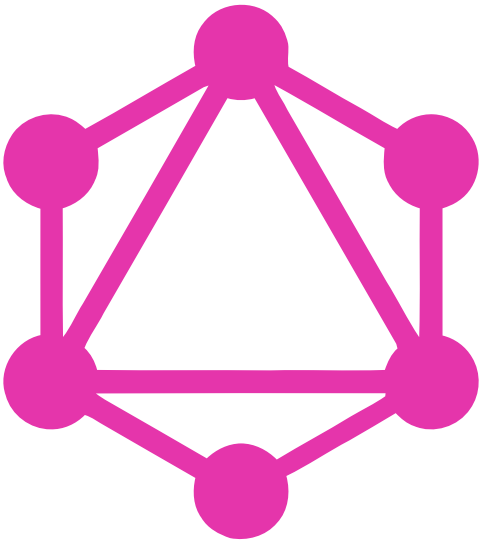
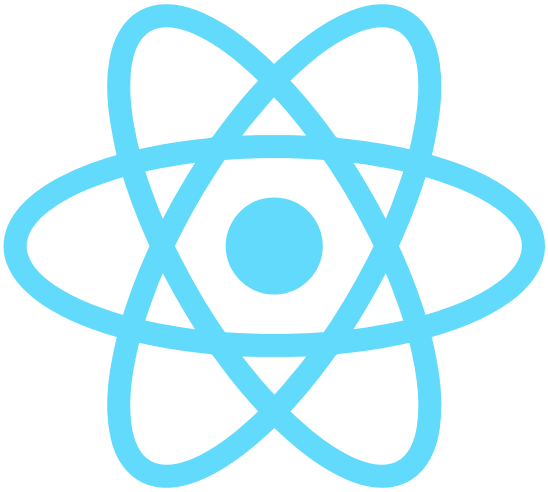
CHALMERS





Server- och klientkommunikation med implementering av GraphQL för applikation i React Native

Kandidatarbete i Computer Science and Engineering

ALBIN SJÖSTRAND
JAKUB DUDEK

Department of Computer Science and Engineering
CHALMERS TEKNISKA HÖGSKOLA OCH GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

KANDIDATARBETE

Server- och klientkommunikation med implementering av GraphQL för applikation i React Native

ALBIN SJÖSTRAND
JAKUB DUDEK

Department of Computer Science and Engineering
CHALMERS TEKNISKA HÖGSKOLA OCH GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2019

Server- och klientkommunikation med implementering av GraphQL för applikation i React Native
ALBIN SJÖSTRAND
JAKUB DUDEK

Examinator: Jonas Almström Duregård
ISSN 1654-4676

Department of Computer Science and Engineering
Chalmers tekniska högskola och Göteborgs universitet
SE-412 96 Göteborg
Sverige
Telefon: +46 (0)31-772 1000

Omslag:
Logotyp för GraphQL och React Native

Chalmers Reproservice
Göteborg, Sverige 2019

Server- och klientkommunikation med implementering av GraphQL för applikation i React Native

ALBIN SJÖSTRAND
JAKUB DUDEK
Department of Computer Science and Engineering, Chalmers tekniska högskola och Göteborgs universitet

Kandidatarbete

Sammanfattning

Decerno som har byggt systemet Ledningskollen vill effektivisera arbetsprocessen och undersöka om nya teknologier kan öka prestandan för server- och klientkommunikation där utsättningar av ledningar ska ske för grävområden. Processen digitaliseras med en applikation byggd i React Native för att ersätta manuella sidor som tidigare krävt att PDF:er med information om områden behövs skrivas ut. Genom utveckling i React Native krävs mindre arbete för att skapandet av en applikation för flera operativsystem, i form av iOS och Android. Storybook används för att strömlinjeforma utvecklingen av komponenter av applikationen vilket medför modularitet inom och utanför applikationen, med återanvändning av färdiga element. För att kommunikationen ska ha bättre prestanda implementeras GraphQL med Apollo i server-delen av applikationen som sammankopplar databaser och APIer på ett mer effektivt sätt än vad traditionella REST APIer tidigare gjort. Det innebär att fördelarna med GraphQL undersöks för att minska latensen och overfetching vid förfrågningar mellan servern och klienten. Utsättningar kan därför ske mer effektivt där tydlig och relevant information om områden markerade för utsättning visualiseras i applikationen, i form av översiktlig information samt detaljerade beskrivningar av områden och kartor med ledningar. Resultatet visar på att GraphQL och React Native har underlättat arbetsgången med minskat beroende av papper och samtal, ersatt med samlad information i en och samma applikation. Ledningar på områden kan identifieras och minska problem där ledningar skadas. Det visar även på att GraphQL med Apollo kan implementeras i både nya och existerande system med en bekvämare utvecklingsprocessen samt fördelarna jämfört med problem som REST APIer innebär för prestanda i form av bättre svarstider och direkta svar på förfrågningar av information från datakällor.

Nyckelord: Ledningkollen, Utsättning, Ledningar, GraphQL, Apollo, REST API, Latens, Overfetching, React Native, JavaScript, Applikation, iOS, Android, NodeJS, Storybook

Förord

Examnsarbetet utfördes vid institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola.

Vi vill tacka Decerno för möjligheten att utföra det här projektet och särskild tack till Henrik Karlsson för all hjälp, handledning och varmt bemötande under vår tid på företaget.

Tack till Uno Holmer för handledning från Chalmers.

Förkortningslista

- NPM - Node Package Manager
- API - Application Programming Interface
- JS - JavaScript
- REST - Representational State Transfer
- App - Applikation
- UI - User Interface
- MVP - Minimal Viable Product
- IDE - Integrated Development Environment
- VPN - Virtual Private Network

Ordlista

- Syntax - Hur programmeringspråket ser ut
- Mockup - Virtuella designförslag
- Boilerplate - Basstruktur för kod
- I/O - Interfaceineraktion med enheter
- Endpoint - Slutnod för kommunikation
- Mocking - Att skapa funktioner utan implementerad backend

Innehåll

1 Inledning

Det här avsnittet behandlar bakgrunden för projektet, vad anledningen är till att utforska nya teknologier, och även syftet med att implementera de teknologierna i en applikation. Här ställs målen upp för att ge en tydligare blick över vad projektet resultat skulle gå emot och även var avgränsningarna går. Dessa avgränsningar är relaterade till hur utvecklingen i projektet skulle utföras.

1.1 Bakgrund

Decerno är ett konsultföretag som gör skräddarsydda lösningar åt sina kunder. Utvecklingen sker i uppdragsform och största delen av arbetet sker hos företaget med undantag för besök hos kunder för att ta del av miljön som systemet kommer att befinna sig i. Systemet som projektet bygger på, Ledningskollen, är en tjänst för företag och privatpersoner som ska utföra grävararbeten i marken.

I dagens läge skapar de en förfrågan hos Ledningskollen som jämför det område som ska behandlas med information om var ledningar går i marken och vem som är ledningsägaren. Om en ledningsägare som får ett ärende har ledningar nära arbetsområdet kan de besluta om att genomföra en utsättning på plats. I dagens läge sker det genom att ledningsägaren skickar ut en utsättare som med hjälp av en utskriven PDF med information och karta, hittar och markerar ledningarna genom utritning på mark eller placering av pinnar. Problemen med denna hantering är att stor del av processen sker manuellt, med papperskopior och samtal till de inblandade parterna.

Uppgiften kommer lösas genom att sammanställa nuvarande databaser samt API:er som tillhör Ledningskollen med hjälp av GraphQL och framställa en applikation för utsättare där utsättaren kan se sin egen position, ledningsägarens ledningar och arbetsområdena där markarbeten ska genomföras.

1.2 Syfte

Projektet ska utföras för att digitalisera manuella sidor av processen för utsättningar. Decerno vill kunna erbjuda en tjänst som förenklar processen för både den som ska genomföra markarbeten (kallad frågare) och den som äger ledningar (kallad ledningsägare). De huvudsakliga vinsterna genom digitaliseringen bedöms vara att personen som ska genomföra utsättningen kan använda GPS för att säkerställa att utsättning sker i rätt område. Detta eftersom fältapplikationen visar såväl utsättarens position, ledningarna som ska markeras ut samt arbetsområdet där markarbeten ska genomföras. Syftet är även att utforska nya teknologier i form av GraphQL inom kommunikationen för klient och server samt att använda sig av nya metoder för att bygga applikationer, med React Native.

1.3 Mål

- Framställa en applikation där utsättaren kan se sin egen position, arbetsområden för planerade markarbeten och ledningsägarens ledningar
- Utveckla en backend som med hjälp av GraphQL sammankopplar databaser och API:er som krävs för att applikationen ska fungera
- Undersöka fördelar med GraphQL jämfört med ett traditionellt REST API
- Framställa applikationen med ramverket React Native

1.4 Avgränsningar

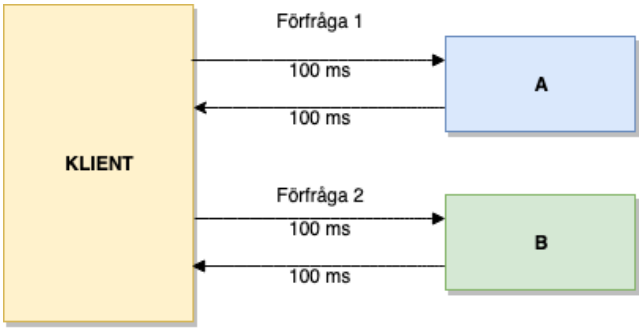
Projektet avser inte att behandla skapandet av ärenden eller lagring av informationen från databaser eller APIer. Applikationen är inte utvecklad till att vara tillgänglig för datorer, surfplattor eller andra operativsystem utöver iOS och Android. Projektet är avgränsat till att endast vara en prototyp av en färdig applikation vilket innebär mindre vikt har lagts på testning och validering av systemet.

