Innstillingene i applikasjonen omfatter oppretting, sletting, redigering og valg av filter, hvor stort radius søket er fra GPS-lokasjon, i tillegg til valg av skilttype som skal søkes etter, som er midlertidig frem til applikasjonen blir koblet opp mot bildegjenkjenningsmodulen til Triona AS. For at endring av innstillinger skal være brukervennlig er det viktig at brukeren ikke må endre de samme innstillingene hver gang applikasjonen brukes. De aktive innstillingene bør derfor lagres lokalt på mobilen, slik at de er like neste gang applikasjonen åpnes.

Ved innlevering av prosjektet er dette implementert i appen. Forhåndsinnstillingene er lagret i JSON-objekter som blir lest første gang applikasjonen kjøres og kan ikke bli endret på. Filtrene som blir opprettet, aktiv filter, radius og valg av skilttype blir lagret i AsyncStorage. På Android lagres disse innstillingene i en database med en nøkkel og en verdi, mens i iOS lagres nøkler med små verdier i en "ordliste" (dictionary), og store verdier i separate filer.

6 Enhetstesting og CI skal være implementert

Implementering av enhetstesting og CI sikrer en mer stabil utviklingsprosess og minker sjansen for store feil som tar lang tid å rette opp i.

På grunn av dette ble både enhetstester og CI implementert tidlig, noe som førte til at vi kunne støtte oss på resultatene fra testene under mesteparten av utviklingsprosessen.

7 Side for å få hjelp om bruk av applikasjonen og generell info om applikasjonen

For å gjøre applikasjonen enkel å bruke ønsket vi mulighet for å få informasjon om

applikasjonen uten å måtte lete i eksterne ressurser. Dette omfatter informasjon om hvordan applikasjonen fungerer, hvordan man bruker den, og opphavsrett.

Dette ble løst som en egen side i applikasjonen med en slags FAQ liste.

8 **Tilpasse utseende av applikasjonen**
Da dette målet ble satt så vi for oss mulighet for å kunne endre mellom mørkt og lyst tema og å kunne endre skriftstørrelse.

Ved levering av prosjektet er denne funksjonaliteten enda ikke utviklet.

Som sagt er målene hentet fra visjonsdokumentet, men ikke alle er skrevet ordrett fra det. I listen er også noen av målene kombinert til samme punkt for å gjøre det mer oversiktlig.

4.2.2 Brukertester

Brukertester ble ikke tatt av arbeidere i veisektoren grunnet utbruddet av Covid-19. På grunn av dette ble det istedet tatt brukertester med personer i samme husholdning som prosjektdeltakerne.

Testen som ble tatt gikk ut på å sammenligne vegvesenet sitt *vegkart* med applikasjonen. Testdeltakerne fikk i oppgave å finne to forskjellige vikepliktskilt i en gitt rundkjøring som var kjent for deltakerne fra før av, først med vegkartet til vegvesenet, og deretter med *SkiltInfo*. Det ble benyttet forskjellige skilt på de to testene, men begge lå i den samme rundkjøringen. Når skiltet ble funnet skulle de finne datoen skiltet ble satt opp på og fortelle det til testadministratoren.

Testadministrator registrerte tiden som ble brukt fra testen startet frem til brukeren fant frem til oppsettingsdatoen for skiltet. I tillegg ble brukeren bedt om å rangere hvor lett det var å finne frem til skiltet på en skala fra 1 til 10, der 1 er veldig vanskelig og 10 er veldig enkelt. Ingen av testdeltakerne hadde brukt vegkartet eller applikasjonen tidligere, de fikk heller ingen innføring i å bruke systemene. Hvis deltakerne satt fast etter ett minutt fikk de tips fra testadministrator. Resultatene er presentert i Tabell 1.

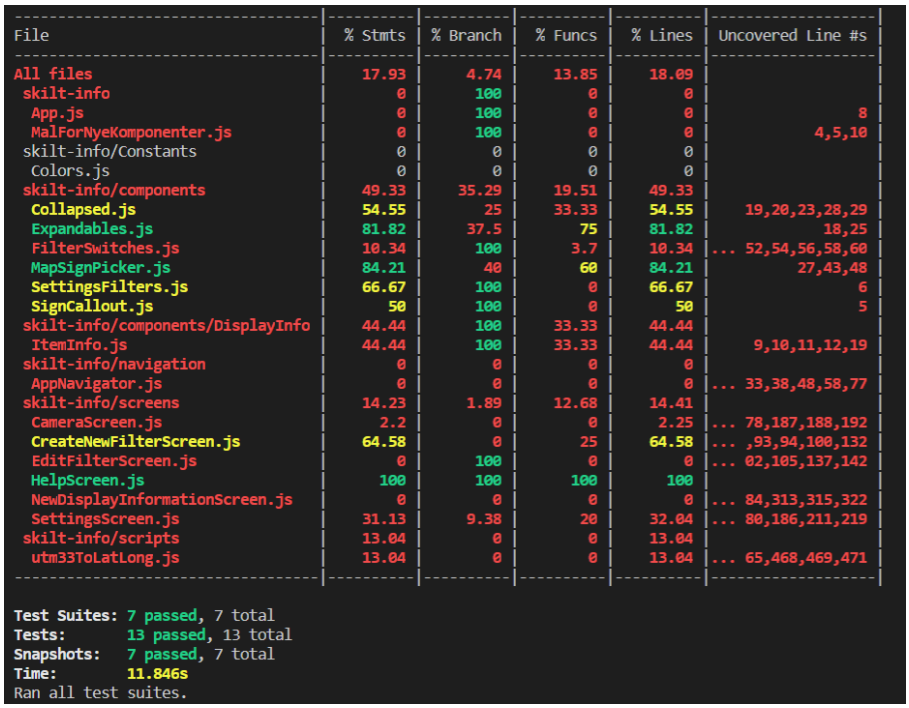
Brukertester				
Deltaker	Tid på vegkart	Tid på *SkiltInfo*	Enkelhetrangering *Vegkart*	Enkelhetrangering *SkiltInfo*
Deltaker 1	1 min, 49 sek	25 sek	4/10	9/10
Deltaker 2	2 min 37 sek	35 sek	6/10	9/10
Deltaker 3	2 min, 14 sek	38 sek	4/10	9/10
Deltaker 4	2 min, 19 sek	56 sek	5/10	9/10

Tabell 1: Resultater fra brukertester

4.2.3 Kodetester

På frontend er det kjørt snapshot tester og tester som sjekker at komponentene har riktig antall barn på alle komponentene som det lot seg gjøre på. Det er ikke gjort enhetstester på frontend. Totalt ligger testdekningen på omtrent 18%. Mer detaljer rundt testene med Jest finnes i 3.4.3.

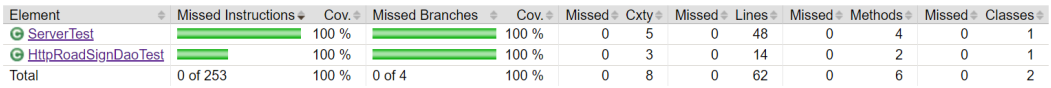
På backend er det gjort en integrasjonstest opp mot databasen, enhetstester av alle metodene og en servertest mot en mock-server. En integrasjonstest er avhengig av at databasen er oppe og at dets innhold ikke er endret siden integrasjonstesten ble skrevet.



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	17.93	4.74	13.85	18.09	
skilt-info	0	100	0	0	
App.js	0	100	0	0	8
MalForNyeKomponenter.js	0	100	0	0	4,5,10
skilt-info/Constants	0	0	0	0	
Colors.js	0	0	0	0	
skilt-info/components	49.33	35.29	19.51	49.33	
Collapsed.js	54.55	25	33.33	54.55	19,20,23,28,29
Expandables.js	81.82	37.5	75	81.82	18,25
FilterSwitches.js	10.34	100	3.7	10.34	... 52,54,56,58,60
MapSignPicker.js	84.21	40	60	84.21	27,43,48
SettingsFilters.js	66.67	100	0	66.67	6
SignCallout.js	50	100	0	50	5
skilt-info/components/DisplayInfo	44.44	100	33.33	44.44	
ItemInfo.js	44.44	100	33.33	44.44	9,10,11,12,19
skilt-info/navigation	0	0	0	0	
AppNavigator.js	0	0	0	0	... 33,38,48,58,77
skilt-info/screens	14.23	1.89	12.68	14.41	
CameraScreen.js	2.2	0	0	2.25	... 78,187,188,192
CreateNewFilterScreen.js	64.58	0	25	64.58	... ,93,94,100,132
EditFilterScreen.js	0	100	0	0	... 02,105,137,142
HelpScreen.js	100	100	100	100	
NewDisplayInformationScreen.js	0	0	0	0	... 84,313,315,322
SettingsScreen.js	31.13	9.38	20	32.04	... 80,186,211,219
skilt-info/scripts	13.04	0	0	13.04	
utm33ToLatLong.js	13.04	0	0	13.04	... 65,468,469,471

Test Suites: 7 passed, 7 total
Tests: 13 passed, 13 total
Snapshots: 7 passed, 7 total
Time: 11.846s
Ran all test suites.

