

# Ontwerp van een smartphoneapplicatie voor reizigersbegeleiding op een perron.

**Stijn DE BELS**

Promotor(en): An Van Haperen

Co-promotor(en): Jonas Van den Bergh  
(NMBS)

Masterproef ingediend tot het behalen van  
de graad van master of Science in de  
industriële wetenschappen: Elektronica - ICT  
afstudeerrichting ICT

Academiejaar 2019 - 2020

©Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Technologiecampus De Nayer, Jan De Nayerlaan 5, B-2860 Sint-Katelijne-Waver, +32 15 31 69 44 of via e-mail [iiw.denayer@kuleuven.be](mailto:iiw.denayer@kuleuven.be).

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

# Voorwoord

De masterproef die u voor u heeft liggen heeft als onderwerp 'Ontwerp van een smartphoneapplicatie voor reizigersbegeleiding op een perron.' en is geschreven in functie van het behalen van het masterdiploma Industrieel Ingenieur Elektronica-ICT. Bij deze zou ik graag enkele mensen willen bedanken die me hebben geholpen bij het tot stand komen van deze masterproef. Om te beginnen bedank ik graag mijn promotores An Van Haperen en Jonas Van den Bergh die mij geholpen hebben op alle mogelijke manieren, ondanks de fysieke afstand die gedurende het gehele jaar tussen ons was. Ik waardeer de inspanningen die zij gedaan hebben voor een vlotte communicatie te verkrijgen enorm, om nog niet over alle tips, verbeteringen en ideeën die zij gegeven hebben te beginnen.

Verder zou ik graag volgende mensen bedanken: mijn ouders, broer en zus om mijn masterproeftekst na te lezen, ondanks hun gematigde interesse in de ontwikkeling van smartphone-applicaties en al mijn vrienden en familieleden die mij gesteund en geholpen hebben. Ook speciale dank aan Google Earth, om mij enkele broodnodige geografische coördinaten te geven en mijn tuin, om perron te willen spelen.

Als laatste wil ik de NMBS bedanken om mij toe te laten om in juli bij hun te komen werken, hun wensen en ideeën over het eindresultaat te geven en ook de nodige data beschikbaar te stellen die nodig is voor het project.

Tot slot wens ik u veel leesplezier!

Stijn De Bels, 24 mei 2020

# Abstract

In het tijdperk van de smartphone zoeken bedrijven voortdurend naar middelen om de gebruiksvriendelijkheid van hun diensten te verbeteren. Zo ook de NMBS, de nationale spoorwegmaatschappij van België. Vanuit deze instelling is het onderwerp van deze masterproef dan ook ontstaan: onderzoek de mogelijkheden om navigatiebepaling op een perron te implementeren in een mobiele applicatie. Hiervoor werd een mobiele applicatie ontwikkeld die aan de hand van de huidige locatie van de gebruiker een navigatiepad weergeeft naar een rijtuig van een trein dat door de gebruiker werd geselecteerd. De data die nodig is om deze navigatie te realiseren, zoals informatie over de treinen en de perrons, wordt opgehaald uit een MySQL databank en verstuurd via PHP bestanden.

Het onderzoek begon met een literatuurstudie waarin de mogelijkheden inzake mobiele applicaties werd onderzocht. Na deze analyse werd het hiervoor geschikte framework onderzocht en werd een keuze in de ecosystemen die we willen bedienen gemaakt.

Na onze literatuurstudie werd een conceptstudie uitgewerkt. Hierin werden verschillende prototypes ontwikkeld, met het oog op het KISS principe. Dit omdat we met onze applicatie alle mogelijke gebruikers wilden bereiken. Vervolgens werd onze startapplicatie in het gekozen framework (React Native) gecreëerd. Deze applicatie leerde ons onder andere de connectie tussen de databank en de applicatie op te stellen.

Het resultaat is een mobiele applicatie waarin navigatiebepaling is uitgewerkt. Deze werd getest in een bepaald treinstation. Dit is een volledig functionele applicatie voor iOS en Android. De gebruiker geeft een eindbestemming in en selecteert een trein en rijtuig. Op basis hiervan wordt het kortste pad gezocht naar dit rijtuig vanaf de huidige locatie. Tenslotte werden hierop enkele testen gedaan die interessante resultaten gaven inzake nauwkeurigheid van de huidige locatie.

**Keywords:** Locatiebepaling, React Native, mobiele applicatie, MySQL, gebruikerservaring

# Abstract

In the era of the smartphone, companies are constantly looking for ways to offer an improved user experience for their services, and Belgium's national railway service (SNCB) is no exception. With this understanding, the master thesis focused on the research and the viable implementation of navigational services into a mobile application.

The mobile application was developed to create a navigation path based on the current location of the user to a carriage of a train selected by them. Necessary data about the Belgian train systems and platforms were put into a MySQL database, and the communication in between is done using PHP files.

The research started with a literature study in which the possibilities for developing mobile applications was investigated. Following this, research for the appropriate framework began and choices in the IT ecosystems we intended to serve were being decided.

After our literature study, a draft study was created. In this study various prototypes were developed, with the KISS principle in mind, as to ensure simplicity and ease of use to the widest number of users possible. Following this, a draft application which served as our basis for the final application in the chosen framework (React Native) was implemented. This trial taught us how to establish the connection between the database and the application.

Lastly, the mobile application, in which the navigation service had been implemented, was developed and was tested in a specific station. We created a fully functional application for both iOS and Android. After typing in their final destination, the user could select a train and carriage. Based on this, the shortest path to the carriage was suggested based on their current location. Afterwards, some tests were done in which interesting results regarding accuracy of the current location were obtained.

**Keywords:** Location services, React Native, mobile application, MySQL, user experience

# Short summary

## Context and goal of this study

We live in the era of the smartphone. Companies are constantly looking for ways to offer an improved user experience for their services, and Belgium's national railway service (SNCB) is no exception. In cooperation with the SNCB, this thesis was made and tries answering the following questions:

- Is it possible to guide a traveller to the right place on the platform using the smartphones current location?
- Is it possible to determine the location of a train and carriage on the platform in advance?
- Is the accuracy of the user's current location good enough for the app to work?
- Does the database contain enough information about the exact composition of the train to represent the different categories of carriages?

## Literature study

In the information technology industry (IT) there are three largely dominant **IT ecosystems**: Apple, Google, and Microsoft. An ecosystem is a collection of software and hardware that evolves in the same environment. For this application, we focused on targeting the Apple and Android ecosystems.

There are a lot of possibilities for developing mobile applications. They can be developed as a **native applications**, which are applications specifically made for an operating system and offers the best performance, though the code is not reusable. Other possibilities are **web-based applications** or **Progressive Web Apps**. Those apps work (partly) online, , and therefore do not use or require a lot of memory. Working (partly) online also has disadvantages, namely that the smart phone requires data (and possibly the user interface) from the server in order to function. This takes time and slows down the performance. For these reasons, **hybrid** and **cross-path** applications were preferred. With Hybrid applications we render pages within webviews, which are webpages. The main advantage is that their accessibility to native API's, which allows more

features to be implemented. Because the performance is still rather slow, we decided to implement a cross-path application. This combines performance, offline user interface and code reusability.

After choosing to pursue a cross-path application, the different frameworks were studied, and the varying features of **React Native**, **Xamarin** and **Dart** architectures were explored. As we preferred a stable framework with a big community, React Native was selected.

## Draft study

Before we began writing the final application we wanted to ensure that we had a grounded idea of what and how we were going to make the application. These ideas had to meet the expectations of the SNCB. For this reason we made three prototypes. We also developed a first application to get familiar with the React Native framework and test the connectivity between an online MySQL database and the application. This had to be an application in which the user provided their destination, and chose their preferred time and other variable options. The application then gives them the best train carriage, based on these preferences and the occupancy rate.

## Specifications

In this chapter, the global aspects of the final application are discussed. Our application has a server-side which contains the data and a client-side with all of the screens in it. The client-side is available offline on the smartphone, but in order for the application to work, it has to be connected to the internet to receive data from the server. Working with components is typical for React Native. Every screen is a component which contains other components (for example a <Text> component to display text). The main components of our client-side application are the following:

- App.JS
  - HomeScreen:
    - \* DestinationScreen
    - \* ChoiceScreen
    - \* NavigatieScreen:
      - Train: Custom component
      - meldingVerkeerdePerron: modal screen
    - \* EndScreen
  - InfoScreen

## Implementation

Now we've worked out the specifications, every interesting component of the application will be described. Though prior to this, the **server-side** of the application has to be described. In the **MySQL database** we have the following tables:

- **TracksSKW:** contains coordinates of the tracks of the Sint-Katelijne-Waver train station
- **ObjectsSKW:** contains coordinates of all the objects (stairs, ...) of the Sint-Katelijne-Waver train station
- **objectTypes:** contains lengths of objects
- **Dacs:** contains the list of trains to a certain destination
- **CommercialTrainCompositions:** contains information about the composition of the trains
- **PTCars:** connects station ID with station name

Connection with the database happens with the use of **PHP files**:

- **DBConfig.php:** contains hostname, databasename, hostuser and hostpassword
- **get\_trains\_info.php:** to get specific data out of DAC and PTCars table
- **get\_train\_info.php:** to get specific data out of ObjectTypes, ObjectsSKW, TracksSKW and CommercialTrainCompositions table.

Regarding the **client-side**, the input of the destination happens in the **HomeScreen**. After typing in this destination, a connection with the database is made. The application will find all possible trains which stop at the intended destination. These possibilities are displayed in the **ChoiceScreen**, where the user can select their preferred train. After this the NavigationScreen will be entered.

In the **NavigationScreen** it's important that we render all visual elements like the platform, objects on this platform and traincomponents. This is done by converting geographical coordinates to metres and/or metres to pixels. Then some conditional statements are made to assure the user is on the correct platform before the navigation is calculated. Equations are made to guarantee the current location is correctly displayed on the platform. The user can select their preferred carriage in two ways: either by selecting the visual image of the desired carriage, or by selecting a preference with the picker. The picker is a small submenu that passes the value of the preference to the train component.

The **train component** gets data from the navigation screen, it then renders an image for every carriage based on its type, length and options. When a preference is selected in the picker, the best carriage is calculated. This is done as follows: first the distance between the current location

and every carriage is calculated. Then these objects are sorted by this distance, followed by eliminating those carriages that do not conform to the specified preferences. The carriage with the shortest distance is determined to be the best option of the user, and the **navigation path** is created. There are two ways to do this: using geographical coordinates or pixels. The latter process was chosen as it requires less computation. To extend this, objects are implemented on the platform, and it is ensured that the navigation adapts to the objects. When the user has arrived at the destination the **EndScreen** is displayed. Here it is stated that the user reached the correct destination. **Animations** are implemented to make the change of location smoother, also **modal screens** for extra information and notification are implemented.

## Results

In the topic results, the **visual result** and a **test** discussion can be found. Real world tests were conducted at Mechelen Train Station, and we conclude that the accuracy in some train stations should be improved.

## Conclusion

In this thesis we tried answering the following questions:

- Is it possible to guide the traveller to the right place on the platform using the smartphones current location?
- Is it possible to determine the location of a train set and carriage on the platform in advance?
- Is the accuracy of the user's current location good enough for the app to work?
- Does the database contain enough information about the exact composition of the train to represent the different categories of carriages?

We can conclude that most questions received a positive answer. The second question works on the condition that we have access to the needed data. The third question is negatively answered. This is due to the findings of the real world tests conducted in Mechelen, where we concluded an improvement of location services may be needed in some stations.

**Further development and testing** must be undertaken before the application could assure successful public use and adoption. Specifically, the following aspects require future work: security, code efficiency, extensibility of the application and the use of real time data.