2 Teknisk Bakgrund

Teknisk bakgrund beskriver olika teknologier, verktyg och arbetssätt som användes i projektet och utvecklingen. Här beskrivs alla system och produkter som använts i projektet vilket återkommer i resterande rapporten.

2.1 GraphQL

Kommunikationen mellan server och klient har traditionellt byggs med hjälp av REST APIer, ett exempel av detta kan i figur 2.1. Det är en arkitektur som man kan hitta implementerat i många applikationer idag men som tar med sig två nackdelar. Den första kallas för 'overfetching' och uppstår när en klient är tvungen att hämta mer data än den kommer använda sig av (se figur 2.2). Klienten behöver bara temperatur men kan ej specificera det. Detta i sin tur innebär att extra latens uppstår och klienten är tvungen att utföra extra arbete genom att filtrera ut det överflödiga. Den andra nackdelen är att kommunikationen mellan server och klient går fram och tillbaka ett flertal gånger.



Figur 2.1: Klienten vill ha svar på förfråga 2 från endpoint B men väntar på svar på fråga 1 från endpoint A. Detta skapar latens vid kommunikationen mellan klienten och endpoints.

Förfråga

```
GET /API/WEATHER/Stockholm
```

Svar

```
country: "Sweden",
city: "Stockholm",
temperature: 21,
wind: "4 m/s",
humidityPercent: 84,
precipitationPercent: 93,
cloudsPercent: 64,
airpreasurePercent: 79
```

Förfråga

```
query {
  weather(city:"Stockholm"){
      temperature,
      precipitationPercent
}
```

Svar

```
{
   "data":{
      "temperature":21,
      "precipitationPercent":93
   }
}
```

Figur 2.2: Vänster: Exempel på REST-förfråga . Höger: Samma förfråga med GraphQL där klienten frågar bara efter temperatur och regn

GraphQL är ett modernt alternativ till REST API. Det är en specifikation som beskriver syntaxen för ett Query-språk (se figur 2.2) samt en exekveringsmiljö. Teknologin tillåter klienten att formulera ett förfråga som innehåller vad för data som systemet behöver samt hur den ska struktureras. Förfrågan besvaras sedan av servern som speglar strukturen och returnerar endast data som efterfrågas. Vid typisk implementation av

GraphQL så agerar den som en tratt för alla applikationens datakällor, servern sammanställer databaser och APIer så att de tillgängliga under en och samma endpoint som betyder att vi bara behöver skicka ett förfråga. GraphQL implementerar även ett typ av system som kräver att utvecklaren typsätter allt data som kommer skickas som bland annat gör det enklare för testning av systemet och validering av förfrågningar Det är viktigt att påpeka att GraphQL är bara en specifikation som innebär att flera olika implementationer av teknologin existerar och finns i många olika programmerings språk, dessa kan innehålla olika extra funktionalitet men alla håller sig till samma specifikation och i grunden beter sig på samma sätt. [15]

2.1.1 Apollo GraphQL

Apollo är en implementation av GraphQL för utveckling tillsammans med web- och mobilapplikationer. En serverdel är inkluderad som är skriven i JavaScript och olika klientversioner vilka är anpassade till olika ramverk. Utöver det man förväntar sig av ett GraphQL implementation som beskrevs ovan implementerar Apollo även funktionalitet som caching, felhantering, övervakning och mocking. [5]

2.1.2 GraphQL Playground

En GraphQL IDE som tillåter interaktion med en GraphQL-server utan att behöva implementera en egen klient. Verktyget tillåter att testa miljön på ett smart sätt genom att kunna strukturera förfrågningar till servern, mäta latens och analysera den automatiskt genererade dokumentationen. Programmet kan användas som en fristående applikation som körs direkt på enheten eller så kommer den bunden med olika implementationer av GraphQL så som Apollo där den är tillgänglig via en webbläsare och agerar som en webbapplikation. [10][15]

2.2 Node.js

Node.js är en exekveringsmiljö i JavaScript med öppen källkod som tillåter utvecklare att köra JavaScript-kod utanför en webbläsare. Den använder sig av Chromes V8-motor som står för exekvering av JavaScript samt implementerar ett lager av I/O-bibliotek för att interagera med systemet och för att skapa en effektiv miljö för att skapa skript och serverprogrammering som är jämförbar med Java och Python. [7]

2.2.1 NPM

NPM tillåter programmerare att dela med sig av JavaScript-paket de har skrivit och lägga upp de på ett gemensamt repository där andra användare kan med hjälp av NPM ladda ner paket och inkludera de i sina egna projekt. Idag finns det över 450 000 paket uppladdade av olika utvecklare och NPM är ett grundläggande verktyg i många fall nödvändig vid arbete med Node.js.[18]

2.2.1.1 Lodash

Ett välkänt NPM-bibliotek som underlättar JavaScriptprogrammering genom att inkludera en stor mängd funktioner för manipulering av vektorer, objekt, strängar och andra datatyper.[1]

2.2.1.2 Geolib

Ett bibliotek som innehåller funktioner för beräkningar koordinater med hjälp av latitud och longitud. [11]

2.3 Express.js

Express.js är ett minimalt ramverk för utveckling av JavaScript webbapplikationer. Den tillåter att snabbt sätta upp en hemsida eller en server och göra den tillgänglig på nätet. Ramverket använder sig av de inbyggda HTTP-bibliotek som finns i Node.js men gömmer det från utvecklaren vilket gör det enklare att utveckla, med bara något rader kod kan man initiera en server och göra den tillgänglig på nätverk via en port. [3]

2.4 EPSG

EPSG är en databas av koordinatsystem framställda av diverse verksamheter, dessa beskriver hur olika värden i respektive system refererar till olika platser på kartor. Värden kan användas för att framställa kartor och identifiera olika positioner. Ett exempel är den vanligaste EPSG:4326 som refererar till latitud och longitud systemet. [21]

2.4.1 Transform Coordinates

NPM paket för omvandling av koordinater mellan olika EPSG-system. [22]

2.5 DBeaver

En SQL-klient som möjliggör sökningar av SQL-databaser och utföra Querys mot de för att ändra och undersöka innehållet i de samt ändra på den. Programmet är gratis, med öppen källkod och tillåter att interagera med de flesta typer av databaser till exempel MySQL, PostgreSQL, Oracle eller MariaDB. [6]

2.6 Scrum

Scrum är ett agilt arbetssätt som innebär att applikationen delas upp i mindre segment, för att kunna enkelt närma sig en MVP genom att arbeta med utvecklingen på ett vertikalt plan. Det ska finnas värde i att varje komponent eller funktion ska ha ett syfte som är sammankopplat till en annan del av applikationen, där värdet sitter i att den delen av applikationen är användbar. Det innebär att en MVP kan sättas utifrån vad för funktioner användare finner viktiga utan att utveckling fokuserar på att få ett komplett resultat, där funktioner existerar utan något värde.

Att arbeta med Scrum innebär även att vid varje arbetstillfälle utgår parterna i utveckling sätter mål som ska göras inför en viss period, veckovis eller längre tid beroende på storlek av projekt. Dessa möten brukar i Scrum-sammanhang kallas för 'stand-up meetings' med tydlig agenda om vad som hittills skett under utvecklingen, vad som kommer göras under kommande arbetsgång samt om förändringar i målsättningen ska ske. Dessa ändringar kan komma från insikter under utvecklingens gång av utvecklare eller av parter av intresse som anser att utvecklingen bör sträva åt ett annat håll.[20]

2.6.1 Trello

Trello är ett verktyg för att enkelt kunna organisera och visualisera hur utvecklingen sker. Det går att jämföra med att använda sig av lappar som flyttas mellan olika stadier i processen för utveckling. Det skiljer sig från projekt till projekt genom att olika stadier är olika viktiga, där fokus kan läggas på exempelvis validering eller finslipning. Generellt delas dessa stadier upp vad som ska göras och av vem, pågående arbete samt avklarade delar. [2]

2.7 React Native

React Native är ett ramverk med öppen källkod som tillåter utvecklare att skapa mobila applikationer till både iOS och Android med en gemensam kodbas, till skillnad från många projekt där man har två kodbaser skrivna i Java och Swift för respektive plattform. Ramverket baserar sig på den populära webbteknologin React och drar sin struktur och funktionalitet därifrån. Utöver det som React erbjuder så tillåter React Native även interagera med telefonens sensorer, kamera, minne och annan I/O funktionalitet.

För att programmet ska kunna exekvera på de olika enheterna har Facebook, som är företaget bakom React Native, utvecklat en brygga som tar kommandon i JavaScript och sparar dem som ett meddelande, som i sin tur sparas i en kö. På andra sidan av bryggan finns ett interface som kan interagera med själva enheten och utföra de olika kommandon i till ex java eller Objective-C. Interfacet hämtar de meddelande som ligger i kön och ser till att de blir exekverade.