Figur 11: Testdekningen i frontend



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ServerTest		100 %		100 %	0	5	0	48	0	4	0	1
HttpRoadSignDaoTest		100 %		100 %	0	3	0	14	0	2	0	1
Total	0 of 253	100 %	0 of 4	100 %	0	8	0	62	0	6	0	2

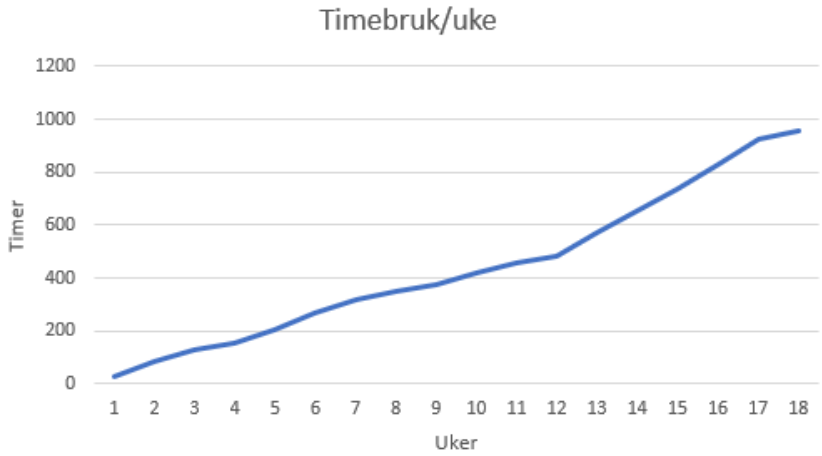
Figur 12: Testdekningen i Backend

4.3 Administrative resultater

Dette kaptittelet tar utgangspunkt i Gantt diagrammet og timelistene fra prosjekthåndboka som ligger som vedlegg. I tillegg beskrives noen detaljer rundt utviklingsprosessen. Mer detaljert informasjon finnes i prosjekthåndboka.

4.3.1 Timebruk

Alle timene som ble brukt i prosjektet ble ført på en aktivitet fra fremdriftsplanen. Totalt skulle hver person ha omtrent 500 timer hver, totalt 1000 timer for dette prosjektet. Dette målet ble oppnådd med en sammenlagt total timebruk på 954 timer. Figur 13 viser timebruket over prosjektets 18 uker.



Figur 13: Timebruk over prosjektperioden

4.3.2 Fremdriftsplan (Gantt diagram)

Hele prosjektet ble delt inn i oppstart, 4 sprinter, og sluttperiode som var satt av til å jobbe med dokumentasjon og denne rapporten. Oppstartsperioden innebar å sette opp første versjon av fremdriftsplanen, samarbeidskontrakt, visjonsdokumentet, wireframe, user-stories og holde oppstartsmøte. Alt app-utviklings arbeidet ble fordelt på de fire sprintene i gantt diagrammet. Dette ble stort sett holdt, med noe overlapp inn i andre perioder. Hele gantt diagrammet med revisjoner ligger i prosjekthåndboka.

4.3.3 Utviklingsmetodikk

Som nevnt i *4.3.2* ble utviklingsarbeidet delt opp i 4 sprinter der én *user story* fra hver sprint ble tildelt hver person i gruppen. Oppdelingen av user stories i sprinter ble gjort med tanke på hvilken funksjonalitet som måtte implementeres før noe annet kunne gjøres, i tillegg til hvilken funksjonalitet som var logisk å implementere sammen.

Det ble også gjennomført noen *sprint reviews*, men ikke på noe fast tidspunkt. Her fikk vi sjansen til å få tilbakemelding og innspill fra oppdragsgiver til hva som var bra og hva som kunne gjøres bedre.

5 Diskusjon

I dette kapittelet vil resultatene fra kapittel 4 bli forklart. Kapittelet er delt opp på samme måte som kapittel 4, der hvert delkapittel tar utgangspunkt det samsvarende delkapittelet. I tillegg diskuteres systemet som helhet, i tillegg til gruppearbeidet.

5.1 Vitenskapelige resultater, diskusjon

> *Utvikling av en mobilapplikasjon for ansatte i veisektoren som effektiviserer og forenkler arbeidet med å hente nødvendig informasjon om skilt fra den nasjonale vegdatabanken.*

Produktet oppfyller kravene i problemstillingen med å effektivisere og forenkle arbeidet. Ved å kun bruke mobilapplikasjonen kan brukeren selv velge hva slags type skilt, hva slags informasjon og hvilket skilt i nærheten de vil ha informasjon om. Etter et enkelt oppsett med å velge hva slags type skilt brukeren vil se, så trenger brukeren kun å ta et bilde for å få et kart over skiltene av typen i nærheten av seg. Det er effektivt siden det er færre steg som må gjøres for å finne frem til informasjonen, og det er enkelt ettersom det ikke er teknisk krevende å bruke.

5.1.1 Server

I begynnelsen av prosjektet ble det valgt å finne frem til skiltene på den nærmeste veistrekningen til posisjonen til brukeren. Dette viste seg å være problematisk ettersom nærmeste vei ikke alltid var riktig vei. Feilene som ble funnet før endringer ble gjort er:

- gps-lokasjon som henviste brukeren til feil vei
- ved overlapp av veier kunne ikke brukeren velge hvilken vei de stod på. Disse overlappene er ved veikryss og over- og underganger.

Valget av nærmeste vei ble endret til å bruke *bounding box*, en metode som avgrenser et område som svarer med alle skiltene innenfor dette området. Alle skiltene i dette området kom i svaret fra NVDB, og ble filtrert ut avhengig av hva slags type skilt brukeren ville ha. Dette ble endret slik at kun skiltene brukeren har valgt vil komme i svaret fra NVDB.

Årsaker til at bounding box ikke ble tatt i bruk med en gang var forskjellen i de geodetiske referanserammene. Skiltene i NVDB er skrevet i UTM-33 verdier (northings og eastings), mens gps-lokasjonen til mobilen er i WGS-84 (latitude og longitude). Løsningen ble dermed å konvertere mobilposisjonen til en bounding box i utm-33, slik at skiltene hadde samme referanseramme som kartet.

Etterhvert som bruken av NVDB API-et ble bedre, ble det også gjort mindre i form av filtrering på server-siden. Serveren tar seg av konverteringen og oppsettet av forespørselen, men det eneste den gjør i filtreringen per nå er å hoppe et hakk lenger inn i JSON-objektet.

Dette er oppgaver som klienten kunne ha gjort, men ved å la serveren gjøre det, letter det på byrden til klienten. Å ha en server gjør det også enklere å utvide applikasjonen i fremtiden og gjøre det enklere å vedlikeholde

Underveis i endringen til bounding box endret NVDB sitt API, og NVDB var nede flere ganger. Det kom også flere udokumenterte endringer som kun ble nevnt i et forum. Dette er siden prosjektet er skrevet opp mot NVDB API 3 som er under utvikling.

5.1.2 Klient

Klienten er utviklet med frontendrammeverket React Native. Det gjorde utviklingsarbeidet enklere enn om vi skulle laget den i ren Java (for Android) ettersom prosjektgruppen har en del erfaring med React fra tidligere prosjekter, som er svært likt som React Native.

En beslutning som måtte tas under utviklingen var om filtreringen av informasjon skulle foregå på serversiden eller klientsiden. I utgangspunktet høres nok dette ut som en oppgave for serveren, men det viste seg å ikke være en veldig tung oppgave å utføre for klienten. I tillegg kompleksiteten av systemet og kommunikasjonen mellom server og klient forenklet av at filtreringen skjer på klientsiden.

5.1.3 Design

Designet av applikasjonen har endret seg mye underveis på noen av sidene, mens andre har holdt seg relativt like. Se kapittel 4.1.4 for figurer som viser hvordan designet av hovedsidene har endret seg fra første wireframe til sluttprodukt. I dette kapittelet tar vi for oss sidene som er illustrert her og forklarer hvorfor designet ble som det ble.

Kameraskjermen som vises i Figur 6 er den siden som ble endret minst fra wireframe til sluttdesign. Prosjektgruppen var enige om at det eneste denne skjermen skulle inneholde var et fullskjerm bilde av kameraet og en knapp for å ta bilde. Dette ble gjort relativt tidlig i utviklingsprosessen, og har ikke endret seg siden.

Siden for visning av data (Figur 7) etter at man har valgt skilt har endret seg en del fra første wireframe til sluttprodukt. Opprinnelig skulle denne siden inneholde egne knapper for å gå tilbake til kameraskjermen, velge et annet skilt og ta skjermbilde. Dette fant vi etter hvert ut var unødvendig ettersom alt dette kan gjøres med knapper i appen og/eller mobilen fra før av. Ønsker man å gå helt tilbake til kameraskjermen kan man trykke på Kameranederst på navigasjonsbaren eller trykke to ganger på tilbakeknappen. Om man vi velge nytt skilt fra kartet kan man trykke på tilbakeknappen én gang, og om man vil ta skjermbilde så har alle mobiler innebygd funksjon for dette. Ganske sent i prosjektet ble det besluttet å heller vise et statisk kartutsnitt som viser det valgte skiltet som markør på kartet, som man kan se i figur 7c.

Hjelp siden (Figur 8) har ikke endret seg mye underveis. Det er en enkel side som kun skal inneholde tekst.

Siden med innstillinger (Figur 9) er relativt lik som første wireframe, men alle feltene utenom filterer byttet ut eller erstattet. Sluttresultatet avviker litt fra andre wireframe. Det er fordi vi mente det var nyttig at listen med alle filtrene kunne åpnes og lukkes i tilfelle det skulle være svært mange forskjellige filtre.