# Inhoudsopgave

<b>Voorwoord</b>	iii
<b>Abstract</b>	iv
<b>Short summary</b>	ix
<b>Inhoud</b>	xiv
<b>Lijst met afkortingen</b>	xv
<b>Figurenlijst</b>	xviii
<b>Tabellenlijst</b>	xix
<b>1 Situering en doelstelling</b>	1
1.1 Situering binnen het geheel . . . . .	1
1.2 Situering binnen het bedrijf . . . . .	1
1.3 Onderzoeksprobleem . . . . .	1
1.4 Onderzoekstelling en Doelstelling . . . . .	2
<b>2 Literatuurstudie</b>	3
2.1 De verschillende IT ecosystemen . . . . .	3
2.1.0.1 Het Apple ecosysteem . . . . .	3
2.1.0.2 Het Google ecosysteem . . . . .	3
2.1.0.3 Het Microsoft ecosysteem . . . . .	4
2.2 Mogelijkheden in het maken van een mobiele applicatie . . . . .	4
2.2.1 Native Applicaties . . . . .	4
2.2.1.1 Voordelen . . . . .	4

2.2.1.2 Nadelen . . . . .	5
2.2.2 Webgebaseerde applicaties . . . . .	5
2.2.2.1 Architectuur . . . . .	6
2.2.2.2 Voordelen . . . . .	6
2.2.2.3 Nadelen . . . . .	6
2.2.3 Progressive Web Apps . . . . .	7
2.2.3.1 Architectuur . . . . .	7
2.2.3.2 Voordelen . . . . .	8
2.2.3.3 Nadelen . . . . .	8
2.2.4 Hybride Applicaties . . . . .	8
2.2.4.1 Architectuur . . . . .	9
2.2.4.2 Voordelen . . . . .	9
2.2.4.3 Nadelen . . . . .	10
2.2.5 Cross-path Apps . . . . .	11
2.2.5.1 Voordelen . . . . .	11
2.2.5.2 Nadelen . . . . .	11
2.2.6 Conclusie . . . . .	11
2.3 Relevante technologien . . . . .	14
2.3.1 Xamarin . . . . .	14
2.3.1.1 Architectuur . . . . .	14
2.3.2 React Native . . . . .	15
2.3.2.1 React . . . . .	15
2.3.2.2 Virtual DOM . . . . .	15
2.3.2.3 Bridge . . . . .	16
2.3.3 Flutter . . . . .	16
2.3.3.1 Architectuur . . . . .	17
2.3.3.2 Dart . . . . .	17
2.3.4 Verantwoording keuze . . . . .	17
<b>3 Conceptstudie</b>	<b>18</b>
3.1 Prototype 1 . . . . .	18
3.2 Prototype 2 . . . . .	20
3.3 Prototype 3 . . . . .	21

3.4 Testapplicatie . . . . .	23
3.4.1 Systeemarchitectuur van de testapplicatie . . . . .	23
3.5 Conclusie . . . . .	26
<b>4 Specificaties</b>	<b>27</b>
4.1 Vereisten van de applicatie . . . . .	27
4.2 Systeemarchitectuur van het finale model . . . . .	28
4.2.1 Globale structuur . . . . .	29
4.2.2 Navigatiestructuur . . . . .	29
4.2.2.1 TabNavigator . . . . .	29
4.2.2.2 StackNavigator . . . . .	30
4.2.2.3 Modal screen . . . . .	30
4.2.3 Custom Components . . . . .	30
4.2.4 Expo platform . . . . .	31
4.2.5 Veelgebruikte variabelen en states . . . . .	31
4.2.6 Vooraf gedefinieerde constanten en veelgebruikte functies . . . . .	32
4.3 Conclusie . . . . .	32
<b>5 Uitwerking</b>	<b>33</b>
5.1 Databank en PHP connectie . . . . .	33
5.1.1 Databank . . . . .	33
5.1.2 PHP . . . . .	36
5.2 Startscherm (App.js) . . . . .	36
5.3 Invoer van bestemming (HomeScreen.js) . . . . .	37
5.4 Keuzelijst van treinen (ChoiceScreen.js) . . . . .	38
5.5 Trein- en navigatiescherm (NavigatieScreen.js) . . . . .	38
5.5.1 Geografisch coördinatenstelsel . . . . .	39
5.5.2 Visualisatie van het perron . . . . .	39
5.5.2.1 Grootteberekening van het perron . . . . .	40
5.5.2.2 Objecten op het perron . . . . .	40
5.5.3 Locatiebepaling en visualisatie . . . . .	43
5.5.3.1 Locatiebepaling op een smartphone . . . . .	43
5.5.3.2 Locatiebepaling in React Native . . . . .	44

5.5.3.3	Randvoorwaarden voor visualisatie van de locatie . . . . .	45
5.5.3.4	Positionering huidige locatie op het perron . . . . .	47
5.5.3.5	Nauwkeurigheid . . . . .	49
5.5.3.6	Implementatie van nauwkeurigheid in de applicatie . . . . .	50
5.5.4	Visualisatie van de trein . . . . .	52
5.5.4.1	Train component (Train.js) . . . . .	52
5.5.4.2	Rendering van de rijtuigen en optiemogelijkheden . . . . .	52
5.5.4.3	Offsetberekening van de trein . . . . .	53
5.5.5	Navigatiebepaling . . . . .	54
5.5.5.1	Eerste optie: berekening met geografische coördinaten . . . . .	55
5.5.5.2	Tweede optie: berekening met pixels . . . . .	58
5.5.5.3	Navigatie bij objecten op het perron . . . . .	59
5.5.6	Overige visuele elementen . . . . .	62
5.5.6.1	Keuzemenu . . . . .	62
5.5.6.2	Meldingsvenster verkeerde perron . . . . .	64
5.5.6.3	Minitatuurweergave . . . . .	65
5.5.6.4	Extra informatiescherm . . . . .	66
5.6	Bevestiging correctie positie (EndScreen.js) . . . . .	67
5.7	Uitbreidung: Animaties . . . . .	67
5.7.1	Navigatiebewegingen vloeiend laten verlopen . . . . .	68
5.7.2	Verschijnen van het perron op het scherm . . . . .	71
5.7.3	Accentueren van het geselecteerde rijtuig . . . . .	72
5.8	Uitbreidbaarheid . . . . .	75
5.9	Conclusie . . . . .	76
<b>6</b>	<b>Resultaten</b> . . . . .	<b>77</b>
6.1	Testing . . . . .	77
6.1.1	Reallife test: Mechelen . . . . .	77
6.1.2	Reallife test: Sint-Katelijne-Waver . . . . .	79
6.1.3	Overige testen . . . . .	80
6.2	Visueel eindresultaat van de applicatie . . . . .	80
6.2.1	HomeScreen . . . . .	80
6.2.2	ChoiceScreen . . . . .	81

6.2.3	NavigationScreen . . . . .	82
6.2.3.1	EndScreen . . . . .	85
6.2.3.2	InfoScreen . . . . .	86
<b>7</b>	<b>Besluit</b>	<b>87</b>
7.1	Future work . . . . .	88
<b>A</b>	<b>Broncode</b>	<b>92</b>
A.1	Javascript bestanden . . . . .	92
A.1.1	constants.js . . . . .	92
A.1.2	functions.js . . . . .	94
A.1.3	Navigatieberekening eerste optie: berekening met geografische coördinaten .	97
A.2	PHP bestanden . . . . .	99
A.2.1	DBConfig.php . . . . .	99
A.2.2	get.trains.info.php . . . . .	100
A.2.3	get.train.info.php . . . . .	101

# Lijst van afkortingen

- **PWA:** Progressive Web App
- **HTML:** HyperText Markup Language
- **CSS:** Cascading Style Sheets
- **API:** Application programming interface
- **OS:** Operating System
- **JSON:** JavaScript Object Notation
- **NMBS:** Nationale Maatschappij der Belgische Spoorwegen
- **SSL:** Secure Sockets Layer
- **URL:** Uniform Resource Locator
- **UI:** User Interface
- **JSX:** Javascript XML
- **XML:** Extensible Markup Language
- **DOM:** Document Object Model
- **JVM:** Java Virtual Machine
- **PHP:** Hypertext Preprocessor
- **KISS:** Keep it simple and stupid

# Lijst van figuren

2.1	Onderscheid tussen een native, web en hybride applicatie [1]. . . . .	9
2.2	De Xamarin structuur, onderverdeeld in verschillende lagen [2]. . . . .	14
2.3	Het virtual DOM merkt een verschil en via algoritmes wordt uiteindelijk het reële DOM efficiënt geüpdatet [3]. . . . .	15
2.4	De bridge bevindt zich tussen Javascript VM en de native modules, die beiden op aparte threads runnen [4]. . . . .	16
3.1	Eerste prototypevoorstel met (a) het welkomstscherf, (b) het scherm met navigatie naar de perronplek aan bovenkant, (c) het scherm met navigatie naar de perronplek aan zijkant, (d) opening van submenu, (e), (f) en (g) kleinere minischermen die geopend staan en (h) het scherm als men op de juiste plek is. . . . .	19
3.2	Tweede prototypevoorstel met (a) het welkomstscherf, (b) keuzemenu voor de juiste trein, (c) het scherm met navigatie naar de perronplek aan zijkant, (d) alternatief scherm met navigatie naar de perronplek aan zijkant, (e), (f) en (g) kleinere minischermen die geopend staan en (h) het scherm als men op de juiste plek is. . . . .	21
3.3	Derde prototypevoorstel met (a) het welkomstscherf, (b) keuzemenu voor de juiste trein, (c) het scherm met keuzemogelijkheid voor rijtuig (d) navigatie naar de perronplek aan zijkant, (e), (f) en (g) kleinere minischermen die geopend staan en (h) het scherm als men op de juiste plek is. . . . .	23
3.4	Testapplicatie met (a) het HomeScreen, (b) het ChoiceScreen, (c) het PreferenceScreen, (d) het Wagonscreen, (e) het manueel wijzigen van de verkozen rijtuig in het WagonScreen, (f) het EndScreen en (g) het InfoScreen. . . . .	25
4.1	Globale structuur van de applicatie. . . . .	29
5.1	ER diagram die de connectie tussen de verschillende tabellen voorstelt. Per tabel zijn enkele interessante attributen hiervan weergegeven. . . . .	35
5.2	Geografisch coördinatenstelsel met een voorbeeldspoor. . . . .	39

5.3	Iconen van verschillende soorten objecten: (a) is een icoon voor een afdak, (b) een icoon voor trappen. . . . .	40
5.4	Technologieën waarmee locatiebepaling wordt gedaan [1]. . . . .	43
5.5	Mogelijke perronorienteerders van een perron in het geografisch coördinatenstelsel. . . . .	45
5.6	Visualisatie van de locatie op het perron. Punt E stelt hier de huidige afstand voor, lijnstuk l is de loodrechte afstand van punt E tot lijnstuk AB en lijnstuk m is de loodrechte afstand van punt E tot lijnstuk AD. . . . .	48
5.7	Verschil in nauwkeurigheid tussen twee positioneringsmethodes. Positioneringsmethode 1 heeft een hogere nauwkeurigheid, maar lagere precisie [1]. . . . .	49
5.8	2 situaties van locatiebepaling. De blauwe bol is de berekende huidige locatie, de grote blauwe cirkel is de nauwkeurigheidscirkel, de rode rechthoeken zijn de perrons. . . . .	50
5.9	Wanneer de huidige locatie zich op het perron bevindt, zal dit verschijnen als een bolletje (a). Wanneer deze zich niet op het perron bevindt, maar wel nog in de nauwkeurigheidscirkel ligt, verschijnt dit als een verticale balk (b) die de hoogte heeft van de hoogte van het perron. . . . .	50
5.10	Omzetting van het nauwkeurigheidsgetal in zijn y-coördinaat. . . . .	51
5.11	Een greep uit mogelijke rijtuigafbeeldingen die gerenderd kunnen worden: bovenaan een voorbeeld van een specifiek subtype trein (AM08M_a), in het midden een algemene afbeelding met een bepaalde lengte (hier 27 meter) en onderaan een algemene afbeelding. . . . .	53
5.12	Pictogrammen die in onze applicatie zullen worden gerenderd. . . . .	53
5.13	Navigatiebepaling eerste optie in het geografisch coördinatenstelsel. . . . .	55
5.14	Navigatiebepaling tweede optie in het pixel coördinatenstelsel. . . . .	58
5.15	Navigatiebepaling met perronobjecten . . . . .	59
5.16	Het keuzemenu in werking met de mogelijke voorkeuren getoond. . . . .	62
5.17	Miniatuurtrein (bovenaan) en trein in verhouding met het perron (onderaan). . . . .	65
5.18	Situatie waarin het linker rijtuig is geselecteerd, en de ander rijtuigen vervaagd worden. . . . .	72
6.1	Uittesten van de applicatie in Mechelen, 8 mei 2020, om 15:00. We bevonden ons op dit moment op het onderste perron (rode rechthoek). . . . .	78
6.2	Uittesten van de applicatie in Sint-Katelijne-Waver, 8 mei 2020, om 15:39. We bevonden ons op dit moment op het middelste perron. . . . .	79
6.3	Welkomstschermscherm in een (a) iOS omgeving en (b) Android omgeving, waar iets wordt ingetypt als eindbestemming. . . . .	81
6.4	Keuzescherm in een (a) iOS omgeving en (b) Android omgeving. . . . .	82