Den här typen av strategi innebär att koden i React Native kan exekveras på ett stort utbud av enheter så länge ett interface är utvecklat kan exekvera medlandena som har lagrats i kön. [8]

2.7.1 Ignite CLI

Ignite är CLI verktyg utvecklat av företaget Infinite Red som tillåter utvecklare att snabbt starta upp ett projekt utifrån en färdig mall. Verktyget erbjuder bibliotek som kan väljas ifrån och installeras automatiskt i projektet. Eftersom Infinite Red är ett företag som utvecklar många applikationer i React Native har de en stor erfarenhet i hur en applikation kan struktureras och vilka verktyg som är bra att ha med sig under utvecklingen, allt för att få en snabbare uppstartningstid av projekt och minskar tiden som utvecklare behöver lägga ner för att skapa projektstrukturen själv. [13]

2.7.2 React Native Maps

Ett väldigt populärt React Native-bibliotek som gör det enkelt att implementera kartor i applikationen som baseras på data från antigen Google Maps med hjälp av ett Google API nyckel eller från Apple Maps som bara fungerar på iOS-enheter. Biblioteket innehåller funktionalitet som låter dig sätta ut markeringar på kartan, rita figurer och navigera till olika destinationer genom att ange koordinater. [19]

2.8 Storybook

Storybook är en utvecklingsmiljö och ett verktyg för UI-komponenter som hjälper utvecklare att skapa komponenter oberoende av en applikation. De kan testas i en testmiljö för att bekräfta deras beteende i en applikation och är isolerad från applikationen vilket inte påverkar befintliga komponentberoenden. Storybook möjliggör skapandet av modulära komponenter som kan återanvändas i befintliga och nya projekt samt att det avlastar utvecklare att behöva skapa nya komponenter för varje ny funktion. [14]

2.9 Invision Studio

Invision Studio är en produkt för att skapa designförslag av applikationer och hemsidor innan utveckling sker av en applikation. Designen är vektorbaserad och kan anpassas till vilken typ av enhet som applikationen ska vara till samt att en simulation kan testas för att bekräfta beteendet och navigering. Verktyget används för att visualisera idéer och designförslag för att enkelt kunna bekräfta eller ändra på nuvarande design. [16]

2.10 Material Design

Material Design är ett visuellt språk, utvecklat av Google, som använder sig av principer för design med innovationer av teknologi och vetenskap. Principerna tar inspiration från hur fysiska objekt reflekterar ljus och kastar skuggor. Det innefattar även typografi, mellanrum, skalning, färger och bilders placering. Material Design har ett gemensamt utseende oberoende av plattform som iOS, Android eller webapplikationer. [4]

2.11 Xcode

Xcode är en IDE utvecklat av Apple och specifikt för utveckling på iOS-enheter och kan användas för att skapa iOS-specifika ikoner och uppstartsskärmar. Xcode har en iOS-simulator inbyggd för testning oberoende av en fysisk enhet.[24]

2.12 Android Studio

Android Studio är en IDE till för utveckling av Android-applikationer och inkluderar simulering av Android-enheter.[17]

2.13 Visual Studio Code

Visual Studio Code, förkortat VS Code, är en utvecklingsmiljö som inte är språkberoende utan är anpassningsbar efter befintliga projekt. [9]

2.13.1 Prettier

Ett verktyg som tillåter programmeraren att med hjälp av en konfigureringsfil bestämma hur kodbasen ska se ut och sedan med hjälp av verktyget automatiskt strukturera om koden så att den följer de regler som satts upp. Detta gör att hela kodbasen är väldigt enkelt att läsa och har samma struktur genom hela projektet. Prettier kan användas tillsammans med VS Code som innebär att koden struktureras varje gång man sparar en fil. [23]

2.14 GitHub

GitHub är ett verktyg för att molnbaserat spara kod tillgänglig för att utvecklare i ett projekt med flödesscheman, versionshantering samt lättåtkomlig dokumentation associerad med specifika filer.[12]

3 Metod

I det här avsnittet beskrivs arbetsprocessen för projektet. Det inkluderar var utvecklingen tog plats, vilket material som användes samt flöden för Storybook och Scrum.

3.1 Arbetsplats

Projektet utfördes på kontoret hos konsultföretaget Decerno i centrala Göteborg på arbetsplatser i ett öppet kontorslandskap med tilldelade arbetsplatser och möjlighet för projektering i konferensrum.

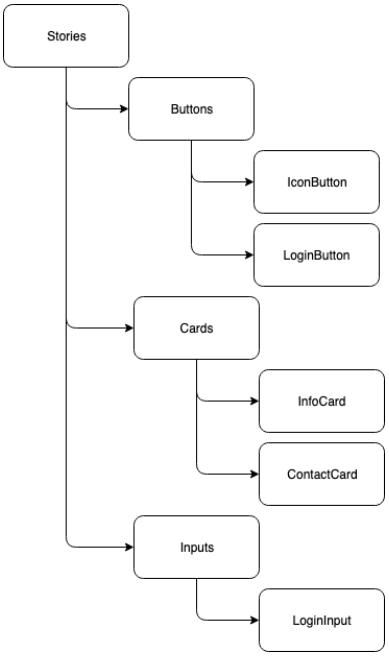
3.2 Material

Utvecklingen skedde på Apple MacBooks försedda av Decerno för att kunna utveckla till iOS-enheter, vilket krävs för att kunna simulera applikationen.

För att kunna använda sig av databaser och system var det tvunget att datorerna var kopplade till det interna nätverket. För att utvecklingen skulle kunna ske på distans var datorerna utrustade med VPN för att ansluta till nätverket.

3.3 Storybookflöde

Utvecklingen av UI skedde med verktyget Storybook vilket möjliggjorde att arbetet kunde delas upp under det tidigare stadiet av projektet. Komponenter kunde skapas och testas separat, utan vetskap om andra funktioner i applikationen, och kunde sedan implementeras. Storybook gjorde det genomförbart att komponenterna kunde skapas innan applikationen skapats och vara baserat på design som tidigare tagits fram. Utvecklingen kunde då fokusera på att skapa en server samt backgrundsprocesserna för klienten. Med en tydlig mappstruktur var det enkelt att förstå sig på vad för komponent som tillhörde vilken del av applikationen och vad dess funktioner var (se figur 3.1).



Figur 3.1: Mappstrukturering av Storybook för bättre översikt för komponenter i en applikation. Stories är samlingsnamnet för mappen för komponenter som har utvecklats med Storybook.

3.4 Scrum

Arbetsgången applicerade det agila arbetssättet Scrum genom att ha dagliga möten om vad som ska göras och vad som har gjort. Under dessa möten prioriterades vilka kommande uppgifter efter de uppsatta målen och teammedlemmar tilldelades uppgifter. Varannan vecka var mötet tillsammans med handledaren på företaget för att diskutera vad som åstadkommits, om några tillägg eller ändringar skulle göras samt att diskutera vart målet i utvecklingen var för nästa möte. Uppgifterna fick ingen ytligare tyngd bortsätt mot prioriteringen i hur svår uppgifterna var att uppfylla, utan om de slutfördes innan nästa tillfälle fortsatte utvecklingen på en ny uppgift efter prioritering. Genom att applicera Scrum, även om inte strikt följt, skapades slutdatum för när uppgifter skulle vara klara.

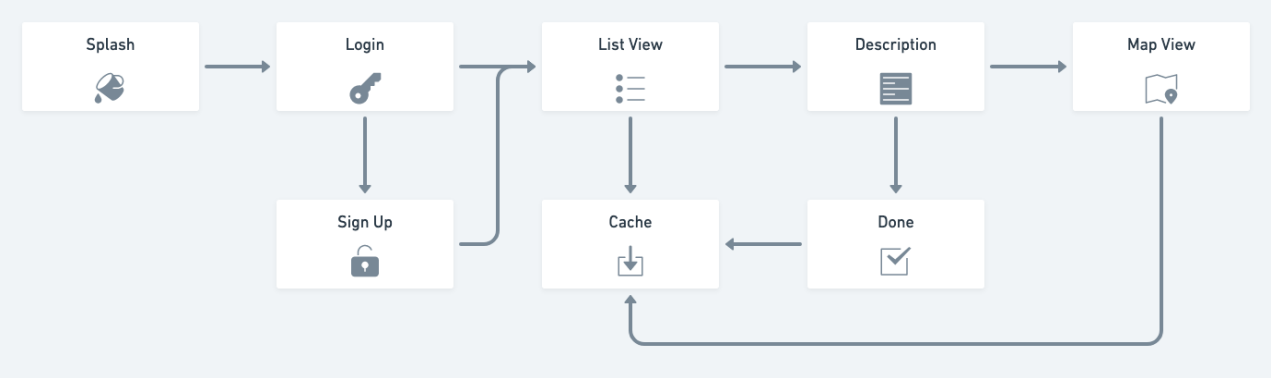
Trello användes för att visualisera Scrum genom att använda olika kolumner för de stadier uppgifterna var i. Första kolumnen innehöll större delar av projektet som skulle utföras och delas upp till mindre uppgifter, så som applikationens komponenter, möten eller forskning. Den andra kolumnen innehöll de mindre uppdelade uppgifterna, till vilken del av projektet de tillhörde och vem som var ansvarig för att utföra arbetet. Dessa uppgifter kunde sedan flyttas över till nästa kolumn när de var klara, specifikt för den dagen de avklarades. Efter dagens arbete lades färdiga uppgifterna i en ny kolumn med allt som var avklarat i avklarad kronologisk ordning.

4 Genomförande

Avsnittet om genomförande behandlar planeringen av projektet, hur det struktureras för att få en bättre bild över vad som ska inkludera i systemet och applikationen. Genomförande inkluderar utformningen för designen av applikationen, i vilka steg som servern utvecklades samt framtagningen av klienten för kommunikationen i form av en applikation.