Siden hvor man tilpasser egne filtre (figur 10) var vanskelig å lage første wireframe til ettersom vi var usikre på hva slags felter som skulle være mulige å velge blant. Det ble besluttet å gruppere en del av kategoriene under 'Metadata', 'Geometri', 'Lokasjon' og 'Relasjoner', i tillegg er også 'Egenskaper' egentlig en egen gruppe med forskjellige felter, men disse feltene er det svært mange av, så disse ble derfor gjort slik at man velger individuelle egenskaper. Mange av feltene som kan velges er nok vanskelig å tolke betydningen av for de fleste, men det er brukt de samme navnene som brukes av Vegvesenet, de vil derfor gi mening for ansatte i vegsektoren, som er de appen primært er laget for.

Generelt sett ble det bestemt å ha et ganske enkelt og minimaslistisk design på appen uten for mye bruk av farger. Grunnen til dette er at det fort kan se rotete og lite pent ut om det tas i bruk mye farger. De eneste fargene som ble brukt er en mørkere rødfarge som primærfarge, og en lysere, mer gjennomsiktig rødfarge som sekundærfarge.

5.2 Ingeniørfaglige resultater, diskusjon

5.2.1 Prosjektmål

Ved leveringstidspunktet oppfyller produktet nesten alle funksjonelle- og ikke funksjonelle krav i *visjonsdokumentet* i henhold til den reviderte oppgaveteksten. Hadde ikke oppgaveteksten blitt endret så hadde appliksjonen hatt en stor mangel i at bildegjenkjenningsmodulen ikke er implementert, vi så oss nødt til å endre oppgaveteksten ettersom det ikke var noen måte å få tilgang til modulen fra Triona AS.

Et annet mål som ble endret er *Kart over skilt som kan velges om det er flere skilt av samme type i nærheten*. Opprinnelig var dette målet en liste i stedet for et kart. På grunn av arbeidsprosessen som ble brukt, Scrum, fikk vi mulighet til å få tilbakemelding fra oppdragsgiver/produkteier etter hver sprint gjennom *sprint review*. Prosjektgruppen var enige med Triona AS om at listen var uoversiktlig og ikke særlig brukervennlig, og vi besluttet derfor å erstatte listen med et kart. Dette viste seg å være en mye bedre løsning for å finne frem til riktig skilt.

Det eneste funksjonelle kravet som ikke er oppfyllt er *Tilpasse utseende av applikasjonen*. Grunnen til dette er at prosjektgruppen revurderte kravet etter planleggingsfasen og mente det at det unødvendig bruk av tid og plass, da det er funksjonalitet som mest sannsynlig ikke ville blitt brukt.

Et ikke funksjonellt krav som kun er delvis oppfyllt er *Applikasjonen skal fungere til både Android og iOS*. Begge medlemmene av prosjektgruppen har kun tilgang på telefon som kjører Android, og PC med Windows eller Linux. Det gjorde det svært vanskelig å få testet applikasjonen på iOS ettersom iOS applikasjoner kun kan kjøres på telefoner laget av Apple eller gjennom emulator på Mac. Det er ingen grunn til at applikasjonen ikke skal være kjørbar på iOS, da React-native kode kan kompileres direkte til Swift kode, som brukes på iOS, men det er vanskelig å si at kravet er fullstendig oppfyllt uten å ha testet det.

5.2.2 Brukertester

Det er flere ting som er verdt å merke seg når man tolker resultatene fra brukertestene:

- Testene ble ikke utført av personer som jobber eller har jobbet i veisektoren.
- På alle testene ble *vegkartet* testet før *SkiltInfo*.
- Det ble kun gjort tester på 4 personer.
- Ingen av testbrukerne hadde brukt noen av systemene før, de fikk heller ingen innføring i hvordan å bruke dem.
- Vegkartet kan brukes til å finne informasjon om flere veiobjekter enn kun skilt.
- Testene ble gjort med kun testadministrator og testbruker tilstede. Brukerne ble selvsagt heller ikke informert om resultatene til de andre brukerne.

Fra resultatene presentert i Tabell 1 ser vi at det er store forskjeller på både tiden som ble brukt, og hvordan brukerne rangerte hvor enkelt det var å bruke applikasjonen. Gjennomsnittstiden på vegkartet lå på 2 minutter og 15 sekunder, og gjennomsnittstiden på *SkiltInfo* lå på 38.5 sekunder. I tillegg var rangeringen på hvor enkelt hvert system var å bruke høyere på *SkiltInfo*, med gjennomsnitt på 9/10, over 4.75/10 på vegkartet. Til tross for punktene nevnt i listen ovenfor er det betydelig forskjell på begge variablene i brukertestene.

5.2.3 Kodetester

Figur 11 viser testdekningen i frontend. Den kunne nok vært høyere, men noen av komponentene lot seg ikke teste på grunn av en bug med mocking av state i React komponenter. I tillegg prioriterte vi testing på backend over frontend fordi vi mente det var viktigere siden det ofte er vanskeligere å oppdage feil på backend; en feil i frontend vil ofte gi utslag i brukergrensesnittet, og da oppdages feilen fort.

5.3 Administrative resultater, diskusjon

Arbeidsprosessen har vært relativt jevn, med en liten innspurt mot slutten av prosjektet, dette kan man se i Figur 13. Fremdriftsplanen ble stort sett holdt, med naturlige avvik på grunn av uforutsette problemer. Fordelen med dette er at utviklingsmetoden som er brukt, Scrum,

legger til rette for at man skal kunne gå tilbake i fremdriftsplanen for å gjøre endringer og forbedre.

5.4 Helhetlig systemperspektiv

Produktet som har blitt utviklet er en mobilapplikasjon som er enkel å bruke, og som krever lite teknisk kompetanse, som reduserer krav for opplæring. Eneste krav til å kunne bruke applikasjonen er at man har den installert på en mobil enhet med kamera som er tilkoblet internett. I tillegg må GPS være aktivert. Det gjør at den er tilgjengelig for så og si alle som skulle ha bruk for den.

Produktet gir økonomisk gevinst gjennom at det går raskere å finne frem skilt. Det gjør at brukere kan håndtere flere skilt på kortere tid.

5.5 Refleksjon over gruppearbeidet

Gruppearbeidet og fordelingen av arbeidsoppgaver har fungert svært bra. Gruppen har jobbet sammen på flere prosjekter og mindre oppgaver tidligere og kjenner hverandre godt. I tillegg kjenner gruppemedlemmene hverandres styrker og svakheter, og har kunnet fordele arbeidsoppgaver på grunnlag av dette.

På grunn av at vi kjenner hverandre godt har kommunikasjonen også vært god, og det har vært lav terskel for å si fra dersom noen mener at noe burde endres på eller kunne vært gjort bedre. Kommunikasjonen ble en litt større utfordring de tre siste månedene av prosjektet etter utbruddet av Covid-19, som førte til at begge kun jobbet hjemmefra og ikke på kontor. I denne tiden ble foregikk kommunikasjonen hovedsaklig gjennom meldinger, men også gjennom avtalte virtuelle møter.

6 Konklusjon og videre arbeid

6.1 Problemstilling

> *Utvikling av en mobilapplikasjon for ansatte i veisektoren som effektiviserer og forenkler arbeidet med å hente nødvendig informasjon om skilt fra den nasjonale vegdatabanken.*

Applikasjonen ble utviklet med Scrum arbeidsmetodikk, og arbeidet gikk for det meste etter tidsplanen. Uforutsette utfordringer og endringer i kravspesifikasjonen etter sprint review ble tatt høyde for da fremdriftsplanen ble satt opp, og de ble derfor håndtert på en god måte. Resultatet er et ferdig produkt som har nesten alt av planlagt funksjonalitet.

En utfordring ved testing av produktet var at vi ikke fikk gjort brukertester av produktet med den relevante brukergruppen. Vi måtte derfor gjøre det nest beste, som var brukertester på medstudenter og personer i samme husholdning. Resultatene fra brukertesetene er presentert i Tabell 1. Selv om testene ikke var optimale, viser resultatene store forskjeller på både tidsbruk og hvor enkelt brukeren synes det var å finne frem til skiltet. Testbrukerne brukte i gjennomsnitt nesten 4 ganger lengre tid på å finne frem til oppsettingsdatoen til det oppgitte skiltet med vegkartet sammenlignet med *SkiltInfo*, i tillegg synes de, i gjennomsnitt, at det var nesten dobbelt så enkelt å finne frem informasjon om skiltet i *SkiltInfo*.

Selv om resultatene kanskje hadde vært annerledes om det var personer som jobbet i vegsektoren som utførte testene, så tilsier resultatene fra de gjennomførte testene at det er både mer effektivt og enklere å finne frem skiltet med applikasjonen sammenlignet med løsningen med vegkartet.

6.2 Videre arbeid

Oppgaven var å utvikle en applikasjon som skulle implementere en eksisterende bildegjenkjenningsmodul, noe som gjør implenteringen en viktig oppgave for videre arbeid.

En annen viktig oppgave er å få utført brukertester for den relevante brukergruppen.