6.5 Bij het navigeren naar dit scherm wordt het begin van de trein weergegeven (a), als we scrollen, scrollt de miniatuurweergave mee, idem in de omgekeerde richting (b). . . . .	83
6.6 Na selectie wordt er automatisch naar een locatie op het perron gescrolld die bepaald is zodanig dat de huidige locatie in het midden staat (a). Het eindpunt van de locatie is de deur dat het dichtst bij de huidige locatie staat. . . . .	83
6.7 Het keuzemenu in (a) iOS en in (b) Android. . . . .	84
6.8 (a) Locatiebalk wanneer de locatie zich net niet op het perron bevindt en dit mogelijk een nauwkeurigheidsfout is. (b) Melding dat de gebruiker zich niet op de juiste positie op het perron bevindt. . . . .	84
6.9 Geopende modalscreen met informatie over wat de lengte en wat de algemene voorzieningen zijn in de trein. . . . .	85
6.10 Het eindscherm, dat tevoorschijn komt bij correcte positonering. . . . .	85
6.11 Het infoscreen. . . . .	86

# **Lijst van tabellen**

2.1 Overzicht van verschillende operatings systems . . . . .	4
2.2 Vergelijking soorten applicaties . . . . .	13

# **Hoofdstuk 1**

## **Situering en doelstelling**

### **1.1 Situering binnen het geheel**

Sinds de populariteit van de smartphone zijn mobiele applicaties een gegeven dat niet meer weg te denken is uit deze wereld. Van de 11 miljoen Belgen gebruiken er 10 miljoen internet en 9 miljoen een mobiel. Het is dan ook vanzelfsprekend dat de mobiele telefoon een extra middel is om de consument mee te bereiken en te verleiden. Bedrijven zoeken dan ook voortdurend naar vernieuwingen om mee te kunnen liften op het succes van de smartphone, denk bijvoorbeeld aan bankapplicaties waar nu ook Apple Pay wordt geïntroduceerd, of de QR codes die gebruikt kunnen worden om toegang te krijgen tot een museum etc. Het is dan ook niet onlogisch dat er naar manieren wordt gezocht om de huidige treinapplicatie aantrekkelijker te maken, door gebruik te maken van de bestaande functionaliteiten van de smartphone [5].

### **1.2 Situering binnen het bedrijf**

Deze masterproef wordt gemaakt in samenwerking met de NMBS. NMBS is de nationale spoorwegenmaatschappij van België. De trein wordt door 6,48% van de Vlamingen dagelijks gebruikt en is het op één na meest gebruikte dagelijkse openbaar vervoersmiddel in Vlaanderen. 45,07% van de Vlamingen gebruikt dit vervoersmiddel zelfs minstens 1 keer per jaar. Hierdoor is de aanbieding van een permanente, goede service aan de reiziger een belangrijk punt voor de NMBS [6].

### **1.3 Onderzoeksprobleem**

Op zich is er voor deze masterproef niet een probleem dat zich vooropstelt. Het is vooral ontstaan vanuit de instelling “Hoe maken we het reizen gemakkelijker?” Zo maken we het andersvaliden en

hun begeleiders eenvoudiger om op te stappen als ze weten waar het rijtuig voor andersvaliden zich bevindt. Ook voor anderen wordt het reizen een stuk minder complex, gezien ze niet in de drukte van een trein die gaat vertrekken moeten zoeken naar het correcte rijtuig. Vanuit de invalshoek van gemakkelijker reizen is het logisch dat men investeert in de smartphonemarkt.

## 1.4 Onderzoekstelling en Doelstelling

Het doel is om vanuit het bedrijf intern te kijken hoe we smartphonetechnologie optimaal kunnen implementeren in een treinapplicatie en zo te zien in welke mate dit nuttig kan zijn in het beter bedienen van de reiziger. Het persoonlijk doel is een applicatie te ontwikkelen die de gevraagde functionaliteiten zonder veel moeite kan uitvoeren, waarbij de functionaliteiten voldoende ontwikkeld zijn om een meerwaarde te kunnen bieden.

Daarbij werpen volgende stellingen zich op:

- Is het mogelijk om aan de hand van een smartphonelocatie de reiziger naar de juiste plaats op het perron te begeleiden?
- Is het mogelijk om de locatie van een treinstel en rijtuig aan het perron vooraf te bepalen?
- Is de huidige locatie van de gebruiker exact genoeg om de applicatie te laten werken?
- Bevat de databank genoeg informatie over de exacte samenstelling van de trein om de verschillende categorieën rijtuigen weer te geven?

# **Hoofdstuk 2**

## **Literatuurstudie**

### **2.1 De verschillende IT ecosystemen**

Alvorens we de verschillende soorten applicaties bespreken, gaan we eerst kijken wat de verschillende ecosystemen zijn waar onze applicaties kunnen terechtkomen. Een ecosysteem is een verzameling van software- en hardware projecten die in eenzelfde omgeving evolueren en één samenwerkingsnetwerk vormen. We bespreken hier de 3 grootste ecosystemen: Apple, Google en Microsoft [7].

#### **2.1.0.1 Het Apple ecosysteem**

Het Apple platform is een groot ecosysteem, het omvat onder andere de iPhone, iPad, MacBook, Apple Watch, Apple TV. Al deze producten hebben een app store waar apps volgens bepaalde richtlijnen in kunnen verschijnen. Acceptatie en goedkeuring gebeurt door Apple, die de inkomsten verdeelt onder een 70-30 ratio: 30% gaat naar Apple en 70% naar de ontwikkelaar. Apple bevordert het ontwikkelen door verschillende functies van zijn toestellen door API's vrij te geven aan de ontwikkelaars. Het ontwikkelen van applicaties gebeurt gewoonlijk in Swift met XCode [7].

#### **2.1.0.2 Het Google ecosysteem**

Google maakt gebruik van het Android OS. Dit is een open source platform en besturingssysteem voor verschillende mobiele telefoons, tablet-pc's, camera's en meer. Doordat het open source is en de meeste smartphone devices niet door Google zelf gemaakt worden, moet er rekening gehouden worden met variatie in beeldscherm, resolutie, beschikbare features,... Programmeren van Android applicaties gebeurt in Java. De inkomstverdeling is net als bij Apple 70-30 [7].

### 2.1.0.3 Het Microsoft ecosysteem

Op vlak van mobiele operating systems is Microsoft nooit door gebroken. Het heeft enkele malen geprobeerd met verschillende versies van Windows Phone en Windows 10 Mobile maar succes bleef uit. Naast de Windows Phone is er echter wel de Surface en Windows Desktop, die de gezamenlijke Windows App Store delen. De programmeertaal voor deze applicaties is voornamelijk C# en de ratio is ook 70-30. Omdat Microsoft praktisch niet meer meedoet op vlak van mobiele operating systems, houden we verder geen rekening met dit ecosysteem voor de ontwikkeling van onze mobiele applicatie [7] [8].

## 2.2 Mogelijkheden in het maken van een mobiele applicatie

Er zijn verschillende mogelijkheden om een mobiele applicatie te maken, zo kan je een webgebaseerde applicatie maken, applicaties voor een specifiek platform maken (native) en applicaties die werkzaam zijn op verschillende platformen (hybride). We zullen hieronder de verschillende platformen bespreken.

### 2.2.1 Native Applicaties

Native applicaties worden gemaakt aan de hand van een platform specifieke programmeertaal. Hierdoor zijn ze heel snel, kunnen ze alle platformspecifieke software en hardware implementeren en hebben ze een goede gebruikerservaring. In onderstaande tabel worden de verschillen tussen de programmeertalen voor elk platform duidelijk gemaakt.

**Tabel 2.1** Overzicht van verschillende operatings systems

Bedrijf	Operating System	Programming Languages	Platform	Store
Apple	iOS	Swift	Mac OS X	App Store
Google	Android	Java	Windows, Linux, Mac OS X	Google Play

#### 2.2.1.1 Voordelen

- **Snelheid en responsiviteit:** Doordat native apps gespecialiseerd zijn voor een specifiek platform en de code gecompileerd wordt naar de core specifieke programmeertaal en API's,

hebben ze de beste snelheid en responsiviteit. Ook wordt de applicatie gedownload, wat de snelheid van het toestel op zich verhoogt, aangezien bijvoorbeeld afbeeldingen en de UI er al op staan. De specifieke programmeertaal zorgt er ook voor dat hardwarecomponenten rechtstreeks worden aangesproken, ook dit zorgt voor een optimale performantie [9].

- **Beveiliging:** Doordat native applicaties gebruikmaken van een platform specifieke programmeertaal, kan ook de data beter beschermd worden [10].
- **Functionaliteit:** Door geen cross-platform te gebruiken, is de kans op bugs kleiner, aangezien de cross-platform code moet omzetten. Ook kan het alle features van de smartphone meteen implementeren, aangezien dit door native code direct ondersteund wordt, in tegenstelling tot een cross-platform applicatie.
- **Vindbaarheid:** De applicatie is uiteraard te vinden in de app store. Doordat de app in een native taal wordt geschreven, is het ook gemakkelijker het validatieproces met succes te voltooien [9].

### 2.2.1.2 Nadelen

- **Herbruikbaarheid:** Het grootste nadeel van native applicaties is voor de hand liggend: het is niet cross-platform. Geprogrammeerde code kan amper tot weinig gerecupereerd worden in andere code [9].
- **Kost:** Doordat code weinig tot niet kan gerecupereerd worden, zorgt dit voor een hoge maak- en onderhoudskost. Ook is het bereik van één platform kleiner, wat de (mogelijke) winst dan ook vermindert. Een oplossing kan zijn om de applicatie in verschillende programmeertalen te schrijven, maar dan moet je ofwel de ontwikkelaar trainen in verschillende programmeertalen, ofwel verschillende ontwikkelaars aanwerven. Dit is uiteraard weer een verhoogde kost of tijdsbesteding [11].

### 2.2.2 Webgebaseerde applicaties

Een andere mogelijkheid is een webgebaseerde applicatie ontwikkelen. Die wordt meestal gemaakt aan de hand van Javascript, HTML5 en CSS. Webgebaseerde applicaties zijn nooit echt doorgebroken. De snelheid is nooit zo optimaal als een native of hybride applicatie, omdat webapplicaties niet voor een specifiek platform worden gemaakt.

Het verschil tussen een website en een webapplicatie is het feit dat een webapplicatie interactiever is in gebruik dan een website. Beide worden wel als website geladen en niet zoals de andere apps geïnstalleerd in de appstore [12].