4.1 Planering

Projektet initierades med att lägga upp en planering utefter krav som ställts av initiativtagaren av att undersöka nya teknologier på företaget, genom att sätta upp mål om vad som utvecklas och till vilken. Kraven var baserade på att få fram en prototyp som kunde demonstrera hur en slutgiltig produkt skulle kunna fungera och samt om teknologin bakom, med GraphQL och React Native, motiverade för att investerade i framtida utveckling. Planeringen är till för att diskutera funktionaliteter och programflöden inom systemet för att förtydliga var fokus ska läggas (se figur 4.1).



Figur 4.1: Flödesschema som framtogs under planeringen, beskriver navigeringen inom applikationen med vyer sammankopplat med offline-stöd i form av cache.

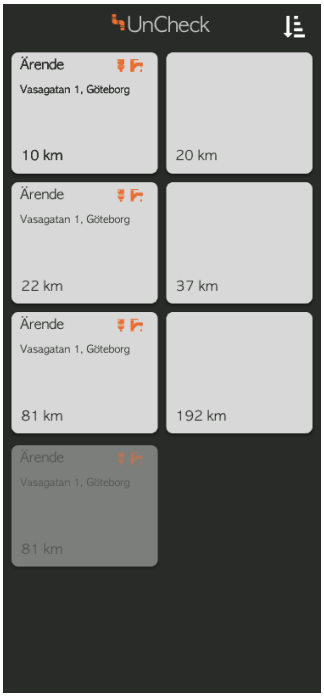
4.2 Design

Designen framtogs i form av att applikationen skulle ha en god översikt av all information som är riktad till användarna samt att förenkla processen av arbetsgången. Idéer lades fram om vad målet skulle vara med olika prioriteringar genom att anpassa utformandet efter mål från både utvecklare och handledaren. Det resulterade att vissa funktioner skulle kunna göras om det fanns tid över efter de funktionerna med högre prioritet.

Brainstorming användes för att ta fram designen i grövre drag, baserat på tidigare fastställning av målsättningar. Genom att skissa med papper och penna skapades de första idéerna om vad applikationen skulle kunna innehålla, och därmed få ett första intryck om alla personer delaktiga inom projektet hade samma visioner. Dessa skisser kunde sedan modifieras så att alla parter kände att funktioner och komponenter i applikationen fyllde sitt syfte. Skisserna kunde sedan användas som bas för att skapa designförslag för hur utformningen skulle se ut. Det betydde även att dessa skisser användes föra att gå tillbaka till de originalidéerna under senare delar av designstadiet för att fastställa att utvecklingen skedde i rätt riktning.

Som grund till den slutgiltiga designen av applikationen skapades en mockup med verktyget Invision Studio. Den baserades tidigare framtagna skisser och vilka funktioner som tillhörde vilken del av applikationen. Färger togs fram baserat på färgerna för Ledningskollens logotyp samt från befintliga standarder inom Material Design (se figur 4.2). Accentfärgen från Ledningskollen anpassades med dessa principer för att både följa ett uniformt utseende men även för att designen skulle vara lätt att navigera genom och innehållet skulle vara anpassat efter vad användare skulle lägga fokus på. Därför valdes att följa den framtagna standarden i Material Design om att använda kort, 'cards' i dokumenteringen (se figur 4.3). Dessa kort skulle vara till för att innehåll tillhörande olika objekt skulle vara lätt urskiljbart samt att sträva bort från utdaterade designmönster som listor. Ikoner som användes var även från utbudet inom Material Design, utformade efter att vara tydliga och med genomgående design, specifika för de funktioner som de skulle inneha. Ikonerna användes för komponenter som knappar och för att indikera var olika typer av information fanns genom att skapa fokus för användaren. Eftersom Material Design har öppen källkod kan dessa ikoner enkelt anpassas efter färger och gränssnitt. För att fortsätta ha ett genomgående tema av Material Design användes typsnitt som var framtaget för dessa designprinciper. Eftersom designmönstret generellt är platt används skuggor för att lyfta fram vart interaktioner kan ske, så som vilka kort eller knappar som gick att interagera med.

Figur 4.2: Färgschema

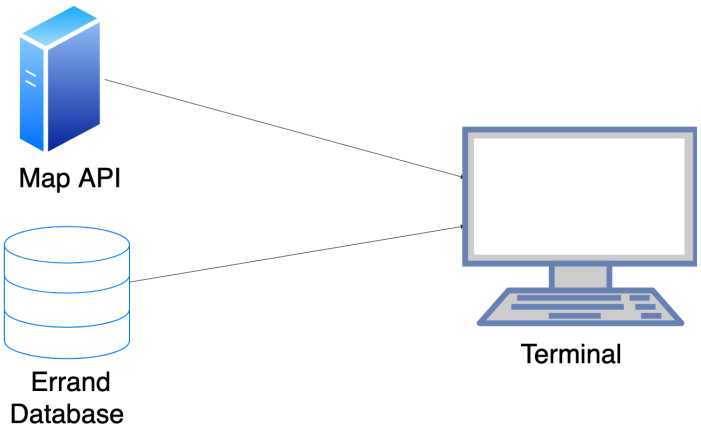


Figur 4.3: Desingprototyp för listvy

4.3 Server

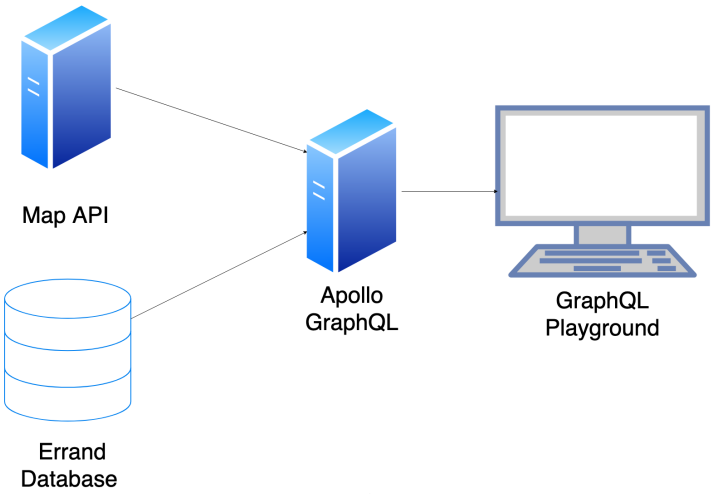
Parallellt till utvecklingen av designen för applikationen började arbetet på serverdelen av projektet. Ramverket Express.js användes med motivation att det är väldigt enkelt att arbeta med och abstraherar bort en stor del av de utmaningar som kan stötas på när uppstrukturering av backend ska ske effektivt. Express.js har även en uppsättning av färdigutvecklade tillägg med funktionalitet som felhantering och loggning som eftertraktas för enklare utveckling och för debugging.

Den första iterationen av servern använde sig inte av GraphQL-funktionalitet utan utvecklades för att testa kommunikationen med databasen som innehåller all data kring ärenden och APIet med ledningsdatan. Verktyget DBeaver användes för att navigera genom databasen och konstruera en SQL-Query som hämtade data från rätt tabell och kunde sedan kopieras över till koden för servern. Liknande process utfördes för APIet med ledningar men istället verktyget Postman för att framställa förfrågan. När servern kunde skriva ut informationen från källorna till en terminal ansågs det att kommunikationen för servern var tillräckligt och projektet kunde fortsätta att implementera kommunikationen till klienten samt fortsätta att skapa fler funktioner.



Figur 4.4: Första iteration av servern

Nästa steg i utvecklingen av servern låg fokus på att använda data som servern hämtar och göra den tillgänglig via en GraphQL-endpoint. Det underlättades främst av den välskrivna dokumentationen och guiden som är tillgänglig för Apollo [5]. Programmeringen för backenden påbörjades med att framställa ett schema, den del av koden som beskriver hur data ska se ut, med hur olika datapunkterna i källorna relaterar till varandra och vilka datatyper de är som exempelvis Integer, Float, String och Object. Det krävs noggrann genomgång av alla datakällor i projektet och undersökning kring vad de innehåller och hur de relaterar till varandra. För att klienten ska kunna hämta informationen enligt scheman som definierades fortsatte utvecklingen med att implementera vad Apollo kallar för 'resolvers'. GraphQL ska tolka förfrågan som kommer från klienten och hämta, filtrera och returnera rätt data som efterfrågas baserat på typerna från scheman. Under det här stadiet sammankopplas koden som utvecklades i den första iterationen för servern.



Figur 4.5: Andra iteration av servern

Vid det här stadiet av projektet har servern tillräcklig basfunktionalitet som efterfrågats under planeringen. Under testningen upptäcktes det att flera förfrågningar till företagets REST APIer orsakar väldigt stor latens. För att åtgärda detta implementerades det ett cache, genom att använda Apollos inbyggda REST-bibliotek istället för den inbyggda i Node.js, Apollo sköter cachingen automatiskt med väldigt lite konfigurering vilket påverkade kodbasen minimalt men hade väldigt stor påverkan på att öka prestandan. Efter att cachingen var avklarad så var det endast den första förfrågan som tog lång tid då minnet behövdes fyllas. Men i och med dessa förändringar reducerades förfrågningstiden från tiotal sekunder till att ligga under 200 millisekunder.

Utöver detta infördes små ändringar i backendens kod då det kom upp nya krav under utvecklingen av applikationen. En av de var en funktion som omvandlar koordinater i databasen från standarden EPSG 3006 till 4326, vilket är det koordinatformatet som React Native maps använder.