Applikasjonen har ingen løsninger for å tilpasse applikasjonen til brukeren. Tilpasningsmulighetene kan være endring av tekststørrelse, skrifttype og bakgrunnsfarge. Siden hvor informasjon om det valgte skiltet vises kan også gjøres slik at brukeren selv bestemmer rekkefølgen på informasjonen som vises.

Ikke alle egenskaper for skilt har like mye informasjon, så de forskjellige egenskapene kan tilpasses og inkludere mer informasjon.

Å utvide serveren til å utføre flere oppgaver for å lette byrden på klienten. En oppgave serveren kunne ha tatt istedenfor klienten er å konvertere posisjonene til skiltene som skal legges inn i kartet istedenfor at det konverteres i klienten.

Å utvide applikasjonen til å gjelde andre faste vegobjekter. Dette krever en opptrening av modulen til å gjenkjenne andre vegobjekter og tilpasning av server og klient til å vise frem denne informasjonen. Informasjonen som står om de forskjellige objektene har ikke samme egenskapsnavn, så filtersystemet må endres slik at brukeren får riktig informasjon.

Mulige utvidelser i fremtiden etter at bildegjenkjenningsmodulen er implementert er å lese metadata fra lagrede bilder istedenfor å ta bilde selv. Dette vil øke bruksområdet til applikasjonen ettersom brukeren ikke trenger å være i nærheten av skiltet for å få informasjonen.

Referanser

Beck, Kent (2003). *Test-driven development: by example*. Addison-Wesley Professional.

CodeAcademy (2019). *What is REST?* https://www.codecademy.com/articles/what-is-rest. Hentet: 09.03.2020.

Cohen, David, Mikael Lindvall og Patricia Costa (2004). «An introduction to agile methods.» I: *Advances in computers* 62.03, s. 1–66.

Difi (2019). *Oppbygging av WCAG 2.0*. https://uu.difi.no/krav-og-regelverk/wcag-20-standarden/oppbygging-av-wcag-20. Hentet: 09.03.2020.

– (2020). *Effektivisering*. https://www.difi.no/fagomrader-og-tjenester/effektivisering. Hentet: 06.03.2020.

Dix, Alan mfl. (2003). *Human-computer interaction*. Pearson Education.

Fielding, Roy mfl. (1999). «Hypertext transfer protocol–HTTP/1.1». I:

Fowler, Martin (2006). «Continuous Integration». I: s. 1–14. DOI: https://moodle2019-20.ua.es/moodle/pluginfile.php/2228/mod_resource/content/2/martin-fowler-continuous-integration.pdf.

Janzen, David og Hossein Saiedian (2005). «Test-driven development concepts, taxonomy, and future direction». I: *Computer* 38.9, s. 43–50.

Kartverket (2020). *GPS og GNSS*. https://www.kartverket.no/Posisjonstjenester/GPS-og-GNSS. Hentet: 04.03.2020.

Lid, Inger Marie (2020). *Universell Utforming*. https://snl.no/universell_utforming. Hentet: 09.03.2020.

Myers, Glenford J, Corey Sandler og Tom Badgett (2011). *The art of software testing*. John Wiley & Sons.

Pwc (2020). *Robotic process automation (RPA)*. https://www.pwc.no/no/teknologi-omstilling/digitalisering-pa-1-2-3/rpa.html. Hentet: 23.03.2020.

Rouse, Margaret (2020). *What is RPA? Everything you need to know*. https://searchcio.techtarget.com/definition/RPA. Hentet: 28.04.2020.

soumya08 (2020). *Version Control Systems*. https://www.geeksforgeeks.org/version-control-systems. Hentet: 27.02.2020.

Taylor, Charles L (2017). *Convert geographical coordinates*. http://home.hiwaay.net/taylorc/toolbox/geography/geoutm.html. Hentet: 09.04.2020.

UIO (2010). *Forskningsmetoder i menneske-maskin interaksjon*. https://www.uio.no/studier/emner/matnat/ifi/INF2260/h11/undervisningsmateriale/Week-1.pdf. Hentet: 10.03.2020.

Prosjektnr 22

SkiltInfo

Utvikling av applikasjon innen veginformatikk

Vedlegg A - Visjonsdokument

Versjon 1.2

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
22.01.2020	<0.1-D>	Førsteutkast	Quan Tran, Patrick Thorkildsen
24.01.2020	<0.2-D>	Endret “funksjonelle egenskaper” og “sammendrag av brukernes behov”	Patrick Thorkildsen
27.01.2020	<0.3-D>	Kap 4-6, endre far veiobjekter til veiskilt	Quan Tran
30.01.2020	<0.4-D>	Problemstilling	Quan Tran
16.04.2020	<1.0>	Omformulert og lagt til flere krav under “Funksjonelle krav”	Patrick Thorkildsen
29.04.2020	<1.1>	Omformuleringer som følge av endring av oppgavetekst	Patrick Thorkildsen
06.05.2020	<1.2>	Siste endringer før det settes inn som vedlegg til hovedrapporten	Quan Tran, Patrick Thorkildsen

Innhold

Innledning 1
1. Sammendrag problem og produkt 1
   1.1. Problemsammendrag 1
   1.2. Produktsammendrag 1
2. Overordnet beskrivelse av interessenter og brukere 2
   2.1. Oppsummering interessenter 2
   2.2. Oppsummmering brukere 2
   2.3. Brukermiljøet 2
   2.4. Sammendrag av brukernes behov 3
   2.5. Alternativer til vårt produkt 3
3. Produktoversikt 4
   3.1. Produktets rolle i brukermiljøet 4
   3.2. Forutsetninger og avhengigheter 4
4. Produktets funksjonelle egenskaper 4
5. Ikke-funksjonelle egenskaper og andre krav 4

Innledning

Dette dokumentet beskriver krav til bacheloroppgaven "Lage applikasjon innen veginformatikk" som skal utføres av Quan Tran og Patrick Thorkildsen i samarbeid med Triona AS ("kunde"). Oppgaven går ut på å lage en mobilapplikasjon som bruker bildegjenkjenning og GPS-lokasjon for å vise frem relevant informasjon om veiskilt.

1. Sammendrag problem og produkt

1.1. Problemsammendrag

Å lage en applikasjon som det kan implementeres en eksisterende bildegjenkjenningsmodul i, for å kunne få vist relevant informasjon om spesifikt veiskilt. Bildegjenkjenningsmodulen må kunne implementeres i produktet på en måte som gjør at det viser riktig skilt. Relevant informasjon skal så vises frem på en ryddig måte.

Hvordan skal vi velge det riktige skiltet når det er flere skilt av samme type i området. GPS-lokasjon kan være unøyaktig og vise feil sted. For eksempel ved kryss og over/underganger.

Problem med	å finne relevant informasjon om bestemte veiskilt.
Berører	mennesker som arbeider med veiskilt.
som resultat av dette	tar det lang tid å finne frem til det bestemte veiskiltet og plukke ut relevant informasjon.
en vellykket løsning vil	Gjøre det enkelt og effektivt å finne relevant informasjon om bestemte veiskilt.

1.2. Produktsammendrag

For	Ansatte i veisektoren
som	har behov for en enklere måte å hente informasjon fra NVDB om bestemte veiskilt
"produktnavn"	er en mobilapplikasjon til Android og iOS
som	gjør det enkelt å hente ut denne informasjonen
I motsetning til	å manuelt lete i databasen
har vårt produkt	bildegjenkjenning og GPS-lokasjon, som forenkler denne jobben.

2. Overordnet beskrivelse av interessenter og brukere

2.1. Oppsummering interessenter

Navn	Utdypende beskrivelse	Rolle under utviklingen
Triona AS	Prosjekteier og oppdragsgiver	Veilede utviklingen med tanke på krav til sluttprodukt. Støtte til nødvendig maskinvare, evt. programvare.
Bruker	Sluttbrukere av systemet	Teste systemet. Ta bilde av veiskilt og få ut informasjon fra NVDB
Prosjektgruppe	Utviklere av systemet	Utvikle, teste, administrere og vedlikeholde systemet. Kalle inn til møter
Veileder	Veileder fra NTNU	Veilede prosjektfremdriften og prosessen. Stille opp på møter

2.2. Oppsummmering brukere

Navn	Utdypende beskrivelse	Rolle under utviklingen	Representert av
Bruker	Alle som laster ned og bruker applikasjonen	Testing	Triona AS og prosjektgruppen

2.3. Brukermiljøet

Løsningen brukes gjennom en applikasjon for Android og iOS, den må derfor tilpasses begge plattformer. Applikasjonen krever tilgang til GPS, internett og kamera.