### 2.2.2.1 Architectuur

Een webgebaseerde applicatie werkt net zoals een website met een client-server model. Er is dus een programma dat aan de clientkant werkt en antwoordt op de input, en een programma dat aan de serverkant werkt en antwoordt op HTTPRequests. De serverkant kan geschreven worden in elke code die op HTTPRequests kan antwoorden. Het bevat de data en de aangevraagde pagina en wordt niet gezien door de gebruiker. De clientkant kan geschreven worden in Javascript, HTML en CSS, kan gezien en bewerkt worden door de gebruiker en kan enkel met de server communiceren aan de hand van HTTPRequests [13].

### 2.2.2.2 Voordelen

- **Geheugenruimte:** Doordat webapplicaties gewoon via de browser worden bereikt, is er weinig tot geen geheugenruimte nodig van de smartphone. Ook zal de app altijd up-to-date zijn, aangezien ze bij elk bezoek opnieuw moet worden ingeladen [14].
- **Uniform:** Doordat ze niet moet worden geïnstalleerd is het ook vanaf elke smartphone bereikbaar. Door de uniforme omgeving moet de applicatie maar op één besturingssysteem getest worden, om te weten dat het op alle besturingssystemen zal werken [15].
- **Beveiliging data:** Webgebaseerde applicaties interageren meestal ook met dedicated servers, die worden bewaakt en onderhouden door ervaren serverbeheerders. Dit zorgt voor een hogere beveiliging van data op de servers dan bij gewone smartphoneapplicaties [16].
- **Kosten en tijd:** De kosten zijn waarschijnlijk gelimiteerd, dit is omdat de webapplicatie meestal al een website had, die dan gewoon licht wordt aangepast tot een webapplicatie. Het is dan ook heel tijdsbesparend [15].

### 2.2.2.3 Nadelen

- **Online:** Een webgebaseerde applicatie is een client-server applicatie: een server is vereist, bij verlies van wifi of 4G, zal de applicatie niet meer werken [15].
- **Performantie:** De user interface staat niet offline en is daarom ook niet responsive (snel reagerend). De applicatie kan ook minder snel werken omdat deze bij elke nieuwe pagina verbinding moet maken met het net [15].
- **Compatibiliteit:** De app moet dan wel maar op één besturingssysteem getest worden, maar doordat het eigenlijk een soort website is, moet de app wel ondersteunend zijn voor alle verschillende browsers. [16]

- **Beperkte functies:** Doordat er niet echt iets geïnstalleerd wordt, kunnen van de applicatie geen achtergrondprocessen werken op de smartphone wanneer de website niet openstaat. Ook heeft een webapplicatie geen toegang tot native API's, waardoor geolocatie, camera en andere API's niet kunnen gebruikt worden [15].
- **Beveiliging:** De grote zwakte van webapplicaties is cross-site scripting: dit is de invoer van een webapplicatie die verkeerd geïnterpreteerd, verwerkt en aan de eindgebruiker doorgegeven wordt. Een veelvoorkomend fenomeen is "Phishing". De gebruiker klikt daarbij op een link waarbij vervolgens de cross-site scripting aanval wordt gestart [17].

### 2.2.3 Progressive Web Apps

Progressive Web Apps (PWA's) zijn webapplicaties die gebruikmaken van webtechnologie, maar meer kunnen dan gewone webapplicaties. Voorbeelden zijn het gebruik van pushnotificaties (berichten die op het scherm verschijnen zonder dat de applicatie openstaat) en locatieservices.

#### 2.2.3.1 Architectuur

PWA's worden opgebouwd in 3 delen: eerst is er de app shell, gemaakt in Javascript, HTML en CSS. Deze is offline op het device beschikbaar, waardoor ze ook sneller laadt op het device dan gewone webapplicaties. Vanaf het moment dat deze webapplicatie voor het eerst geladen wordt, laadt hij het design in de cache, waarbij bij volgende bezoeken dit offline gebeurt [18].

Het tweede deel is een service worker. Service workers zijn scripts die in de achtergrond geladen worden, waardoor features als pushnotificaties geïnstalleerd worden. Het is de cruciale factor in het niet (altijd) nodig hebben van een internetconnectie bij deze applicaties. Een service worker heeft een andere levenscyclus dan de webpagina waarin hij werd aangemaakt. Het werkt parallel met de webpagina en het roept API's op voor pushnotificaties, background synchronization enz. Bij background synchronization wordt er gewacht tot er een stabiele connectie is voor onder andere berichten te zenden. Een service worker heeft 2 toestanden:

1. het kan in een terminated toestand zijn waarin het zijn levenscyclus beëindigt en het weer opstart wanneer nodig
2. het zit in een fetch/message toestand waarin het gebeurtenissen zal ophalen of berichten wanneer een netwerkverzoek- of bericht is aangevraagd vanuit de PWA [19].

Derde deel is een web app manifest. Dit is een JSON file die toelaat dat de applicatie geïnstalleerd wordt via een browser en op het homescreen kan worden geplaatst [20] [21].

### 2.2.3.2 Voordelen

- **Geheugenruimte:** PWA's werken nog altijd deels online, waardoor ze weinig ruimte in het geheugen innemen [18].
- **Performantie:** Doordat sommige delen eenmaal online ingeladen worden en dan offline werken, zijn ze sneller dan gewone webapplicaties. In vergelijking met native en hybride applicaties loopt de performantie en snelheid echter nog ver achter [21].
- **Kost:** De kosten zijn minder dan native applicaties, doordat ze via het web worden gemaakt en voor elk platform kunnen worden gebruikt. [22]
- **Compatibiliteit:** PWA's zijn volledig onafhankelijk van elk platform. Ze worden ondersteund in zowel Apple, Android als Windows Phone [18].
- **Functies:** Ontwikkelaars kunnen ook gemakkelijk hun PWA updaten zonder het goedkeuringsproces van een app store te doorlopen. Toch hebben ze toegang tot specifieke tools waar normale webapplicaties geen toegang tot hebben zoals camera-applicaties. [18]

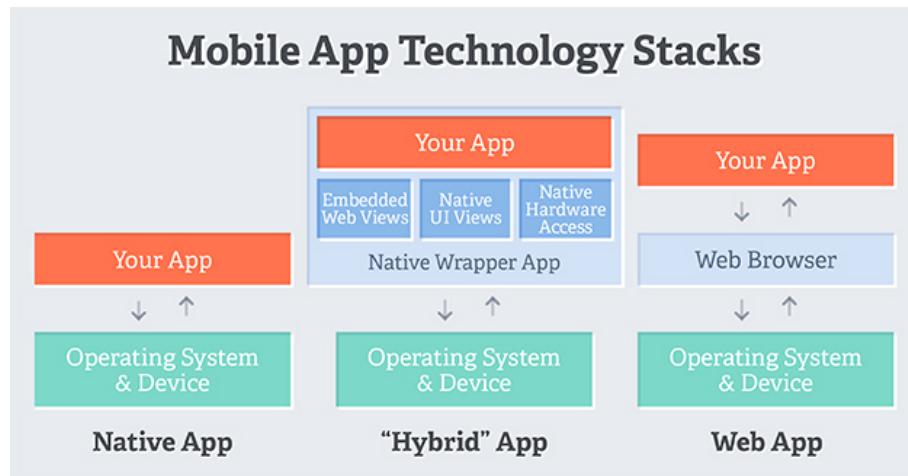
### 2.2.3.3 Nadelen

- **Ondersteuning afhankelijk van OS:** De mogelijkheden van PWA's zijn erg afhankelijk van het platform. Zo is het in iOS niet mogelijk om meer dan 50 Mb op te slaan voor offline gebruik en is er geen ondersteuning voor toepassingen die native apps wel ondersteunen zoals pushnotificaties, TouchID, FaceID, Siri integratie etc. Dit is één van de grootste nadelen: de beperkte iOS-ondersteuning, waardoor dit de evolutie van de PWA vertraagt. [18]
- **Vindbaarheid:** Doordat het niet in de App Store te vinden is, zullen PWA's mogelijk ook niet het juiste publiek vinden. Wel kunnen ze door zoekmachines opgemerkt worden. [21]
- **Functionaliteit:** Een PWA kan meer dan een webapplicatie, maar er zijn nog steeds gebreken, denk aan het gebrek van onder andere geolocatie, wat normale apps wel kunnen [23].

### 2.2.4 Hybride Applicaties

Hybride applicaties zijn applicaties die op elk smartphoneplatform werken. Ze zijn webgebaseerd en de onderliggende code is voornamelijk Javascript, HTML en CSS. Het zijn dus (simplistisch gezien) webapplicaties die geïnstalleerd worden zoals native apps. Ze onderscheiden zich van webapplicaties doordat ze toegang hebben tot native API's. Het verschil met een PWA is voornamelijk dat een PWA direct vanuit het web gebouwd wordt.

### 2.2.4.1 Architectuur



Figuur 2.1: Onderscheid tussen een native, web en hybride applicatie [1].

Hybride applicaties zijn webapplicaties die rond een native browser gebouwd worden. Via een native wrapper heb je toegang tot de verschillende "webpagina's". Deze worden via webview in de applicatie geladen. Een goed voorbeeld is Instagram. Deze bestaat uit native applicatieknoppen en menubalk, maar de timeline op zich is een volledige webcode die via webview wordt ingeladen. In figuur 2.1 wordt het verschil getoond ten opzichte van een native en een webapplicatie. Waar in de webapplicatie de app geladen wordt via een browser, wordt bij een hybride applicatie de app geladen via een native wrapper app, deze bevat een WebView, een native UI view en een native hardware toegang. [1]

### 2.2.4.2 Voordelen

- **Herbruikbaarheid:** Het grootste voordeel van hybride applicaties is natuurlijk dat een groot deel van de code platformonafhankelijk is. Dat maakt de code makkelijker te beheren omdat je maar één code hebt om bv. features toe te voegen, waardoor je er ook **tijd en kosten** mee bespaart [24].
- **Vindbaarheid:** Hybride applicaties kunnen wel in app stores gevonden worden. Dit open mogelijkheden zoals in-app aankopen en een eenvoudige vindbaarheid [24].
- **Opslag:** Doordat de meeste content in webview pagina's wordt ingeladen, wordt er weinig op de telefoon zelf opgeslagen [24].
- **Functionaliteit:** Als we hybride applicaties vergelijken met webapplicaties zien we nog een voordeel: het kan wel native API's implementeren waardoor het toegang heeft tot functies als camera, geolocatie e.d. [24]

- **Tijd en kosten:** Doordat de app nog altijd voornamelijk in Javascript/CSS geschreven wordt, is de hoeveelheid tijdsbesteding gelijkaardig aan deze van een webgebaseerde applicatie. Enkel de native API's zullen iets meer tijd en kost vereisen, maar verhogen ook de complexiteit en kwaliteit [15].

#### 2.2.4.3 Nadelen

- **Herbruikbaarheid:** Ondanks dat de meeste code platformonafhankelijk is, zullen er toch delen nog altijd moeten aangepast worden: eisen voor acceptatie verschillen tussen Apple, Google en Microsoft. Ook kan je geen functies als geolocatie en camera gebruiken zonder dat je native API's aanspreekt. Het native deel is hier dus zeker nog aanwezig [15].
- **User Interface:** De user interface van een applicatie wordt gebouwd vanuit een bepaald framework, om het zoveel mogelijk op native te doen lijken. De bitmap compositie gebeurt echter niet in de hardware, wat wel zo is bij native applicaties. Dit vermindert de responsiviteit. Ook kunnen sommige grafische functies niet geïmplementeerd worden [15].
- **Beveiliging:** De veiligheid van hybride applicaties is minder in vergelijking met native applicaties. Dit is doordat native applicaties een eigen beveiligingslaag hebben, voorzien door de platformeigenaar. Bij hybride applicaties kan je dus best wat extra beveiliging inbouwen op volgende manier:
  - **Framework en plugins up-to-date houden:** hier kunnen zich kwetsbaarheden in bevinden.
  - **Cross-site scripting:** dit is een beveiligingsfout: input wordt ontvangen, fout verwerkt en komt bij de eindgebruiker terecht. Hiermee kan kwaadaardige code geïmplementeerd worden. Een oplossing is het gebruik van een soort teller.
  - **URLs whitelisting:** door de URL the whitelisten, kan de app enkel met die URL communiceren, wat het risico op aanvallen vermindert.
  - **Input encoding:** Wanneer je invoer ingeeft, is dit een zwak punt. Het encoderen ervan vermindert het risico.
  - **Servercommunicatie:** Wanneer een applicatie met een server communiceert, kan je best een SSL gebruiken. Dit zorgt voor een beveiligde verbinding tussen twee systemen.
  - **Code obfuscatie:** Dit is het bemoeilijken van broncode, maar toch de functionaliteit behouden. Doordat het moeilijker leesbaar is, is het ook moeilijker om kwaadaardige software in de code te injecteren [10].