4.4 Klient

Utvecklingen av applikationen påbörjades när servern var i det stadiet där den kunde erbjuda funktionalitet från GraphQL. Då syftet med applikationen var främst att undersöka GraphQL och påverkan på system som tidigare använt sig av REST APIer bestämdes en utvecklingsstrategi kallad 'bottom-up' skulle utnyttjas. Utvecklingen skulle då börja med kartvyn i applikationen som skulle vara den del där en den nya teknologin med GraphQL skulle ha mer användning samt vad som kräver mest uppgradering för prestanda.

Projektet använde sig även av StoryBook så att arbetet kunde delas upp i frontend- och backendutvecklingen för att fortsätta arbetet parallellt. För det visuella av applikationen kunde komponenter skapas och testas i en sandlådemiljö utan att vara beroende på hur funktionaliteten och strukturen såg ut i huvudapplikationen. Genom strategin av uppdelning mellan frontend och backend kunde det enkelt samarbeta med att beståndsdelar togs fram utefter de funktioner som den slutgiltiga applikationen krävde. Funktioner kunde sedan utvecklas och när de visuella delarna skulle sammankopplas var de färdiga, allt för att föra projektet framåt utan att behöva ett steg i taget för varje funktion och komponent. Det underlättade även för projektet att utnyttja de olika kunskaperna inom teamet på fullaste sätt genom att strukturera utvecklingen efter inom vilket område störst kompetens fanns.

Efter att ha framställt kartvyn som successivt hämtar och visar information från servern fortsatte arbetet med listvyn som hämtar en alla ärenden i systemet i en lista och låter användaren interagera med komponenterna. Listvyns innehåll var sedan sammankopplat med att användaren navigerades till respektive karta för varje ärende. Listvyn låter även användaren att uppdatera innehåller genom att utföra gesten att svepa ner över listan och på så sätt få senaste information från databasen som kan ha förändrats. Tack vare dokumentering för React Native kunde implementeringen av funktionalitet enkelt utföras då urvalet av applikationsegenskaper är stort och lättillgängligt för alla utvecklare.

5 Resultat

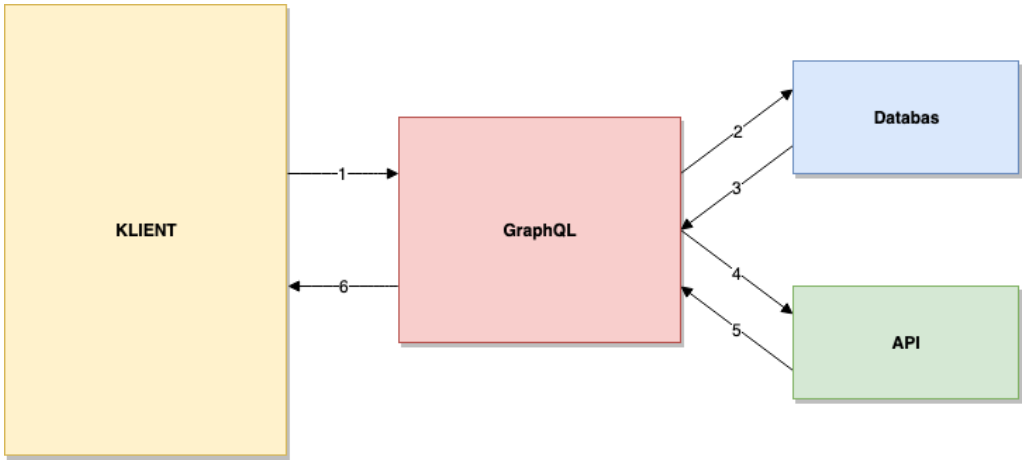
Avsnittet om resultat beskriver vad som togs fram under planeringen av projektet, resultatet av servern med GraphQL samt applikationen i React Native med dess funktioner, vyer och komponenter.

5.1 Systemarkitektur

Den resulterade systemarkitekturen blev väldigt simpel men innehöll alla nödvändiga komponenter för att undersöka GraphQL och dess fördelar gentemot traditionella REST APIer. I systemet hittar vi en Microsoft SQL databas som innehåller data för alla ärende såsom position, namn, ID eller datum, och ett API som tar emot koordinater från databasen och skickar tillbaka information om ledningar som befinner sig inom koordinaterna. GraphQL sammanställer de två datakällorna och sköter interaktionen mellan de.

5.2 GraphQL

Den färdiga implementationen av GraphQL agerar som en tratt vilket tillåter klienten att på ett enkelt sätt få tag på data från datakällorna utan att behöva ta hänsyn till att den kommer från flera olika källor. Det innebär att i de flesta fallen räcker det med bara en förfråga jämfört med ett REST API där vi hade behövt minst två.



Figur 5.1: En förfråga skickas till GraphQL som sedan bryter ner det och sköter resten av kommunikationen. Se figur 2.1 för respektive lösning med REST

Systemet som har utvecklats tillåter två olika förfrågningar, den första låter klienten ange ett ID på det ärendet vi är ute efter och det andra tar inget argument och returnerar en lista på alla ärende och respektive data. Klienten kan även filtrera vilka parametrar från respektive ärende som ska returneras och kommer förbi overfetching problemet då ingen irrelevant information är med i svaret från servern. Med testmiljön i GraphQL Playground möjliggörs det för utvecklaren att undersöka olika förfrågor för att bekräfta att rätta data returneras samt att mäta responstiderna för varje förfråga.

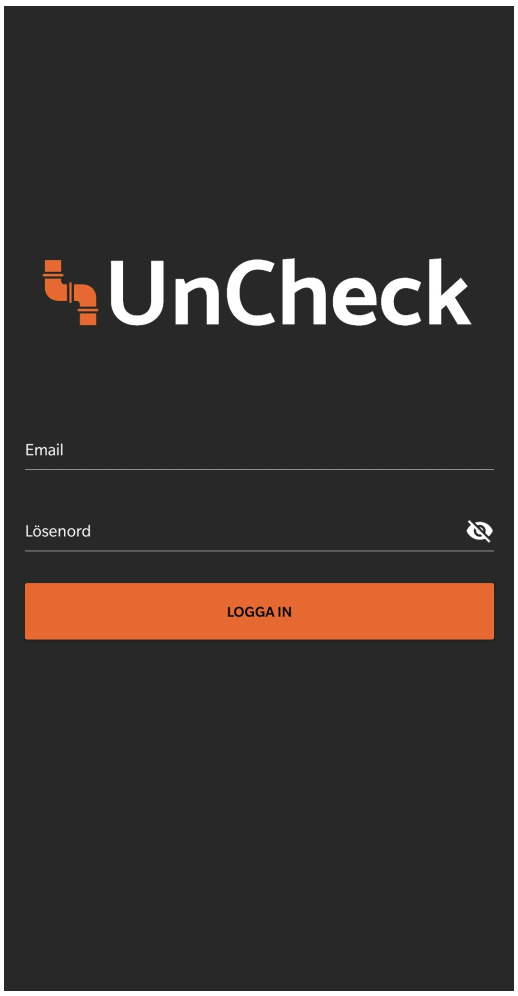
5.3 Applikation

Applikationen följer designstrukturen för Material Design för knappar, fonter och interaktioner. Generellt för hela applikationen gäller ett genomgående tema för att användaren ska kunna känna igen sig oberoende på vilket vy de befinner sig i. Applikationen har en nyans av en svart som grundfärg för att detaljer och information kan lättare identifieras av användare. Informationen är placerad i vita komponenter med accentfärgen orange med anledning till att essentiella funktioner eller information ska kunna hittas till utan att behöva läsa sig till information. Tydliga knappar i nedre delar av vyerna indikerar till att de kan användas för att avsluta nuvarande uppgift eller för att utföra en funktion som tar dig vidare till ett annat steg i processen för utsättning. Applikationen använder sig av Material Fonts i utformat för applikationer som använder sig av Material Design. Beroende på vilket typ av text som ska visas används de designförslag som Google har indikerat i Material Design, specifikt typsnittsstorlek, bokstavsmellanrum och vart versaler ska användas. Navigeringen följer den traditionella metoden för att gå tillbaka genom en bakåtknapp uppe i vänstra hörnet. Logotypen och systemets namn finns i varje vy, bortsätt från kartvyn, se 5.2.4, då kartan fyller hela skärmen. För att fortsätta med att ha en röd tråd genom hela applikationen för att användaren ska känna igen sig används samma logotyp, se figur 5.1, tillsammans med applikationens namn ”UnCheck”. Färgen för logotypen är samma som den tydliga accentfärgen som genomgår alla vyer samtidigt som den speglar vad applikationen behandlar. Eftersom främsta användningsområdet är att sköta utsättningar för grävande vid ledningar användes ett simpelt rör för att referera till de ledningar som visas i ärendena.