2.4. Sammendrag av brukernes behov

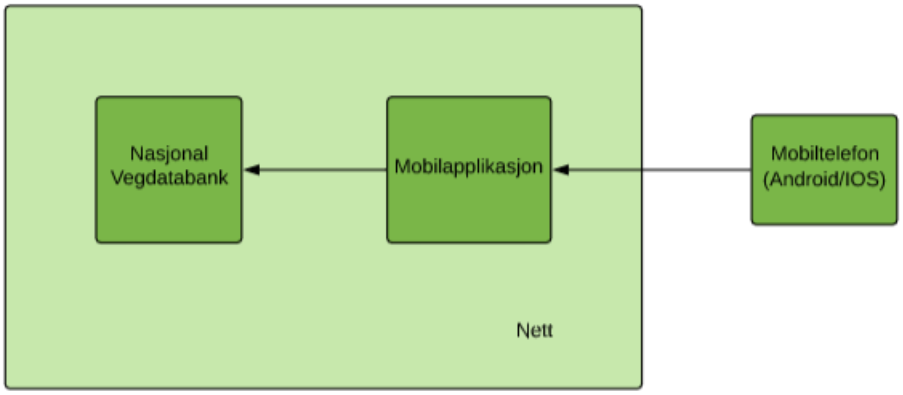
Behov	Prioritet	Påvirker	Dagens løsning	Foreslått løsning
Ta bilde av et veiskilt.	Høy	Henting av informasjon	Ingen	Integrere mobilkamera i applikasjonen.
Hente ut informasjon	Høy	Henting av informasjon	Manuell	Bruke GPS-lokasjon og simulering av bildegjenkjenningsmodul.
Filtrere informasjon	Høy	Henting av informasjon	Manuelt via vegvesenet sin nettside	Bruker bestemmer hva slags informasjon som skal vises før den hentes.
Se informasjon	Høy	Visning	Manuelt	Presentere data på en ryddig måte i applikasjonen.
Tilpasse visning	Lav	Visning	Ingen	Mulighet for tilpasning av visning; størrelse og farge
Få hjelp og informasjon om applikasjonen	Lav	Brukervennlighet	Ingen	Hjelp-side i navigasjonsbaren

2.5. Alternativer til vårt produkt

Vegvesenet sitt vegkart gir mulighet til å manuelt finne frem ved å lete i kartet

3. Produktoversikt

3.1. Produktets rolle i brukermiljøet



Figur 1. Brukermiljø

3.2. Forutsetninger og avhengigheter

Applikasjonen utvikles med forutsetningen om at det skal implementeres en tidligere utviklet bildegjenkjenningsmodul. Modulen skal identifisere skiltet ved hjelp av gps-lokasjon og bildegjenkjenning. Hvis bildegjenkjenningsmodulen har svakheter vil dette videreføres til applikasjonen.

Applikasjonen er avhengig av at NVDB ikke er nede for å fungere.

4. Produktets funksjonelle egenskaper

Funksjonelle egenskaper

- Filtrering av informasjonen som skal vises
- Oppretting av egendefinerte filtre, samt forhåndsdefinerte filtre
- Side for å få hjelp om bruk av applikasjonen og generell info om applikasjonen
- Lokal lagring av innstillinger
- Mulighet for å integrere bildegjenkjenningsmodul i applikasjonen som bruker bildet som er tatt
- Kart over skilt som kan velges om det er flere skilt av samme type i nærheten
- Tilpasse utseende av applikasjonen

5. Ikke-funksjonelle egenskaper og andre krav

- Enhetstesting og CI skal være implementert
- Applikasjonen skal fungere til både Android og iOS.
- Løsningen skal være brukervennlig, informasjonen skal fremstilles på en ryddig måte
- Kode skal lagres på GitHub

Prosjektnr 22

SkiltInfo

Utvikling av applikasjon innen veginformatikk

Vedlegg B - Kravdokumentasjon

Versjon 1.2

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
<27.01.2020>	<0.1-D>	Førsteutkast	Patrick Thorkildsen Quan Tran
<03.03.2020>	<0.2-D>	Oppdatering av userstories og laget sekvensdiagram	Patrick Thorkildsen
<16.04.2020>	1.0	Fjernet duplikat-userstories og la til en ny om bruk av kart	Patrick Thorkildsen
<05.05.2020>	1.1	Oppdatert sekvensdiagram	Patrick Thorkildsen
<06.06.2020>	1.2	Klargjøring til å legge inn i hovedrapport	Patrick Thorkildsen, Quan Tran

Innhold

Innledning 1
1. User Stories 1
2. Domenemodell 3
2.1 Sekvensdiagrammer 4
3. Prototyper 4
3.1. Wireframes 4

Innledning

Dette dokumentet er skrevet i forbindelse med bacheloroppgaven for dataingeniører på NTNU. Hensikten med dokumentet er å sette kravene som trengs for systemet. Dokumentet inneholder User Stories, domenemodell og wireframe. Alle diagrammer og figurer i dette dokumentet er laget med Lucidchart (www.lucidchart.com)

1. User Stories

Som bruker Ønsker jeg å tilpasse utseende av applikasjonen Slik at jeg kan få den til å se ut som jeg vil
Akseptansekriterie: Kan endre fargetema (mørkt/lyst), endring av skriftstørrelse hvis den ikke følger skriftstørrelsen som er satt i OSet. Dette skal gjøres gjennom en meny med innstillinger

Som bruker Ønsker jeg å ta et bilde av et veiskilt Slik at det kan brukes til å finne riktig skilt
Akseptansekriterie: Få oversikt av veiskilt basert på gps-lokasjonen, bildet som tas kan brukes i videreutvikling av applikasjonen når bildegjenkjenningsmodulen skal implementeres

Som bruker Ønsker jeg å ha tilgang til ulike forhåndsinnstillinger og kunne legge til mine egne Slik at jeg enklere kan velge hva slags informasjon som vises
Akseptansekriterie: Brukeren får mulighet til å velge “Enkel” (vis minimalt med informasjon om objektet), “Fullstendig” (vis all lagret informasjon om objektet) eller egendefinert (egendefinerte forhåndsinnstillinger for hva som skal hentes ut)

Som bruker Ønsker jeg å ha tilgang til en “hjelp” side som forklarer hva applikasjonen gjør, hvordan man bruker den og hvor informasjonen kommer fra Slik at jeg skal kunne bruke applikasjonen uten å måtte spørre om andre om hjelp
Akseptansekriterie: Når man trykker på symbolet for hjelp-siden i navigasjonsbaren blir man vist en slags faq side hvor man finner all nødvendig informasjon

Som bruker Ønsker jeg en side hvor jeg kan se innstillingene mine og endre på dem Slik at jeg lett kan få oversikt over og endre innstillingene
Akseptansekriterie: Brukeren har tilgang til en innstilling-side fra naviagsjons-baren. På denne siden få man oversikt over hvilke innstillinger som er satt samt mulighet til å endre innstillingene.

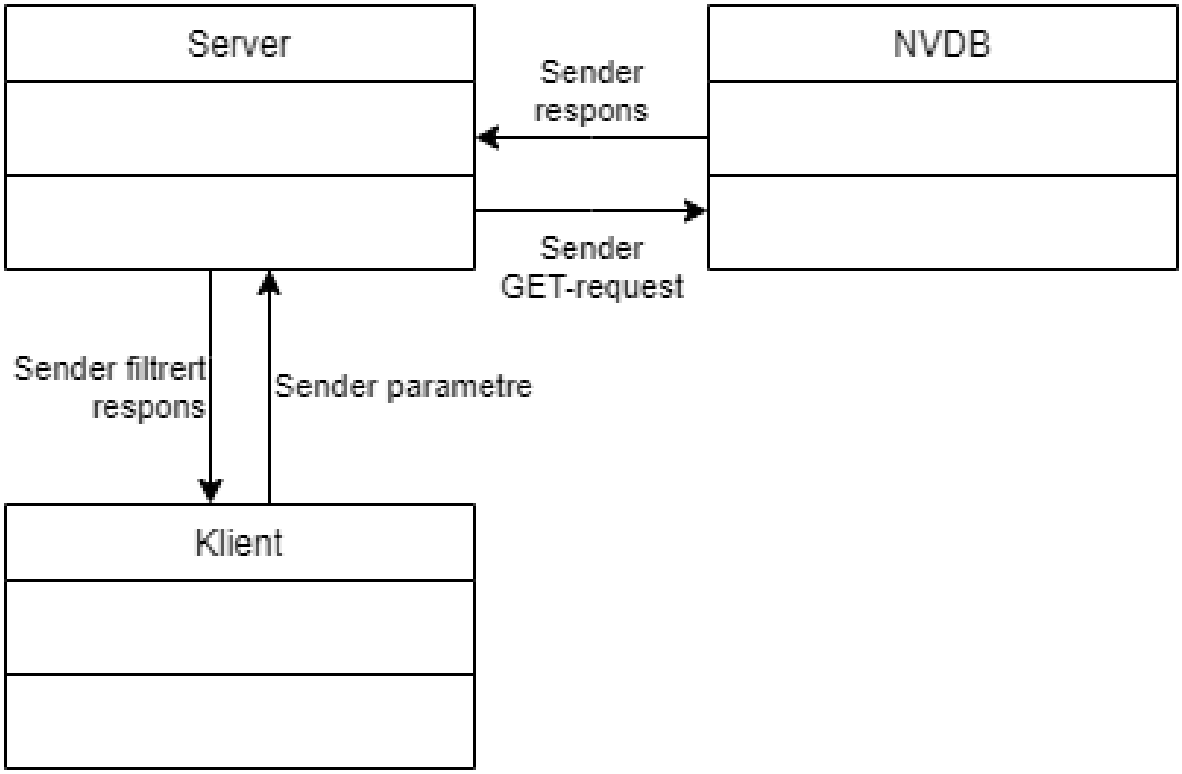
Som bruker Ønsker jeg at innstillingene mine skal lagres lokalt Slik at jeg slipper å endre på dem hver gang
Akseptansekriterie: Innstillingene lagres lokalt på enheten, og kan hentes tilbake selv om appen startes på nytt.