### 2.2.5 Cross-path Apps

Dit is een alternatieve vorm van een hybride applicatie. De omgeving zorgt voor een extra laag waarin de code geschreven wordt. De geschreven code wordt dan door een bridge gecompileerd naar de juiste code base voor een bepaald platform. Voorbeelden, die later besproken worden, zijn Xamarin en React Native. Het is deze extra laag waarin de code geschreven wordt, dat zorgt voor een discussie of we deze soort applicaties kunnen beschouwen als een hybride applicatie of een aparte categorie. Er wordt ook veel minder gewerkt met webviews dan bij hybride applicaties. Door de snelle evolutie in de technologie worden hybride en cross-path applicaties vaak met elkaar verward, en lijken cross-path applicaties de plaats van hybride applicaties meer en meer in te nemen [25].

#### 2.2.5.1 Voordelen

- **Performantie:** De performantie is op bijna gelijke hoogte als native apps, doordat er gecompileerd wordt naar de code base van het gewenste platform [25].
- **Kost en tijd:** Het is uiteraard ook kost- en tijdsbesparend, aangezien veel code herbruikbaar is [26].
- **User Interface:** De user interface kan geheel omgezet worden naar pure native elementen. Dit verhoogt de **responsiviteit** aanzienlijk [26].

#### 2.2.5.2 Nadelen

- **Afhankelijkheid:** Wanneer er nieuwe features zijn voor een bepaalde smartphone, moet je wachten tot bv. React Native of Xamarin deze ondersteunt, vooraleer je deze features in je app kan verwerken [26].

### 2.2.6 Conclusie

In tabel 2.2 lijsten we de verschillende soorten applicaties die we besproken hebben op. Daarnaast vergelijken we elementen die ons belangrijk lijken voor de applicatie die we willen maken. Zo is veiligheid van de treindata en een snelle bediening belangrijk voor zowel de treinorganisatie en de reiziger. Het hebben van geolocatie is ook een vereiste. Het is namelijk een belangrijk deel van de masterproef om de reiziger te begeleiden naar het juiste perron op basis van de huidige locatie. Ook is het interessant om een herbruikbare code te hebben, aangezien we niet enkel reizigers met een bepaald type smartphone willen bedienen.

Als we deze elementen in gedachte nemen en de tabel erbij nemen, zien we dat op basis van geolocatie zowel de webgebaseerde applicatie, de PWA en de hybride applicatie afvallen. Omdat

herbruikbare code ook interessant is, komen we uit bij cross-path applicaties, wiens veiligheid en snelheid ook voldoende is.

Voor deze masterproef is het het handigst om te werken met Reactive-Native of Xamarin. Verdere verduidelijking voor onze keuze voor cross-path is onder andere dat webapplicaties niet dezelfde performantie leveren als native applicaties. PWA's hebben dan weer beperkte mogelijkheden, met nieuwe technologie die niet alle functies kan integreren die native applicaties biedt. Daarbij wordt het ook niet volledig ondersteund in iOS.

**Tabel 2.2** Vergelijking soorten applicaties

Soort Applicatie	Programmeertaal	Veiligheid	Snelheid	Geolocatie	Herbruikbaar
Webgebaseerd	Javascript, HTML5, CSS	-	--	Niet ondersteund	Volledig
PWA's	Javascript, HTML5, CSS, JSON	-	-	Niet ondersteund	Volledig
Hybride	Javascript, HTML, CSS	+ -	+ -	Ondersteuning afh van framework	Gedeeltelijk
Cross-path	C, Dart, JSX	+	+	Ondersteund	Gedeeltelijk
Native	Swift of Java	++	++	Ondersteund	Niet

## 2.3 Relevante technologiën

Aangezien we herbruikbare code willen, kunnen we voor onze implementatie kiezen tussen Xamarin, React Native en Flutter. Er zijn uiteraard nog meer mogelijkheden, maar we beperken ons tot deze, aangezien deze het bekendste zijn, de meeste libraries hebben en een grote community achter zich hebben. Deze voordelen zullen ons bij de ontwikkeling dan ook zeker helpen.

### 2.3.1 Xamarin

Xamarin is een door Microsoft overgenomen platform. De programmeertaal is C# en compileert deze dan naar de juiste native taal. Xamarin heeft meer dan 1.7 miljoen ontwikkelaars en is daarmee de tweede grootste ontwikkelaarssoftware voor cross-path applications [27].

#### 2.3.1.1 Architectuur



Figuur 2.2: De Xamarin structuur, onderverdeeld in verschillende lagen [2].

Xamarin is gebouwd op een open-source .NET framework met iOS en Android specifieke libraries in C#. Xamarin.forms zorgt ervoor dat cross-platform UI code wordt geconverteerd naar platform-specifieke API's. Hiermee wordt hergebruik gemaximaliseerd. Voor platform-specifieke code hebben we Xamarin.ios en Xamarin.Android. Het verschil tussen Xamarin.iOS en Xamarin.Android is het feit dat Apple geen Just in Time compilation ondersteunt. Just in Time compilatie is een term die gebruikt wordt om een interpreter te benoemen die geen broncode naar machinetaal omzet, maar naar een tussentaal of deze direct uitvoert. Doordat Apple dit niet ondersteunt, wordt hier gebruikgemaakt van Ahead of Time compilatie, waarmee code gecompileerd wordt naar native machine code. Hierdoor wordt er bij elke update van iOS ook een ondersteuningsupdate

uitgebracht door Xamarin. In figuur 2.2 wordt deze structuur grafisch voorgesteld [2].

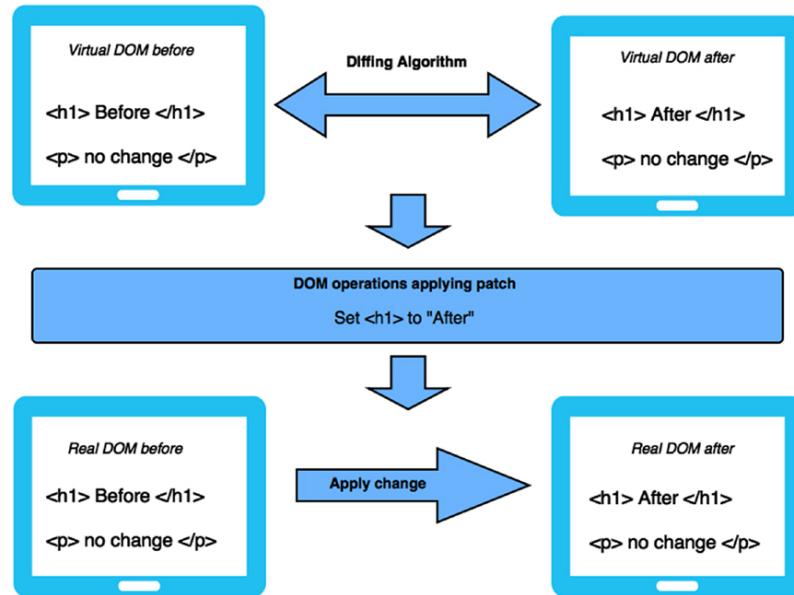
### 2.3.2 React Native

Een andere mogelijkheid is React Native. De programmeertaal is gebaseerd op Javascript: JSX, dit staat voor Javascript en XML. Het werd ontwikkeld door Facebook developers, en is ondertussen uitgegroeid tot één van de meest gebruikte talen voor applicaties. Hierdoor heeft het ook een grote community [28].

#### 2.3.2.1 React

React Native komt voort uit React. Dit is een Javascript framework ontwikkeld door Facebook ingenieurs om een UI, waarvan data over tijd verandert werkend te houden. Het wordt gebruikt in verschillende applicaties zoals Facebook, Instagram, Tesla,... Al moet men hierbij een kanttekening maken: sommige aspecten worden in native code geschreven, anderen dan weer in React Native code. Zo worden componenten die te maken hebben met geld, verplicht door Apple om in native code te worden geschreven [29].

#### 2.3.2.2 Virtual DOM

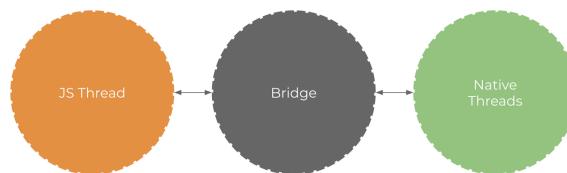


**Figuur 2.3:** Het virtual DOM merkt een verschil en via algoritmes wordt uiteindelijk het reële DOM efficiënt geüpdatet [3].

Het Document Object Model (DOM) is een boom model bestaande uit knopen van alle HTML tags. Wanneer de Javascript wordt aangepast, moet de geüpdateerde boomstructuur door de browser

opnieuw gerenderd worden. Het telkens opnieuw renderen en updaten kan vermeden worden door Virtual DOM. In de Virtual DOM wordt een representatie van de DOM bijgehouden voor er gerenderd wordt. Bij elke update wordt niet de DOM gerenderd, maar toegepast op de virtuele representatie. Op basis van vergelijking kan er dan via slimme algoritmes efficiënter de echte DOM geüpdatet en gerenderd worden zodat niet de gehele DOM altijd opnieuw moet gerenderd worden. Dit is te zien in figuur 2.3 waar de representatie vergelijken wordt met het virtuele DOM. Er wordt door een algoritme een verandering gemerkt (after tussen `<h1>`) en dit wordt uiteindelijk toegepast op het reële DOM [30].

### 2.3.2.3 Bridge



**Figuur 2.4:** De bridge bevindt zich tussen Javascript VM en de native modules, die beiden op aparte threads runnen [4].

Hoe gebeurt de overgang van virtual DOM naar het scherm? De virtual DOM vertelt hoe het er zou moeten uitzien en aan de hand van een bridge wordt dit geconverteerd naar platform-specifieke API's zoals Android of iOS [28].

Zoals te zien in figuur 2.4 zit de bridge tussen de Java Virtual Machine (JVM) en de native modules. De Java Virtual Machine is de virtuele processor van Java, waar al de code die we geschreven hebben op runt. In tegenstelling tot talen als C wordt Java code niet gecompileerd naar machinecode, maar naar de Java bytecode. Dit heeft als grootste voordeel dat het platformonafhankelijk werkt [31].

Het communiceren via de bridge gebeurt via een aangepast protocol. Bij het opstarten van een applicatie, wordt eerst een beetje native code gerund, vervolgens start de JVM op die de code van de applicatie inlaadt. Via de bridge communiceert de JVM thread dan naar de native componenten instructies om onder andere de UI te tekenen, een bepaald scherm te tekenen,... Wanneer dit voltooid is, communiceert de native thread dit naar de JVM [28].

### 2.3.3 Flutter

Een laatste mogelijkheid is Flutter. Flutter is de oplossing van Google voor cross-platform applicaties. Het wordt geschreven in Dart, een programmeertaal die ontwikkeld is door Google voor het maken van snelle apps op meerdere platformen [32].