Figur 5.2: Splashscreen

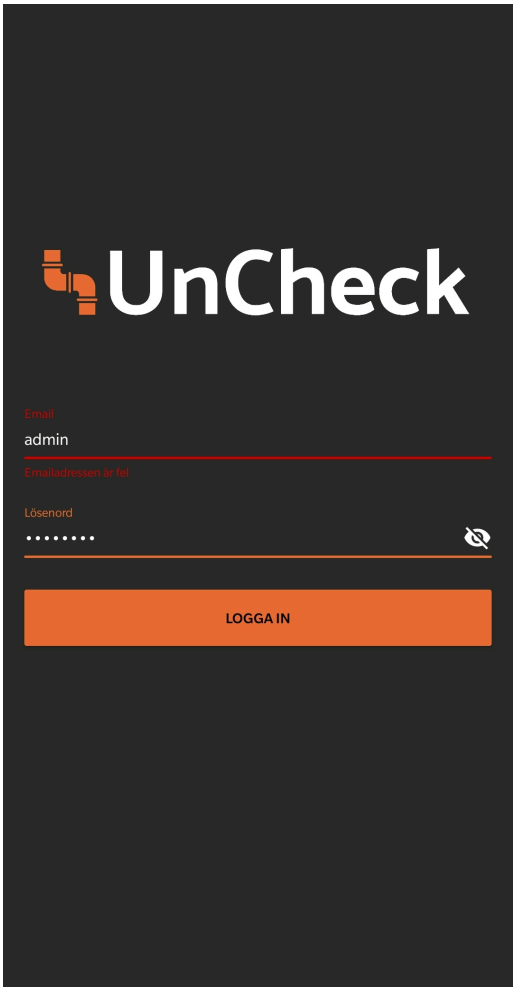
5.3.1 Autentiseringsvy

Användaren möts av en inloggningsskärm efter uppstart av applikationen som är en demonstration av hur autentisering skulle fungera. Den slutgiltiga versionen använde sig endast av fasta inloggningsuppgifter som inte var kopplade till någon databas och utan säker inloggning. Detta berodde främst på tidplanen och målen uppsatta i början av projektet, där autentisering inte var något som vi strävade efter då det varken skulle ge fler fördelar till användare men även med anledning att data som visas i applikationen endast är testdata utan känslig information. Genom att fortfarande demonstrera hur inloggningen skulle se ut skulle det påvisa att olika användare kan ha olika data designerat till sig i applikationen, vilket skulle motsvara verkliga ärenden där utsättare får olika uppgifter och områden sig tilldelade. Startsidan innehåller applikationens logga med samma



Figur 5.3: Autentiseringsvy

färgschema som genomgår hela applikationen. Fälten för inloggningsuppgifter är specifika för mailadress och lösenord vilket betyder att tangentbordet anpassar sig efter den typ av input. Tangentbordet får at-tecken med på första sidan föra att enklare och snabbare kunna skriva in sina uppgifter, samt att lösenordet är dolt när det skrivs in för att öka säkerheten för att andra personer inte ska se vad som skrivs in. En funktion för att visa lösenordet är implementerat, för att användare kan bekräfta att rätt lösenord har skrivits in. För att fullfölja inloggningen finns en tydlig knapp för att logga in som följer samma färgmönster som resterande applikationen. Den följer Material Design i utformningen och att texten är i versaler med tydligt mellanrum mellan bokstäverna (se figur 5.2). Autentiseringen bekräftar att mailadressen och lösenordet stämmer överens med varandra, att de är associerade, och varnar användaren om något av fälten har fyllts i inkorrekt. Även den funktionen följer mönstret uppsatt i Material Design med att tydligt demonstrera vilket av fälten som är felaktigt med att markera det med röd färg (se figur 5.3).



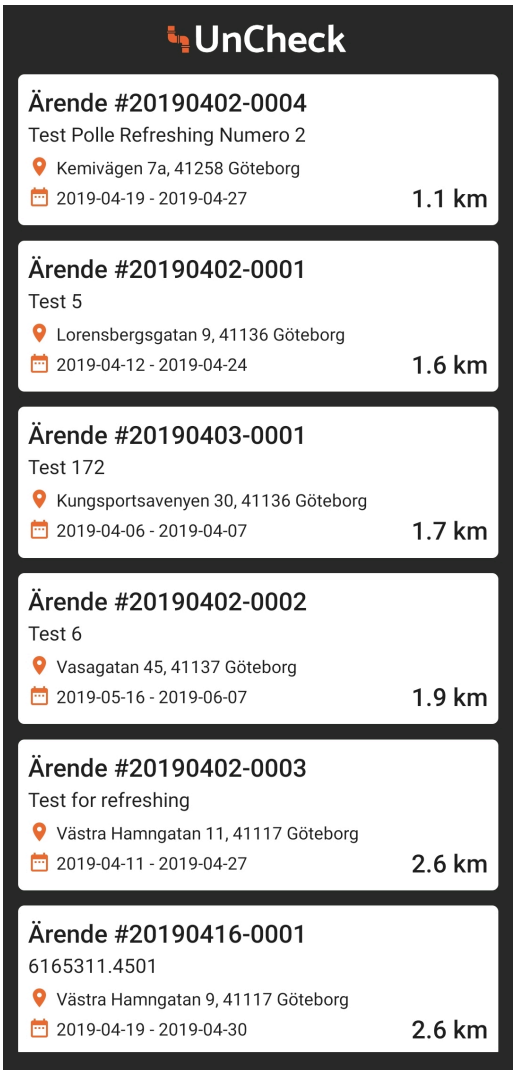
Figur 5.4: Felmeddelande i autentiseringsvy

5.3.2 Listvy

Nästa vy efter inloggningen är alla aktiva ärenden som en användare har, ordnat efter avståndet till området för utsättning baserat på positionen för enheten. Listan visas efter avstånd för att effektivisera arbetet hos en utsättare där de kan basera utförandet av uppdrag på vad som är närmast dem. Utsättningarna kan innehålla flera områden för en och samma utsättning vilket gör att avståndet från användaren baseras på närmsta område inom ärendet.

Listan är baserad på kort, så kallade 'cards' i Material Design, som skapar en enklare uppfattning om olika utsättningar. Det skapar en tydligare överblick för användare där uppdrag enklare kan urskilja dem. Informationen i korten kan struktureras annorlunda från en traditionell lista där uppgiftens identifiering agerar som rubrik med information som följer under (se figur 5.4). Underrubriken innehåller namn på ärendet som har skapats av utföraren av utsättningen. Informationen kan anordnas vertikalt med ikoner för att ytterligare förtydliga vilken typ av information korten innehåller. Ikonerna är specifika för Material Design och utvecklade för att användare ska känna igen dem från andra applikationer eller system. Närmsta adress för området är beskrivet nedanför underrubriken med en GPS-ikon som är generellt gemensamt för alla olika existerande enheter. Adressen som är angivet som närmsta är specificerat av skaparen av ärendet för att mer effektivt kunna identifiera var platsen ligger, kontra om det hade varit angivet i enbart koordinater. Utsättningen har specifika datum när arbetet förväntas vara avklarat vilket visas med en kalender-ikon och ett spann på när arbetet kan tidigast och senast utföras. Vyn har en gemensam standard för avstånd mellan olika komponenter och mot yttre gränser för applikationen. Detta är för att det ska vara behagligt att utläsa information och kunna associera vilken information tillhör vilken komponent. Alla artiklar i listan är interagerbara för att

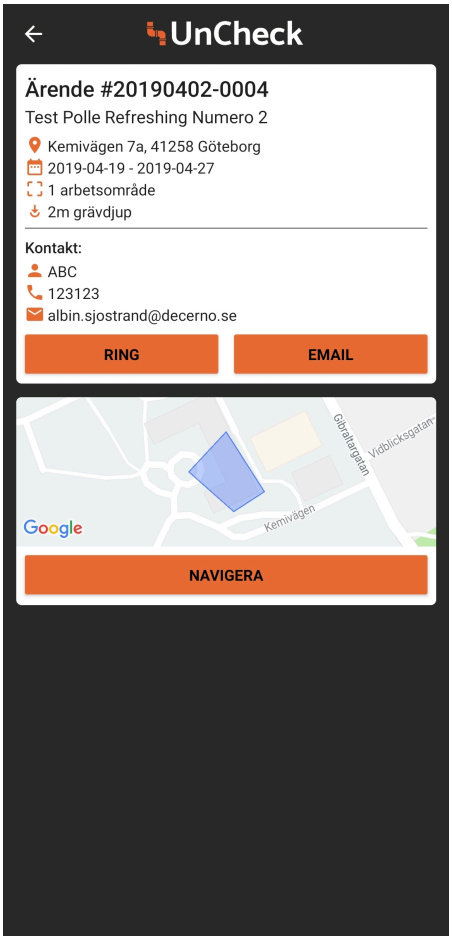
navigera vidare till nästa vy där mer detaljerad information och uppgifterna beskrivs. Genom att välja en av utsättningarna visualiseras trycket med en så kallad 'Ripple Effekt', från hur Material Design är designat för att visa användaren att en interaktion har gjorts. Informationen i listan är sparade för offline användning där användaren fortfarande kan komma åt utsättningar oberoende på om enheten har uppkoppling, dålig mottagning eller ingen uppkoppling alls, till för avlägsna områden.