Som bruker ønsker jeg å velge selv hva jeg vil se av data Slik at jeg ikke får unødvendig informasjon
Akseptansekriterie: Data som vises er kun det som er valgt i det aktive filteret

Som bruker ønsker jeg at bildet jeg tok skal forhåndsvises når jeg leser informasjonen som hører til Slik at jeg vet hvordan bildet som ble brukt ser ut
Akseptansekriterie: Bildet vises på “DisplayInformationScreen.js”

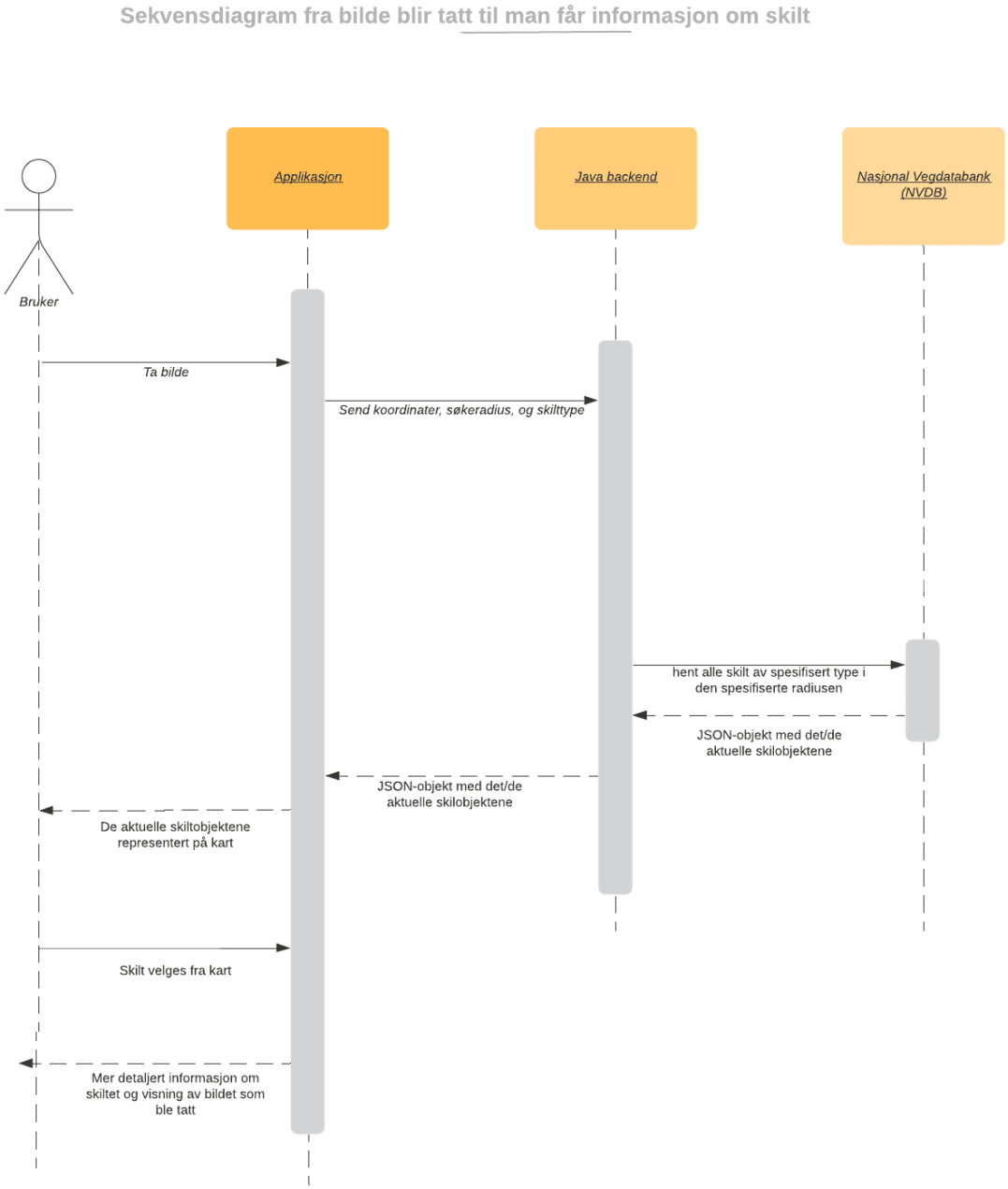
Som bruker ønsker jeg å få opp et kart med alle mulige skilt når jeg har tatt bilde av et skilt, fra dette kartet kan man velge det skiltet man er ute etter Slik at det blir enklere å velge riktig skilt
Akseptansekriterie: Etter man tar bilde vil man få opp et kart med markeringer av alle skilt som matcher riktig type og er innen en viss avstand fra der bildet ble tatt. Et av disse skiltene kan velges, og man vil da få en mer detaljert visning av dette skiltet.

2. Domenemodell



Figur 1. Domenemodell av applikasjonen

2.1Sekvensdiagrammer



Figur 2. Sekvensdiagram fra når et bilde blir tatt til man får informasjon om skilt

3. Prototyper

3.1. Wireframes

Prototype ble laget med Balsamiq og kan åpnes med følgende link:

https://drive.google.com/file/d/1-HggqZ-ltzT9AtuBHFHOxDhL5kib2s_4/view?usp=sharing

Prosjektnr 22

SkiltInfo

Utvikling av applikasjon innen veginformatikk

Vedlegg C - Systemdokumentasjon

Versjon 1.1

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
13/02/2020	<0.1-D>	Førsteutkast	Quan Tran Patrick Thorkildsen
18/02/2020	<0.2-D>	Endringer etter tilbakemelding fra veileder	Patrick Thorkildsen
06/05/2020	1.0	Klargjøring til å legge inn i hovedrapport	Patrick Thorkildsen, Quan Tran
15/05/2020	1.1	Oppdaterte figurer/bilder	Quan Tran

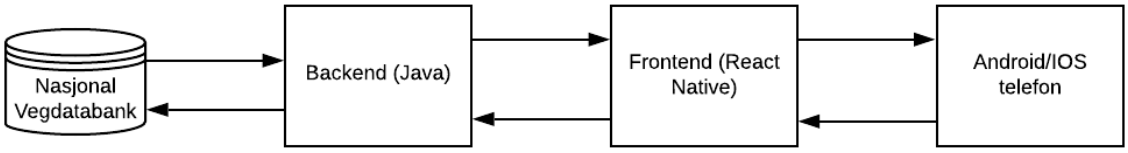
Innhold

1. Innledning 1
2. Arkitektur 1
3. Prosjektstruktur 2
4. Klassediagram 3
5. Server-tjenester 5
6. Sikkerhet 5
7. Installasjon og kjøring 6
   7.1. Backend: 6
   7.2. Installasjon og oppstart: 6
   7.3. Kjøring: 6
8. Dokumentasjon av kildekode 7
9. Kontinuerlig integrasjon og testing 7

1. Innledning

Dette dokumentet er skrevet i forbindelse bacheloroppgaven for dataingeniører ved NTNU. Dokumentet skal beskrive hvordan systemet er bygd opp. Dokumentet inneholder dermed systemarkitektur, prosjektstruktur, klassediagram, hvordan server-tjenesten er bygd opp, sikkerhet, installasjonsguide, javadoc og testing. Alle diagrammer og figurer i dette dokumentet er laget med Lucidchart (www.lucidchart.com)

2. Arkitektur

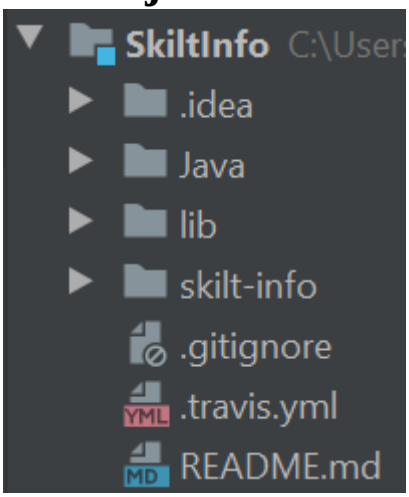


Figur 1. Systemarkitektur

All data om skilt som brukes i appen hentes fra Nasjonal Vegdatabank (NVDB). De relevante skiltene hentes av backend basert på hva slags skilt som er gjenkjent og posisjonen til telefonen som tok bildet, det vil kun hentes flere skilt dersom det er flere skilt av samme type i samme område. Disse skiltene sendes videre til frontend der brukeren får mulighet til å velge ett av dem i kart. Så vil informasjonen om skiltet filtreres basert på brukerens valg, og deretter vises frem.

Lese-APIet til NVDB er åpent og kan brukes av alle. Det lagres heller ikke sensitiv informasjon i noen andre deler av applikasjonen. På grunn av dette er ikke sikkerhet noe problem

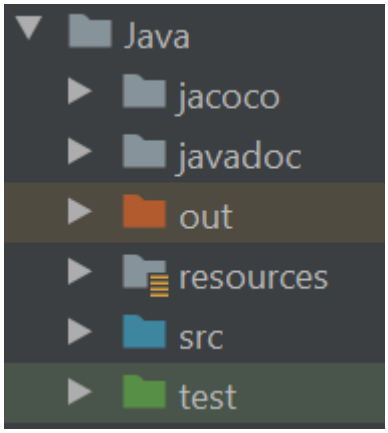
3. Prosjektstruktur



Figur 2. Overordnet struktur av prosjektet

Prosjektet er delt inn i to hoveddeler, Java som er backend, og skilt-info som er frontend skrevet i Javascript. I lib ligger de eksterne bibliotekene som må tas med for å få kjørt tester og metoder i backend.