### 2.3.3.1 Architectuur

Het grote verschil van Flutter ten opzichte van Xamarin en React Native is dat Flutter geen native componenten rendert maar gewoon zijn eigen User Interface componenten. Verder bestaat het uit 4 grote delen: Dart, the main flutter engine, the foundation library en widgets.

The main flutter engine is een C++ gebaseerde codebase. Het implementeert Flutters kernbibliotheeken, zoals animaties en grafieken, compileren, ... The foundation library is een basisbibliotheek die ervoor zorgt dat de verschillende API's van de verschillende vormen samensmelten in één API die in de code kan gebruikt worden. Wanneer je dus deze API oproept, zal deze voor u de API voor het gewenste platform activeren. Widgets bevatten de verschillende user interface onderdelen, zoals RaisedButton, Text,... [32]

### 2.3.3.2 Dart

Dart is een door google gecreëerde object-georiënteerde programmeertaal. Het voordeel van Dart is dat het een syntax heeft die niet veel verschilt van andere object-georiënteerde talen. Ook is het erg gelijkaardig aan Java(script). Zo heeft het onder andere ook een garbage collection feature. Dit betekent dat het aan automatisch geheugenbeheer doet, waardoor je je geen zorgen moet maken over allocatie en deallocate. Het heeft meerdere run environments. Als je het wilt runnen in je webbrowser, wordt het gecompileerd in Javascript maar je kan het ook in andere talen compileren. Dit is één van de grootste voordelen, want het heeft hierdoor een hoge performance [33].

### 2.3.4 Verantwoording keuze

Verschillende argumenten spelen mee in onze keuze. We kiezen voor een stabiel framework. Aangezien Flutter nog redelijk nieuw is, lijkt dit ons niet de beste keuze. React Native heeft de grootste community en heeft de meeste documentatie beschikbaar. Het lijkt ons dan ook het beste om met deze programmeertaal verder te gaan.

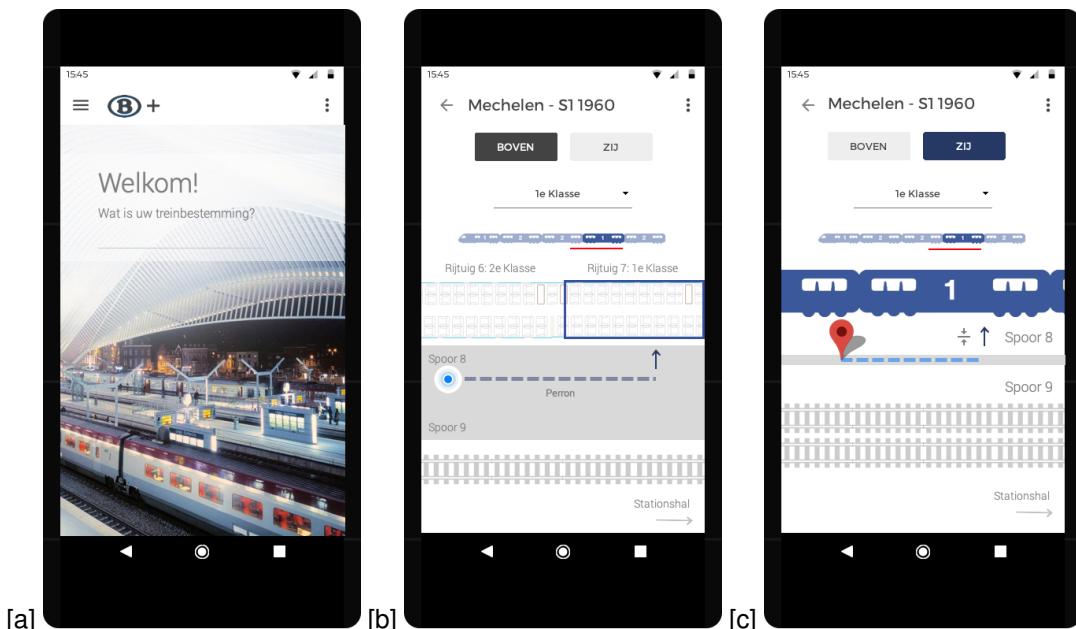
# Hoofdstuk 3

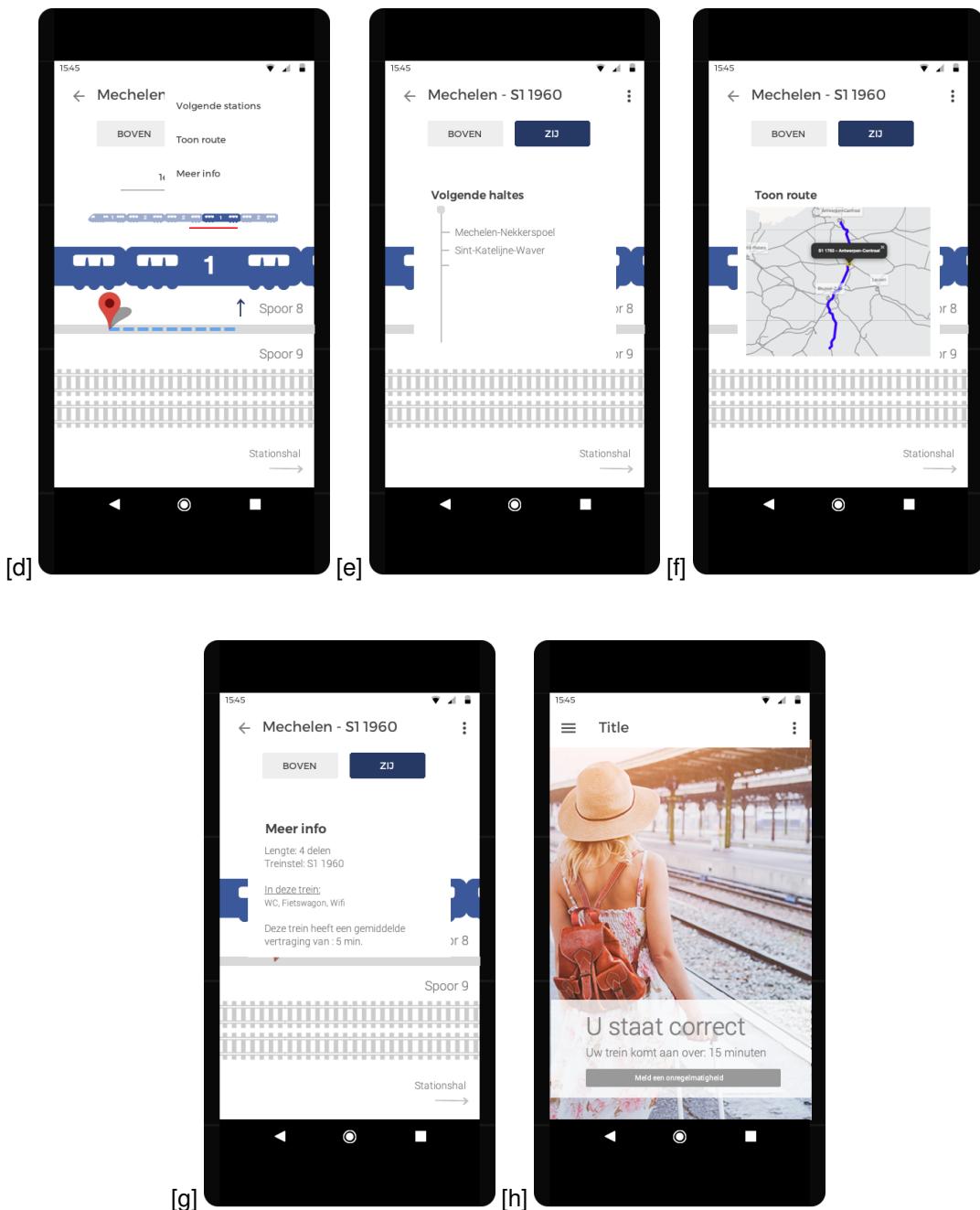
## Conceptstudie

Voordat we aan het schrijven van onze code beginnen, gaat hier een proces van denkwerk rondom de vormgeving aan vooraf. Voor deze mobiele applicatie hebben we dit gedaan aan de hand van prototypes, waarmee we een visuele voorstelling geven aan de mobiele applicatie.

### 3.1 Prototype 1

In het eerste prototype zijn er verschillende schermen gemaakt: een welkomstscherm; een scherm met de te volgen route naar het juiste rijtuig op basis van je huidige locatie en met keuze tussen bovenzicht of zijzicht; enkele minivensters die zeggen wat de volgende haltes zijn, wat de gehele route is en info over de rijtuigen, lengte, ... Er is ook een eindscherm dat tevoorschijn komt wanneer je op de juiste plek op het perron staat.