Figur 5.5: Listvy

5.3.3 Detaljerad vy

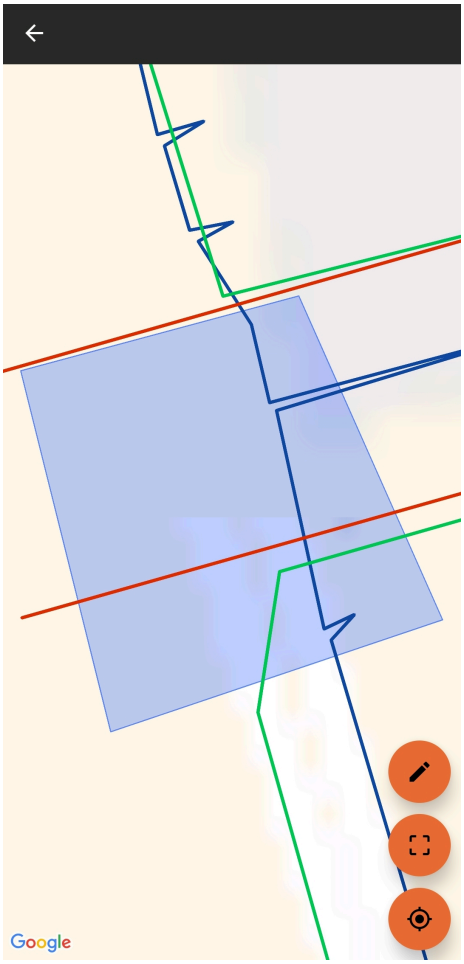
Den detaljerade vyn i applikationen är en utvecklad version av den tidigare listvyn där mer information om utsättningar kan utläsas. Fortsättningsvis följer samma tema för designen där färger och kort används. Korten är utformade på liknande sätt som listvyn där rubriken, underrubriken, adressen och datumet är densamma. Mer information kring kontaktuppgifter finns att tillgå som tillhör samma kort som informationen nämnd ovan genom att vara inom samma behållare. Den nya detaljerade informationen är skild från resterande genom en linje med en mindre text som beskriver att informationen nedanför är kontaktuppgifter till den person som är satt som kontaktperson. Kontaktpersonens namn, telefonnummer och mailadress är beskrivet med ikoner som motsvarar vilken typ av information det är. Silhuett för namnet, en telefon för numret och en email-symbol för mailadressen. Inom området för kontaktuppgifter finns knappar för att enkelt kunna kontakta personen i fråga. De utgår från samma stil som resterande knappar i applikationen med tema och typsnitt. Funktionerna för dessa knappar är även specifika för den typ av enhet som användaren har, om det är Android eller iOS, så att användaren navigeras till enhetsspecifika applikationer för samtal och email. De fyller i automatiskt telefonnumret eller mailadressen när knapparna trycks på för att effektivisera processen för att kontakta ansvariga personer (se figur 5.5). I samma vy finns ett kort med snapshot av området i en kartvy. Den visar utsatta koordinater, ett färglagt område för utsättningsplats och ledningars position inom områden. Kartan är kan interageras med genom om den trycks på där användaren tas till en fullskärm av kartan vilket beskrivs ytterligare under 5.2.5. I kortet för kartan finns en knapp för att kunna navigera till området med en utomstående navigeringsapplikation. Funktionen för navigering är utformad för olika enheter där adressen för iOS-enheter öppnas i Apple Maps och för Android i Google Maps. För Android startar navigeringen automatiskt när knappen för navigering interageras med och för iOS fylls adressen i och kräver att användaren själv startar navigeringen. Oavsett enhet är startpunkten baserat på nuvarande position för användarens enhet.



Figur 5.6: Detaljerad vy

5.3.4 Kartvy

Kartvyn innehåller en fullskalig karta baserat på React Native Maps som utgår från Google Maps. Den innehåller två knappar, så kallade 'Floating Action Button' som ligger på ytan av kartan. De är placerade lättåtkomliga i det nedre högra hörnet, efter Material Designs utformning för den här typen av knappar, vilket gör dem lättåtkomliga för användaren både en- och tvåhändig fattning av enheten. Den första knappens funktion är att lokalisera sin egen position utifrån enhetens GPS och den andra är för att se var områdena ligger. De både flyttar kartan till de koordinater som motsvarar funktionerna. För ärenden som har fler en ett område visas en central översikt där alla områden passar inom skärmens ramar. Kartan visar information om området genom att det är färgat mellan de angivna koordinaterna för området. Här visas ledningarna för ärendet genom linjer utefter dess placering. För att kunna skilja på vilka ledningarna som är vilka är har de olika färgkoderna, mörkblått för vattenledningar, grönt för säkra fiberledningar och rött för osäker fiber. Eftersom datalagringen av ledningars position är ungefärliga med en viss felmarginal ger kartvyn endast en indikation på vart i området ledningarna ligger. Ledningarna är utmärkta även utanför det designerade utsättningsområdet för att användare ska ha möjlighet till att identifiera från vilken riktning de kommer och går ifrån. Användaren kan endast se en viss area kring området vart ledningar går för att behålla säkerheten till att de inte ska kunna se anslutningar som inte hör till utsättningen (se figur 5.6).



Figur 5.7: Kartvy

6 Slutsats

Slutsatsen beskriver hur GraphQL kan användas för att implementeras i befintliga och nya system och hur React Native kan utnyttjas för att enkelt skapa applikationer med en gemensam kodbas för både iOS och Android. Avsnittet sammanfattar lärdomarna kring Storybook för hur det förenklar arbetsgången för nya projekt samt effektivisering av processen för Ledningskollen.

6.1 GraphQL

Då GraphQL i sig är bara en specifikation och kan förekomma som olika implementationer, som i sig innebär att de kommer ha olika nackdelar och fördelar bakom sig. Med det sagt om den implementationen håller sig till vad specifikationen säger så kommer den lösa de nackdelarna som förekommer med REST API och ha ett positivt inflytande på applikationens prestanda. I projektet ifråga så användes implementationen Apollo, som inte bara löste nackdelarna med ett REST API men gjorde utvecklandet av klienten väldigt bekvämt då den abstraherar bort många svårigheter bakom funktionalitet som har med datahantering att göra.

GraphQL har blivit implementerad i många populära programmeringsspråk och det är möjligt och relativt enkelt att implementera det som ett extra lager i ett existerande system så är GraphQL i våra ögon en fantastisk investering för förbättring av upplevelsen för både användare i form av snabbare prestanda och utvecklare med hjälp av ett bekvämare utvecklingsprocess. Det är enligt oss givet att GraphQL är något vi kommer se mycket av i snar framtid och bör i längden ersätta REST APIer helt. Det ökar prestandan för förfrågningar mot databaser och APIer samt minskar problemen som REST har, med overfetching och lång latens, vilket resulterar i att målet med att undersöka fördelarna med GraphQL jämfört med REST API är uppfyllt. Det hela resulterar i att kommunikationen är effektivare med snabbare kommunikation och att information är specifika till de förfrågningar som görs för sammankopplingen av databaser och APIer som applikationen kräver för att fungera och visa information som är relevant för användaren.

6.2 React Native

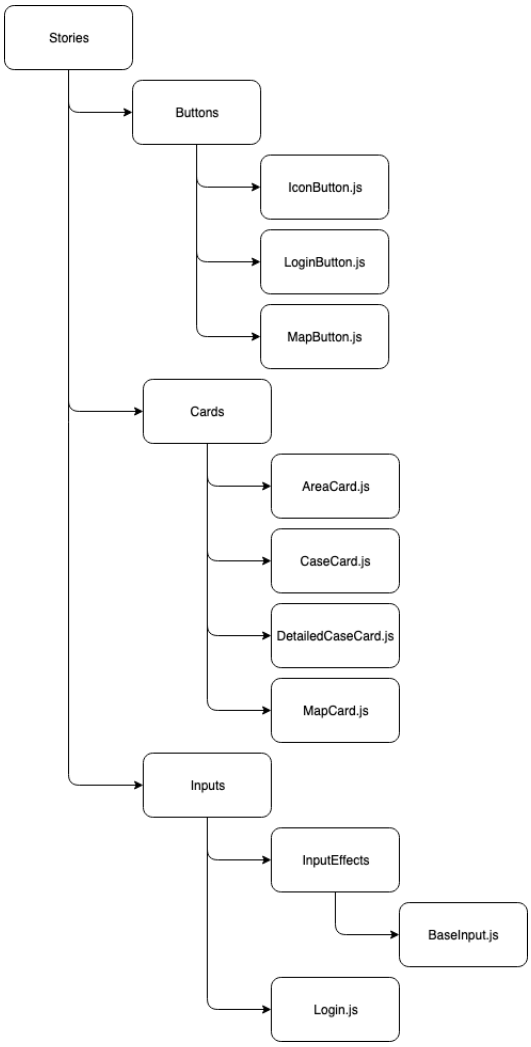
Eftersom React Native erbjuder att utvecklare kan skapa en gemensam kodbas för både enheter på operativsystemen iOS och Android försvinner dilemmat att prioritera till vilket OS utveckling och funktioner ska till. Tid och kraft kan istället läggas på att ta fram lösningar till problem som gäller generellt för ett helt system eller en applikation. Funktionerna som skapades i projektet kunde appliceras utan att behöva specifikt skapas för särskilda enheter. Det essentiella med att kunna utnyttja samma kod var att kunna demonstrera hur applikationen fungerade, men målet för användare i fokus. Med React Native effektiviseras utvecklingen att enhetsspecifik kod inte längre är ett krav, att för iOS-applikationer ska skrivas i Swift och att för applikationer i Android behövs skrivas i Java, utan endast den gemensamma koden i JavaScript. I fåtal fall krävs det att det definieras till vilken typ av enhet en funktion ska vara till då det skiljer i hur operativsystem hanterar öppnande av utomstående applikationer, som exempelvis navigation, email och telefonsamtal. Då React Native är utvecklat med att kunde appliceras på olika OS krävs det endast småjusteringar i utförandet av kallandet på funktioner.

Genom användning av React Native i projektet förs projektet framåt med ett enkelt språk att lära sig med tidigare vanor inom programmering. Upplärningstiden är snabb för både erfarna programmerare och de utan speciellt stor bakgrund i applikationsutveckling då det finns stort utbud av lösningar från andra utvecklare. Problem eller felkoder har i många fall redan uppmärksammats av andra personer och har publicerat publikt.

Målet om att framställa en applikation där utsättaren kan se sin egen position, arbetsområden och ledningsägare ledningar är där med uppfyllt. Applikationen kan visa informationen för att underlätta och effektivisera arbetet vid utsättningar. Även målet med att applikationen ska använda sig av ramverket React Native är uppfyllt, vilket demonstrerar fördelarna med att använda sig av en gemensam kodbas till iOS och Android.