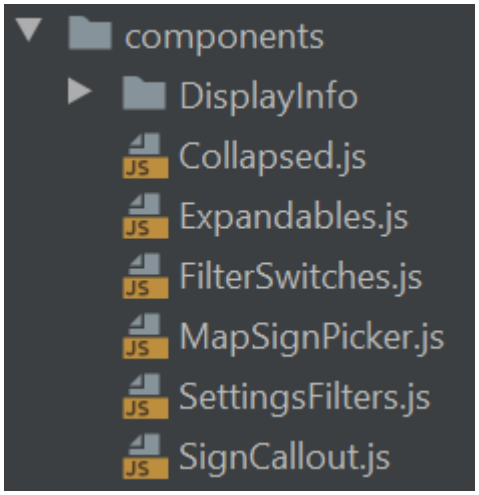
Skilt-info er frontenden av prosjektet og inneholder alle komponenter og sider som vises på mobilen.



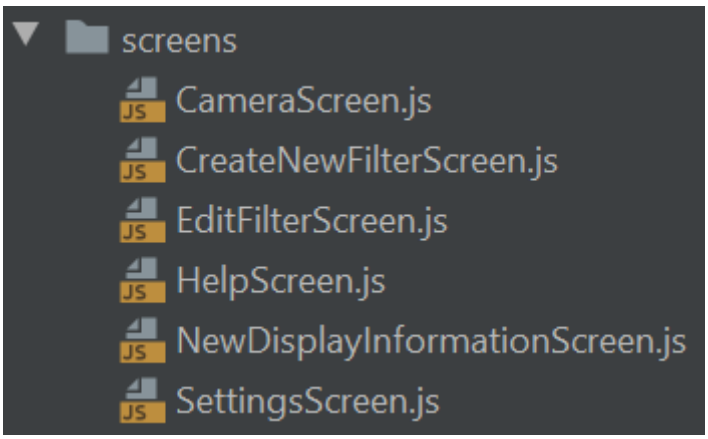
Figur 3. Strukturen i Java

Java har 6 mapper, out, resources, src og test. I out ligger de kompilerte klassene. I resources ligger properties filen som brukes. I src ligger kildekoden og i test ligger testene. Javadoc inneholder javadoc-filene og jacoco inneholder testdekningen.

I skilt-info er det flere mapper ettersom det er en expo-prosjekt, men de to mest relevante er components og screens. I components ligger de gjenbrukbare komponentene og i screens ligger «skjermene», som er sidene som vises på applikasjonen.

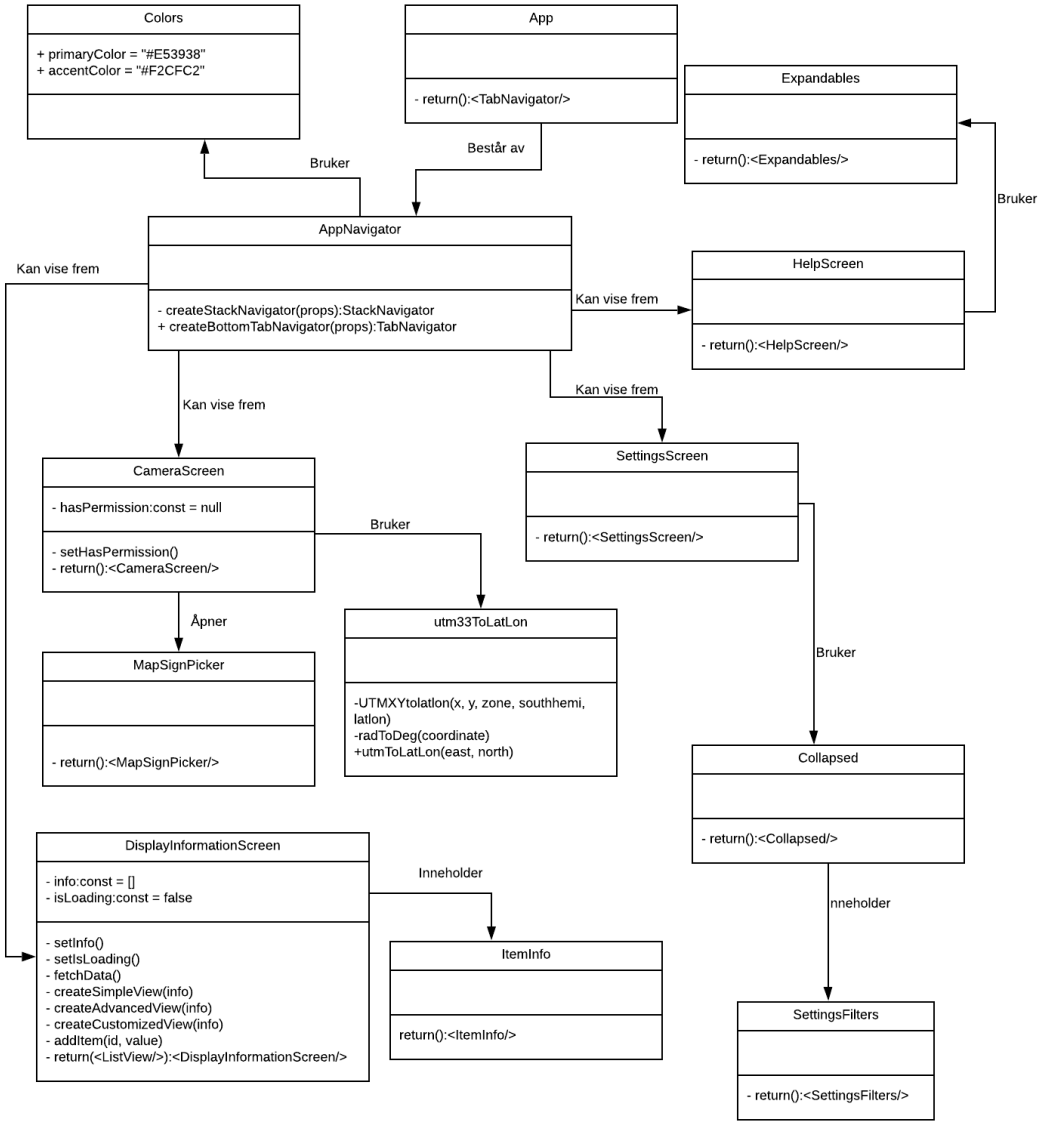


Figur 5. Components mappen med dets komponenter



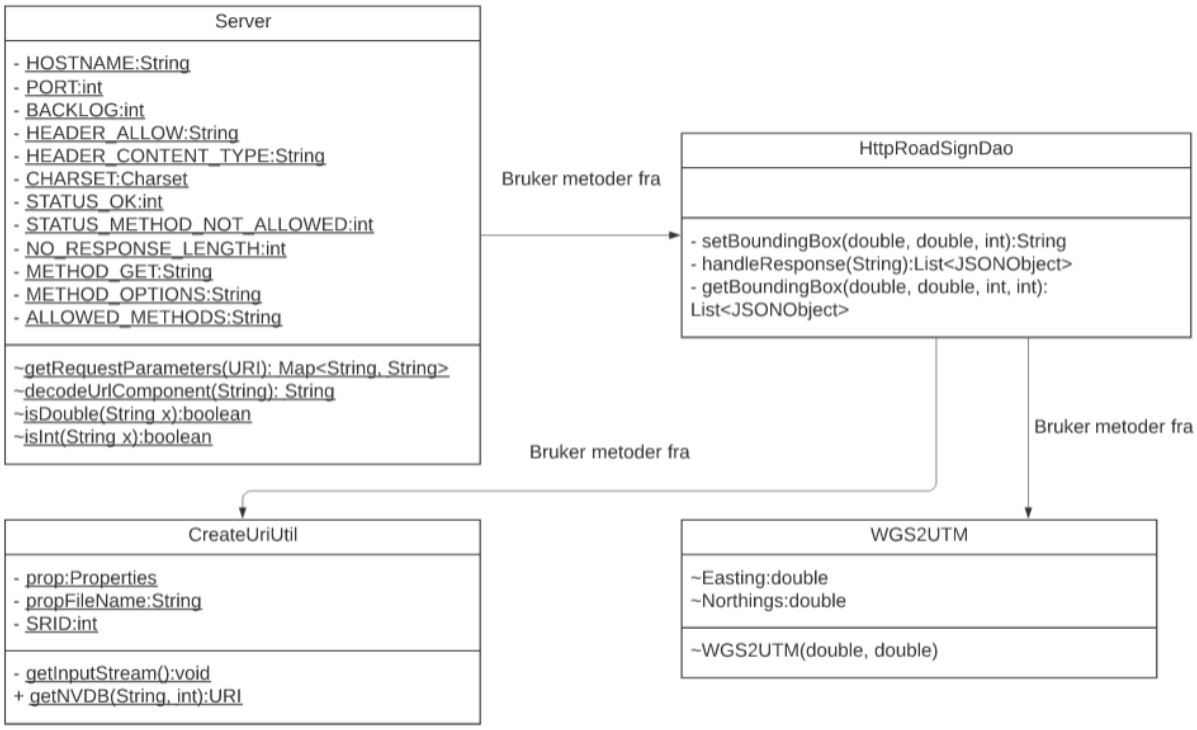
Figur 4. Screens mappen med dets skjermer

4. Klassediagram



Figur 6. Klassediagram av frontend

Klassediagram for frontend. Teknisk sett er det ikke brukt klasser, men funksjonelle komponenter i React. Rent teknisk er disse komponentene JavaScript funksjoner som tar inn "props" som argument, og returnerer et React element, men strukturen til funksjonene er på mange måter likt som klasser.