6.3 Storybook

Storybook gjorde det enklare för oerfarna utvecklare att sätta sig in i utformningen av applikationen och medförde att komponenterna som skapades under projektets gång är modulära och kan användas om fortsatt utvecklingen av applikationen skulle ske, samt användbara för andra utvecklingsprojekt i React Native. Det gav även en tydligare mappstrukturering för vilka komponenter tillhörde vilka vyer (Se figur 6.1).



Figur 6.1: Mappstruktur för Storybook

6.4 Ledningskollen

Arbetsflödet för hur utsättningar skedde digitaliserades från personlig kontakt med ansvariga personer till att ett gemensamt system som hade samma funktionalitet. Det skapar en effektivare process där användare inte är beroende av varandra för att utföra sina arbetsuppgifter. Data från skapade ärenden i Ledningskollen kopplades ihop med information om ledningar och kartor. Applikationen agerade som en brygga där användare får en bättre översiktlig blick om den informationen som finns att tillgå. Från ett systemutvecklingsperspektiv ökar verkningsgraden genom en mer effektiv GraphQL-server som hanterar flera källor av information i fallet av Ledningskollens APIer och databaser med högre prestanda än de tidigare använda REST APIer. Kapaciteten för systemet blir högre trots hanteringen av multipla informationskällor.

Behovet av hur mycket av den tillgängliga informationen från ärenden och ledningsdata skiljer sig mellan användare och användningsfall. Bedömningen av vilken information ska visas i applikationen och hur har gjorts i samråd med Ledningskollens användarstöd baserat på hur de allra flesta använder systemet och data.

Vår handledare Henrik Karlsson på Decerno, som varit projektledare för Ledningskollen i tio år bedömer att ett införande av en applikation liknande den som utvecklats skulle kunna bidra till minskade skador. Applikationen som tagits fram som prototyp och kopplats till Ledningskollen förtydligar, förenklar och kvalitetssäkrar processen kring utsättningar på ett utmärkt sätt, vilket uppfyller syftet om en bättre arbetsprocess.

7 Diskussion

Kapitlet om diskussion handlar om hur Apollo har utvecklats för lättare implementering av GraphQL, alternativa metoder till React Native för utvecklingen av applikationer, användning av Storybook för att effektivisera utveckling av komponenter i React Native samt vad applikationen har för påverkan i samhället.

7.1 React Native

I tidigare stadie av projektet analyserades om Bowser skulle användas som klient för att påskynda applikationens utveckling med struktureringen och förinstallerade paket som ingår i Bowser. Detta var dock något som inte applicerades på grund av att energi och fokus skulle vara tvunget att läggas på att förstå hur upplägget för Bowser såg ut, till skillnad från att skapa en egen grund för applikationen. Det hade även inneburet att anpassa funktionaliteter efter vad som är möjligt i en färdig boilerplate och kunna ha förståelse för vilka delar i strukturen är beroende av varandra, för att inte i ett senare stadie stöta på problem. Ur Bowser användes strukturen för mappstrukturen med namngivning och vad för typer av filer skulle placeras för att underlätta för både utvecklare och utomstående personer att förstå hur beståndsdelarna var relaterade till varandra.

Det finns andra metoder jämfört att ta fram gemensamma kodbaser än React Native som exempelvis Flutter. Det var något som upptäcktes och diskuterades efter att utvecklingen i projektet påbörjats, om det fanns andra möjligheter att använda sig Flutter. Principen är densamma oavsett vilket ramverk som används, att för mindre utvecklingsteam krävs det att mindre tid behöver läggas på programmering av olika operativsystem. Skillnaden som kan utläsas mellan dessa system är att React Native har varit etablerat på applikationsmarknaden längre tid än Flutter vilket har resulterat i att stödet för lösningar skapta av andra användare är större. Flutter har däremot ett stort stöd av Google bakom sig vilket har gjort att processen för uppstart av Androidapplikationer påskyndats. För större företag med fler personer i varje team finns istället möjligheten att ha separat utveckling till olika operativsystem. I dessa fall finns fördelarna i att applikationer utvecklas specifikt till enheten och att lösningar kan tas fram som kan prestera bättre då de inte behöver lösa problem som kräver olika lösningar beroende på system.

7.2 Storybook

Storybook är inte enbart framtaget för React Native och kan även appliceras på andra projekt. Det är framförallt något som kan ses som positivt där utvecklare är oerfarna eller inte har kunskap i programmering för applikationen i sig. Eftersom komponenter skapas och testas i en separat miljö kan de fokusera på att endast få dessa delar att agera efter önskat beteende. I den andra aspekteten att inte använda sig av Storybook alls är beroende på kunskapen inom ett team. Om personer redan besitter kunskap om React Native blir det istället ett mellansteg som kräver resurser som istället kan läggas på en direkt implementering. Nackdelen i detta fall skulle vara att problematiska beteenden inte kan härledas specifikt till en komponent utan kan även bero på andra beroenden inom applikationen som med användnings av Storybook skulle kunna elimineras. Projekt bör utgå från det spektrum vart prioriteringarna ligger från att kunna härleda vart problemen kommer ifrån till att påskynda utvecklingen framåt.

7.3 Applikationens påverkan på samhället

Varje dag skadas ledningar vid markarbeten med stora konsekvenser för privatpersoner, företag och samhälle. Många av dessa skador uppstår trots att alla inblandade egentligen gjort rätt enligt dagens processer vilket kan lättare undvikas med hjälp av applikationen som redan innan markarbeten utförs kan indikera vart ledningarna går. Onödigt material som behövs för att ersätta skadade ledningar minskas vilket ur både miljö- och kostnadsperspektiv är positivt där ersättningsmaterial inte behöver tillverkas. Vanligtvis använder sig arbetsprocessen av papperskopior vid utsättningarna som behövs skrivas ut för varje ärende, vilket elimineras genom att applikationen visar kartor och detaljer om ledningar och området digitalt.

Referenser

[1] *A modern JavaScript utility library delivering modularity, performance extras — Lodash*. [Online; hämtat 2019-05-07]. 2019. URL: https://lodash.com/.

[2] Atlassian. *Getting Starded With Trello — Trello*. [Online; hämtat 2019-05-07]. 2019. URL: https://trello.com/en/guide.

[3] E. Contributors. *4.x API — Express.js*. [Online; hämtat 2019-05-07]. 2019. URL: http://expressjs.com/en/4x/api.html.

[4] *Create intuitive and beautiful products with Material Design.. — material*. [Online; hämtat 2019-05-07]. 2019. URL: https://material.io/design/.

[5] *Documentation — Apollo Docs*. [Online; hämtat 2019-05-07]. 2019. URL: https://www.apollographql.com/docs/.

[6] *Documentation — DBeaver Community*. [Online; hämtat 2019-05-07]. 2019. URL: https://dbeaver.io/docs/.

[7] N. Foundation. *About Docs — Node.js Docs*. [Online; hämtat 2019-05-07]. 2019. URL: https://nodejs.org/en/docs/.

[8] *Getting Starded — React Native*. [Online; hämtat 2019-05-07]. 2019. URL: https://facebook.github.io/react-native/docs/getting-started.

[9] *Getting Started — Visual Studio Code*. [Online; hämtat 2019-05-07]. 2019. URL: https://code.visualstudio.com/docs.

[10] *GraphQL Playground - Visually exploring an Apollo Server — Apollo Docs*. [Online; hämtat 2019-05-07]. 2019. URL: https://www.apollographql.com/docs/.

[11] *Growing library to provide some basic geo functions — Geolib*. [Online; hämtat 2019-05-07]. 2019. URL: https://github.com/manuelbieh/Geolib.

[12] *How devevelopers work — GitHub*. [Online; hämtat 2019-05-07]. 2019. URL: https://github.com/features.

[13] *Ignite CLI - The hottest CLI for React Native — Infinite Red*. [Online; hämtat 2019-05-07]. 2019. URL: https://github.com/infinitered/ignite.

[14] *Introdcution — Storybook*. [Online; hämtat 2019-05-07]. 2019. URL: https://storybook.js.org/docs/basics/introduction/.

[15] *Introduction to GraphQL — GraphQL*. [Online; hämtat 2019-05-07]. 2019. URL: https://graphql.org/learn/.

[16] *Master the power of InVision Studio — InVisionApp*. [Online; hämtat 2019-05-07]. 2019. URL: https://www.invisionapp.com/studio/learn.

[17] *Meet Android Studio — Android Developer*. [Online; hämtat 2019-05-07]. 2019. URL: https://developer.android.com/studio/intro/.

[18] *npm Documentation — npmjs*. [Online; hämtat 2019-05-07]. 2019. URL: https://docs.npmjs.com/.

[19] *React Native Map components for iOS + Android — React Native Community*. [Online; hämtat 2019-05-07]. 2019. URL: https://github.com/react-native-community/react-native-maps.

[20] K. Schwaber och J. Sutherland. *The Scrum Guide*. [Online; hämtat 2019-05-07]. 2017. URL: https://www.scrumguides.org/scrum-guide.html.

[21] M. Team. *Coordinate Systems Worldwide — espg.io*. [Online; hämtat 2019-05-07]. 2018. URL: https://epsg.io/.

[22] *Transform coordinates from one coordinate system to another. — coords*. [Online; hämtat 2019-05-07]. 2019. URL: https://github.com/derhuerst/transform-coordinates.

[23] *What is Prettier? — Prettier*. [Online; hämtat 2019-05-07]. 2019. URL: https://prettier.io/docs/en/index.html.

[24] *Xcode 10 — Apple Developer*. [Online; hämtat 2019-05-07]. 2019. URL: https://developer.apple.com/xcode/.