Figur 7. Klassediagram for backend.

Server tar i bruk metoder fra klassen HttpRoadSignDao for å kjøre spørringer mot NVDB og filtrere bort objekter som ikke passer inn i parameterne som er spurt. HttpRoadSignDao bruker linken som lages i CreateUriUtil og gjør om fra latitude og longitude til northings og eastings i WGS2UTM.

5. Server-tjenester

Figur 8. Bildet viser et eksempel på respons fra NVDB, denne responsen inneholder 7 skilt

REST-ressursen tar i bruk fire parametre, id, søkeradius, lat og lon. Id skal si hva slags type skilt det er, for eksempel forkjørsveg eller fartsgrense 50 km/t. Lat og lon er latitude og longitude verdiene hvor bildet er tatt. Søkeradiusen definerer området som skal søkes i. Ut fra disse parameterne vil NVDB svare med skiltene som har riktig id innenfor søkeområdet definert av søkeradiusen.

6. Sikkerhet

Ingen spesielle sikkerhetstiltak må tas. Det er ingen form for autentisering ettersom det kun skal leses offentlig data, og det tas ikke hensyn til cross-site scripting ettersom det ikke kan legges inn egne verdier. Parameterne må også være tall for at forespørselen skal kjøres.

Fra applikasjonens side logges eller lagres hverken GPS-lokasjon til brukerne eller bilder som tas i appen.

7. Installasjon og kjøring

7.1. Backend:

com.sun.net.httpserver - Brukes for å sette opp serversiden.

org.json - JSON-behandler. Brukes for å konvertere fra og til JSON og for å hente ut verdier i et JSON-objekt

org.osgeo.proj4j - Konvertering mellom WGS84 og UTM-33

7.2. Installasjon og oppstart:

1. Last ned og installer Node.js fra https://nodejs.org/en/
2. Last ned Ngrok fra https://dashboard.ngrok.com/get-started
3. Kjør *git pull https://github.com/Patrickthork/SkiltInfo.git*
4. Kjør Server.java
5. Åpne ngrok og kjør kommandoen *ngrok http 8080*
6. Kopier en av de to url-ene (http eller https)
7. Lim inn url-en i variabelen URL som finnes i *skilt-info/screens/CameraScreen.js*
8. Kjør *npm install* i */skilt-info* og vent på at pakkene lastes ned
9. Kjør *npm start* i */skilt-info*

7.3. Kjøring:

Kjøring på android emulator:

1. Last ned og installer Android Studio https://developer.android.com/studio
2. Gå til AVD manager og last ned et "Virtual Device"
3. Start emulatoren
4. Gå tilbake til konsollvinduet der applikasjonen kjører og trykke "a" for å åpne applikasjonen på emulatoren

Kjøring på Android:

1. Last ned *Expo* fra play store https://play.google.com/store/apps/details?id=host.exp.exponent&hl=no
2. Åpne Expo og skann QR-koden som ligger i konsollvinduet der applikasjonen kjører

8. Dokumentasjon av kildekode

Javadoc er en metode å generere dokumentasjon fra kildekode i Java laget av Sun. Javadoc blir skrevet for å enklere forstå koden og dermed gjøre det enklere å vedlikeholde. Den vil generere dokumentasjonen i HTML format. Javadoc påvirker ikke ytelsen til programmet ettersom all kommentering blir fjernet ved kompilering.

For å se Javadoc, åpne allclasses.html i en nettleser. Denne filen ligger i mappen Java/javadoc. En måte å få generert javadoc på er å gjøre det gjennom en IDE, for eksempel IntelliJ.

1. I verktøylinjen, klikk på Tools > Generate JavaDoc.
2. Velg å generer for hele prosjektet.

3. Velg hvor javadoc-filene skal genereres. Javadocen kan automatisk åpnes i standardnettleser om dette velges.

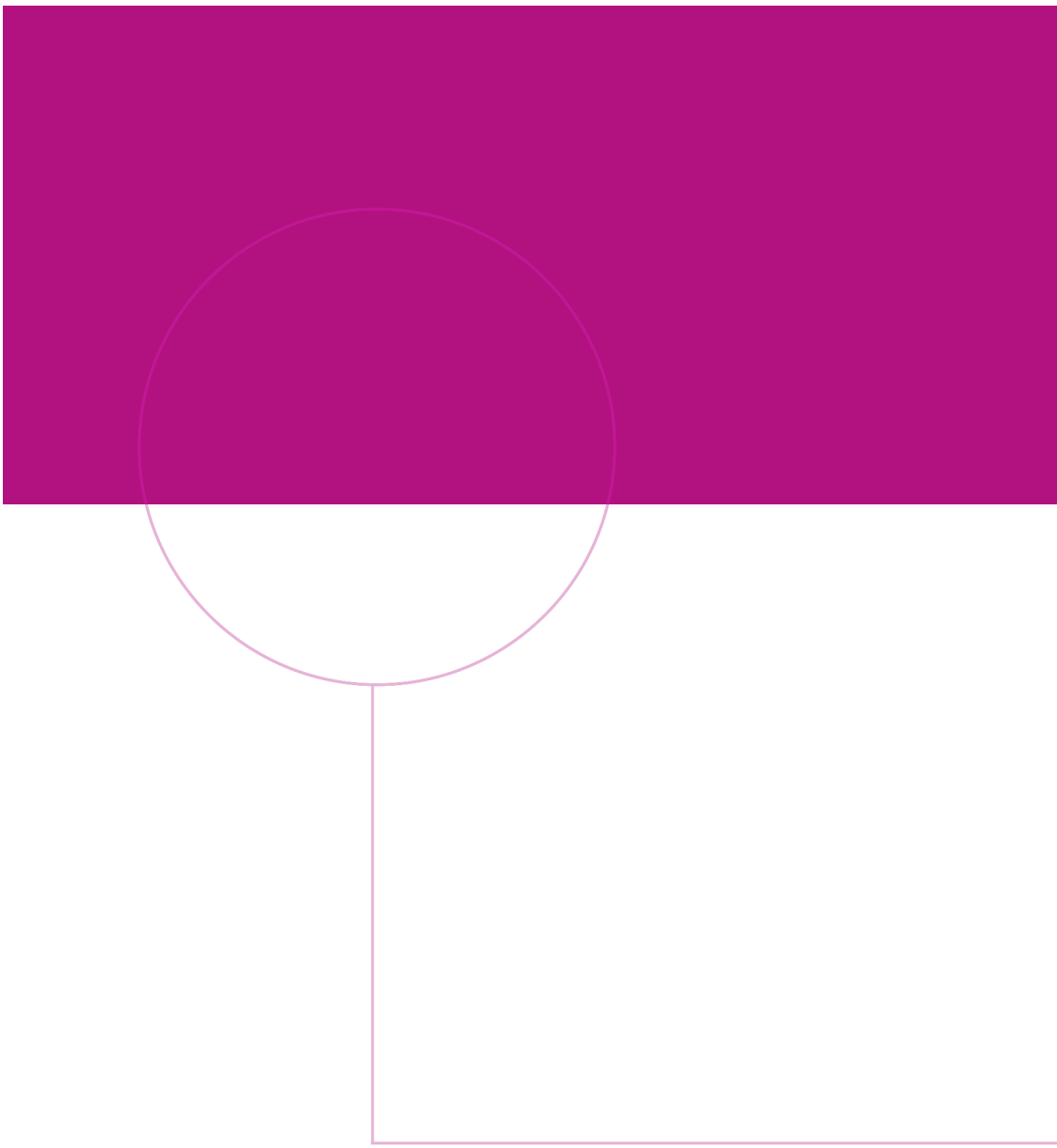
.

9. Kontinuerlig integrasjon og testing

Kontinuerlig Integrasjon er satt opp gjennom Travis CI for både Java og Javascript koden. Begge kjører på Linux. For Java delen er testene laget med JUnit, og kjøres med JDK versjon 11 på pipelinen. JavaScript (frontend) testene er laget med Jest, og består av både enhetstester og snapshot tester. Snapshot-tester lagrer en snapshot av forskjellige komponenter sammen med testene og sjekker at de er like som den tilhørende komponenten når testen kjøres. Dette er for å passe på at brukergrensesnittet ikke har endret seg uventet.

For både backend og frontend genereres det rapport av testdekning på pipelinen, om man vil ha denne rapporten i HTML format for frontend, må man kjøre testene lokalt med *npm run test.*

Backend testes ved å kompilere og kjøre testfilene ServerTest.java og HttpRoadSignDaoTest.java. Frontend testes ved å kjøre kommandoen *npm run test* i mappen “skilt-info”. HTML for dekning av testene i backend ligger i mappen Java/jacoco.



NTNU

Kunnskap for en bedre verden