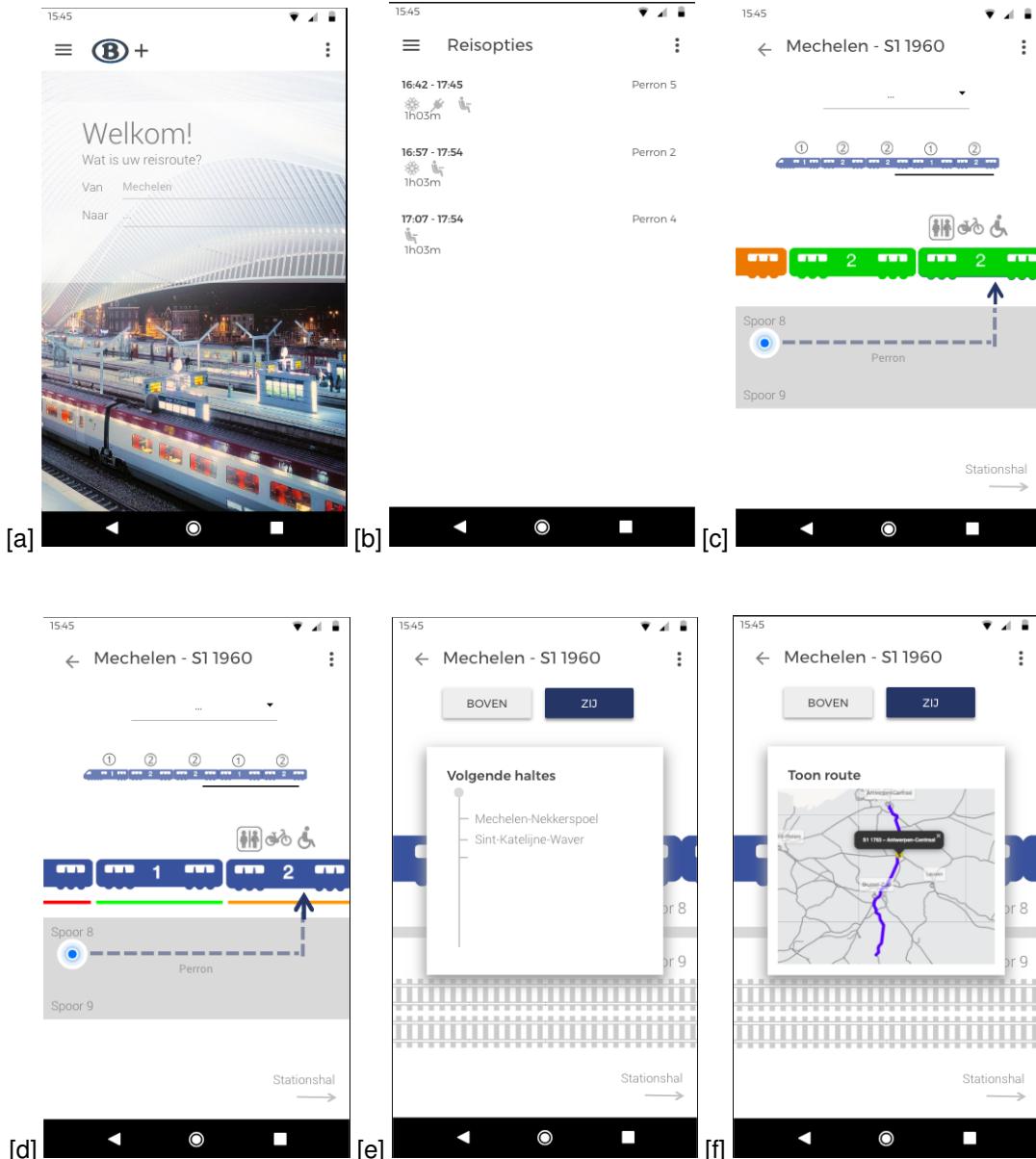


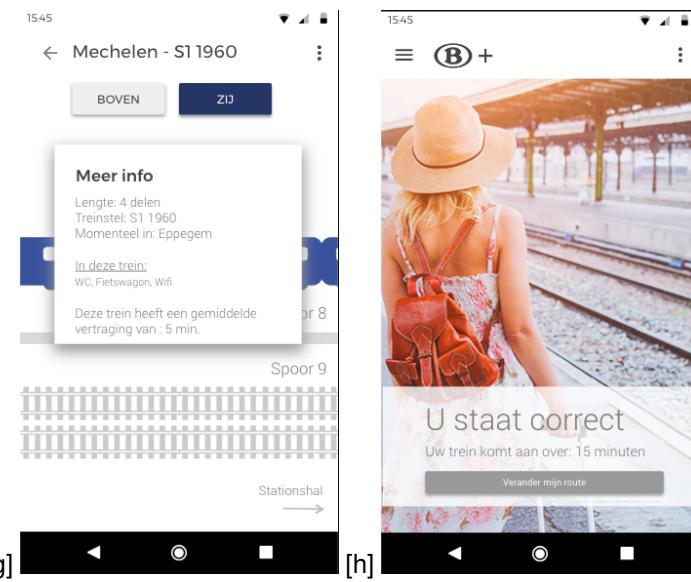


**Figuur 3.1:** Eerste prototypevoorstel met (a) het welkomstscherms, (b) het scherm met navigatie naar de perronplek aan bovenkant, (c) het scherm met navigatie naar de perronplek aan zijkant, (d) opening van submenu, (e), (f) en (g) kleinere minischermen die geopend staan en (h) het scherm als men op de juiste plek is.

## 3.2 Prototype 2

In het tweede prototype zijn er enkele zaken aangepast: zo kan je in het welkomstscherm zeggen van welk station je wil vertrekken en naar welk eindstation je wil reizen, is er een keuzemenu voorzien (b) waarmee je de juiste trein kan selecteren. (c) en (d) geven de mogelijke aanduidingen van de bezettingsgraad aan. De minischermen blijven onveranderd.

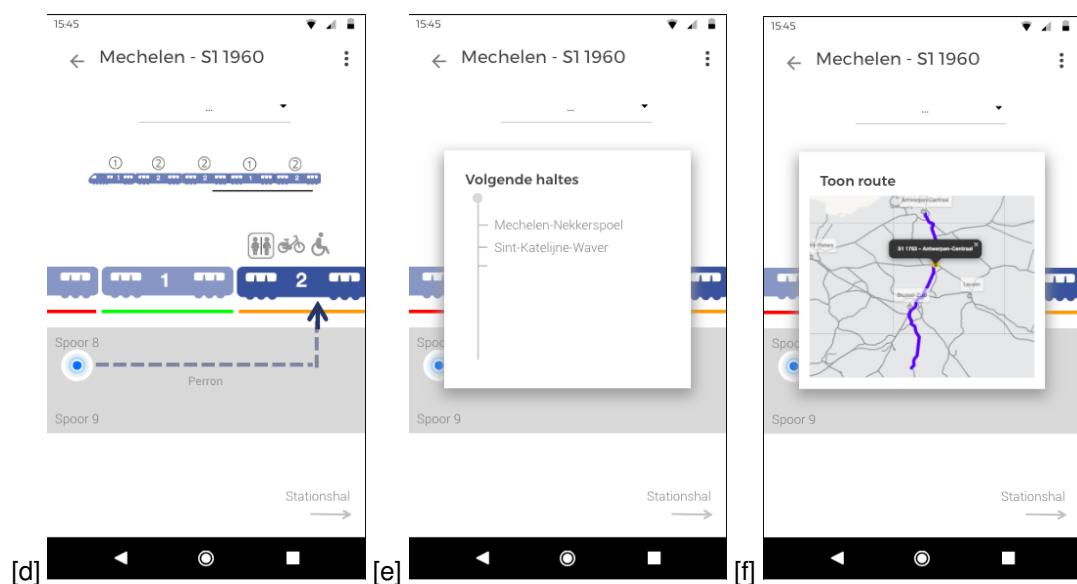
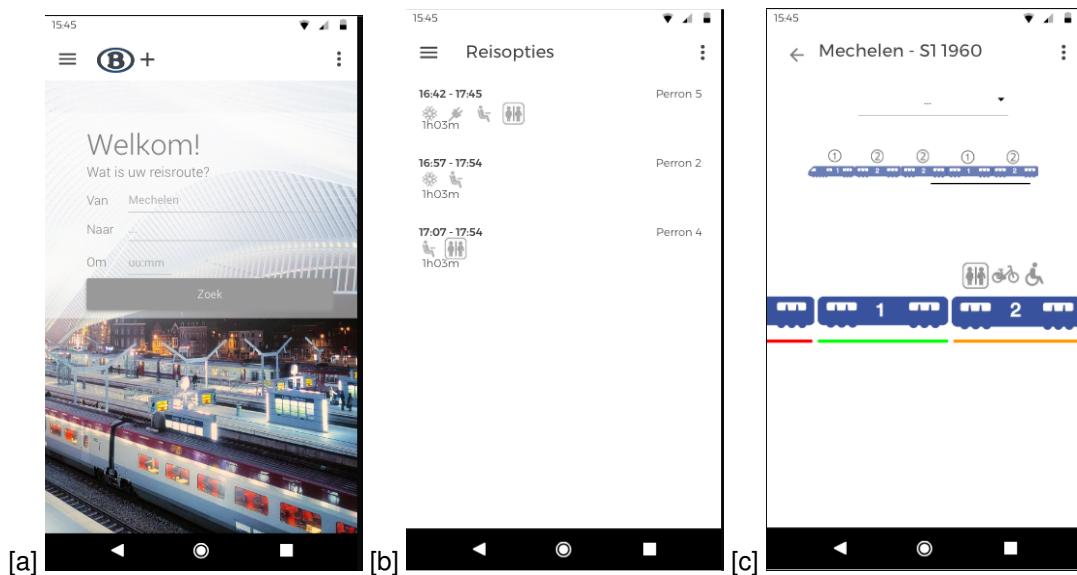


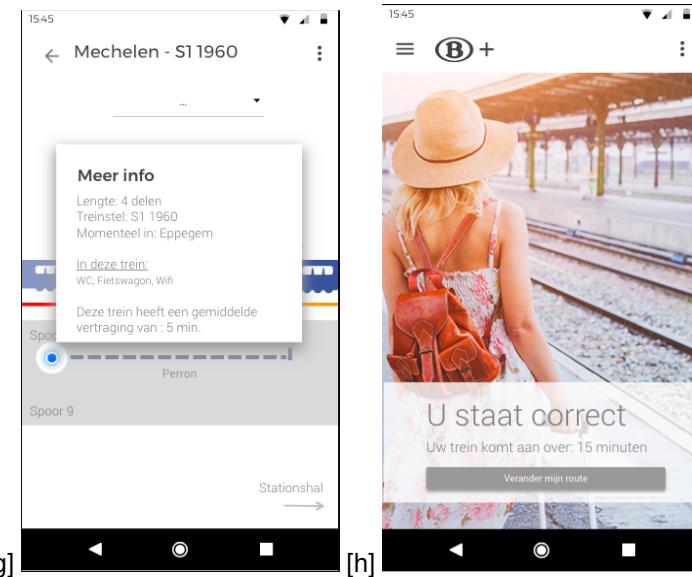


**Figuur 3.2:** Tweede prototypevoorstel met (a) het welkomstscherf, (b) keuzemenu voor de juiste trein, (c) het scherm met navigatie naar de perronplek aan zijkant, (d) alternatief scherm met navigatie naar de perronplek aan zijkant, (e), (f) en (g) kleinere minischermen die geopend staan en (h) het scherm als men op de juiste plek is.

### 3.3 Prototype 3

In het derde prototype zijn verwarringe elementen gewijzigd. Zo kan je ook een uur ingeven op het welkomstscherf; staat er ook een pictogram als er een toilet is op de trein bij het keuzemenu (b) en is er eerst enkel een trein te zien met de bezettingsgraad per rijtuig in (c). Pas als het rijtuig geselecteerd is komt er een perron tevoorschijn met de route die je moet lopen (d). Ook is er geen optie meer tussen bovenkant en zijkant, om geen verwarring te zaaien.





**Figuur 3.3:** Derde prototypevoorstel met (a) het welkomstscherf, (b) keuzemenu voor de juiste trein, (c) het scherm met keuzemogelijkheid voor rijtuig (d) navigatie naar de perronplek aan zijkant, (e), (f) en (g) kleinere minischermen die geopend staan en (h) het scherm als men op de juiste plek is.

## 3.4 Testapplicatie

Als startapplicatie, hebben we eerst een eenvoudige applicatie gemaakt in React Native. Deze applicatie is later (met aanpassingen) als basis gebruikt voor onze finale applicatie.

### 3.4.1 Systeemarchitectuur van de testapplicatie

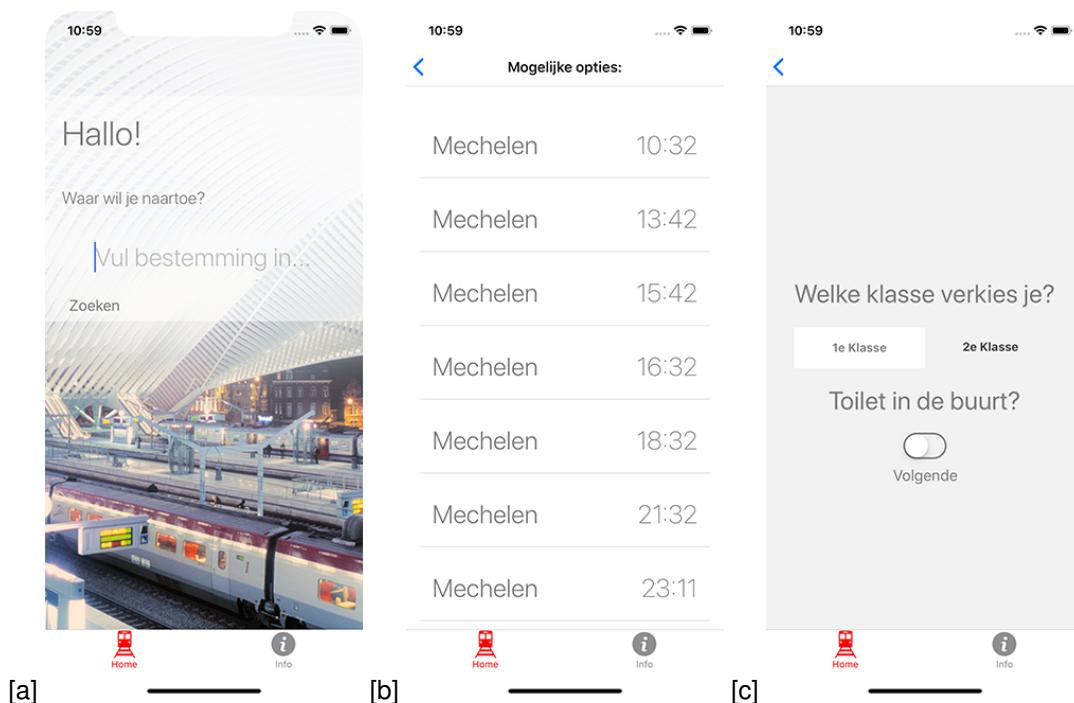
Net zoals de uiteindelijke applicatie bestaat deze applicatie uit een client en server-deel. Op de server bevinden zich de PHP bestanden en de MySQL databank. De databank bevat de volgende dummy data:

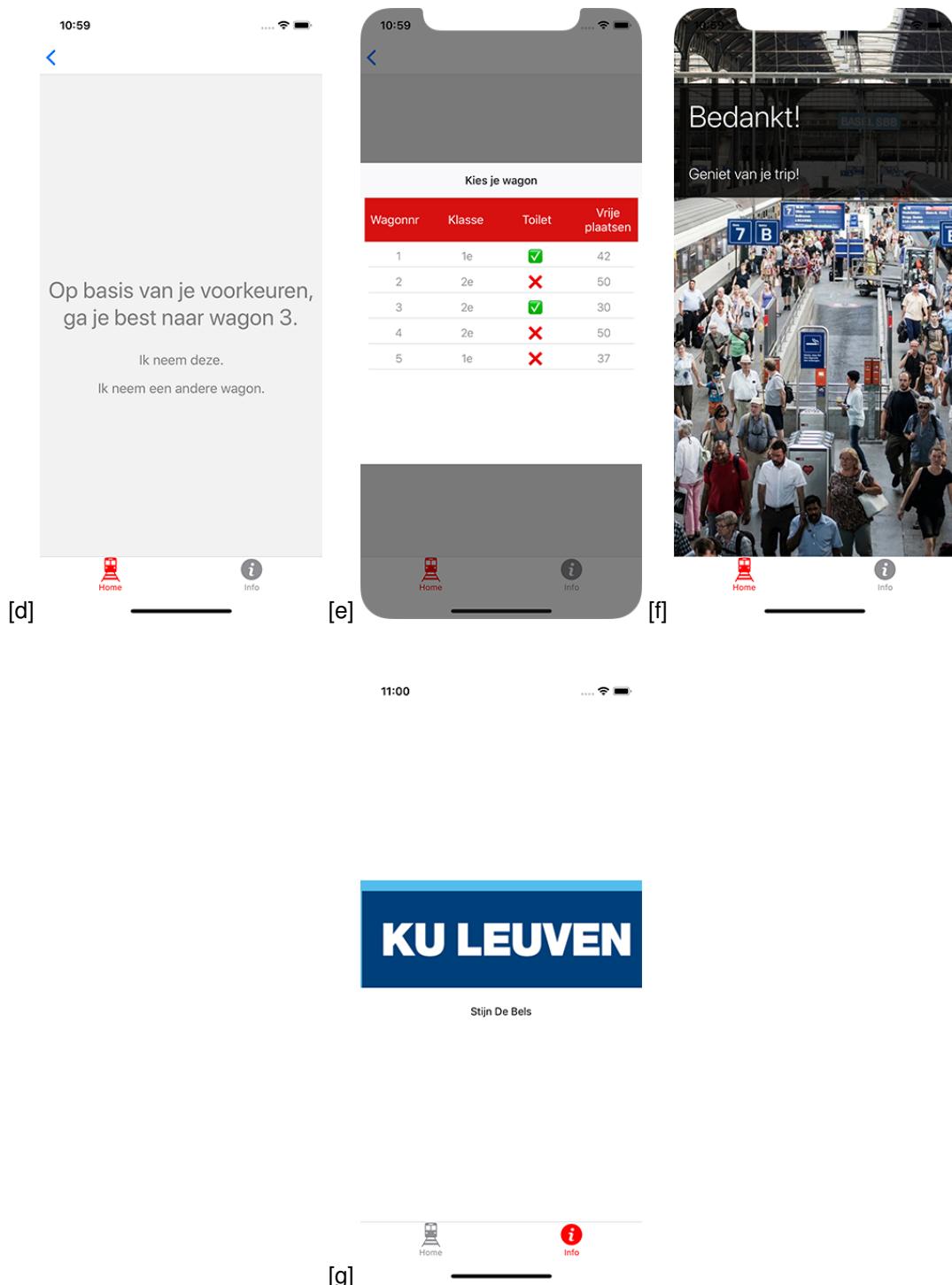
- **'trains' tabel**, dat voor alle treinen het TrainID, de bestemming, het uur en het aantal rijtuigen weergeeft.
- **'train' tabel**, dat per rijtuig weergeeft welke opties deze heeft, (1e klasse, Toilet, plaats voor personen met handicap) en hoeveel plaatsen er vrij zijn.

De applicatie aan clientzijde bestaat uit 6 schermen:

- **HomeScreen:** Hier typ je de eindbestemming in, visueel gezien is dit integraal overgenomen. Bij het klikken op zoeken zal info van mogelijke treinen uit de 'trains' tabel worden opgevraagd.

- **ChoiceScreen:** Hier kies je welke trein je neemt, visueel gezien werd dit bijna integraal overgenomen. Bij het selecteren zal alle info over de trein uit de 'train' tabel van de databank worden opgehaald.
- **PreferenceScreen:** bevat vragen waarin de gebruiker kan aanduiden welke opties hij wilt. Op basis hiervan zal het ideale rijtuig worden opgehaald.
- **WagonScreen:** Op basis van voorkeuren en de bezettingsgraad in een rijtuig, wordt het ideale rijtuig geselecteerd. De gebruiker kan echter manueel deze voorkeur wijzigen. Bij bevestiging wordt er connectie met de database gemaakt en wordt in de 'train' tabel het aantal vrije plaatsen met 1 verminderd. De 'train' tabel wordt hier dus gewijzigd.
- **EndScreen:** Scherm dat wordt getoond nadat de gebruiker beslist heeft welk rijtuig hij gaat nemen.
- **InfoScreen:** bevat het KU Leuven logo, werd visueel bijna integraal overgenomen.





**Figuur 3.4:** Testapplicatie mat (a) het HomeScreen, (b) het ChoiceScreen, (c) het PreferenceScreen, (d) het Wagonscreen, (e) het manueel wijzigen van de verkozen rijtuig in het WagonScreen, (f) het EndScreen en (g) het InfoScreen.

Vooral van de beginschermen zijn uiteindelijk veel elementen in de uiteindelijke applicatie gebleven. Zo wordt ook op dezelfde manier data uit JSON bestanden gehaald. Echter wordt er niet gekeken

naar de bezettingsgraad van de rijtuigen in de finale applicatie, omdat deze data bij de NMBS niet beschikbaar is. Dit wordt hier vervangen door het rijtuig te selecteren op basis van het rijtuig dat zich het dichtst bij de huidige locatie bevindt. Ook hier kan de gebruiker de rijtuigen manueel veranderen. In onze finale applicatie zullen we ook gebruikmaken van data die we verkrijgen van de NMBS.

### 3.5 Conclusie

In dit hoofdstuk deden we een kleine conceptstudie voordat we begonnen met het ontwikkelen van onze applicatie. We maakten in samenspraak met de NMBS enkele prototypes om zo onze verwachtingen gelijk te stellen. Vervolgens maakten we een startapplicatie om de React Native programmeertaal te leren.

# **Hoofdstuk 4**

## **Specificaties**

In dit hoofdstuk bekijken we de algemene architectuur van onze applicatie. We bekijken ook de navigatiestructuur en lijsten de verschillende custom components, modal screens en veel gebruikte variabelen en states op. Met deze informatie kunnen we in het volgende hoofdstuk al onze interessante componenten gedetailleerder gaan specifiëren.

De applicatie wordt in eerste instantie gemaakt voor Android, wegens licentieredenen. Door het gebruik van het expo platform en native-specifieke API's te mijden is onze applicatie ook iOS compatibel.

### **4.1 Vereisten van de applicatie**

Om een goede werking van de applicatie te verkrijgen, moeten er enkele eisen worden gesteld:

- De locatie moet bepaald worden en moet een bepaalde nauwkeurigheid hebben.
- De smartphone moet over een internetconnectie beschikken.
- De applicatie moet kunnen gebruikt worden door gebruikers van elke leeftijd, educatieve achtergrond en verschillende fysieke en mentale condities. Daarom werd de applicatie ontwikkeld met het KISS-principe (Keep it simple and stupid) in het achterhoofd.
- De applicatie moet makkelijk aanpasbaar zijn om uitbreidingen en extra functies toe te voegen. Dit met behulp van variabelen, structuur, aparte functies en uitleg bij delen van de codering.

## 4.2 Systeemarchitectuur van het finale model

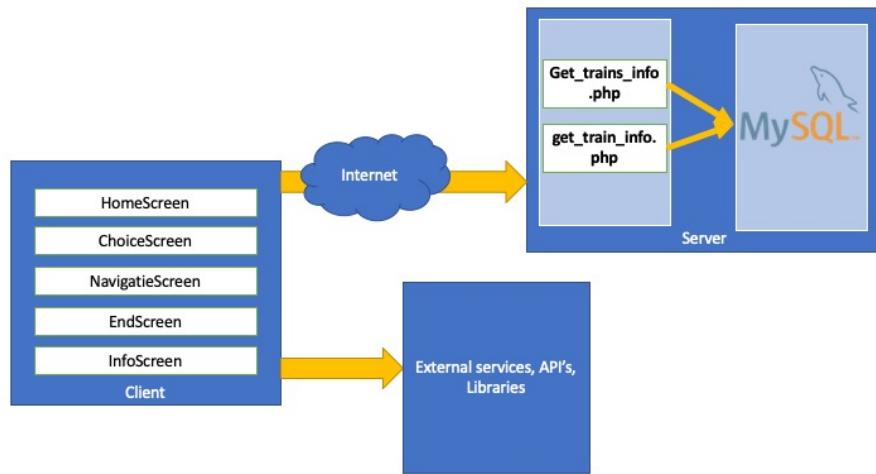
React Native bevat een architectuur in de vorm van componenten. Elk component is een op zichzelf staand stuk code dat de beschrijving van een visueel element bevat. Dit omvat de huidige toestand van variabelen en states (variabelen die een rerendering triggeren), attributen en de rekenkundige bewerkingen erop. Componenten kunnen componenten bevatten die op hun beurt dan ook weer componenten kunnen bevatten,... Ook schermen zijn dus componenten. Een overzicht van de hoofdcomponenten die wij in onze applicatie gebruiken:

- App.JS
  - HomeScreen:
    - \* DestinationScreen
    - \* ChoiceScreen
    - \* NavigatieScreen:
      - Train: Custom component
      - meldingVerkeerdePerron: modal screen
    - \* EndScreen
  - InfoScreen

Alle componenten erven over van React.Component en hebben al zeker een render methode. Deze methode zorgt voor de visuele voorstelling. Uit de render methode kan bijvoorbeeld een <View> en <Text> component verkregen worden. Meerdere componenten komen altijd samen in één component: globaal in App.js, maar ook visuele componenten als <Text> en <Image> zijn ingecapsuleerd in een <View> component. [28]

Ondanks de filosofie dat elke component een op zichzelf staand stuk code bevat, zullen we in onze code enkele grote constantes met stijlingselementen in een aparte Javascript file schrijven. Dit wordt gedaan om meer structuur te brengen in de code. Ook worden globale functies gemaakt waar verschillende klassen gebruik van kunnen maken (zie bijlage A.1.2). Indien data moet worden doorgegeven, zal dit als argument meegegeven worden. Er wordt geen globale opslag gecreëerd omdat data enkel moet worden doorgegeven aan het opeenvolgende scherm.

### 4.2.1 Globale structuur



**Figuur 4.1:** Globale structuur van de applicatie.

Zoals te zien in figuur 4.1 kan de applicatie onderverdeeld worden in 2 delen:

- **Client-side:** bevat al het offline gedeelte dat op de smartphone staat, dit zijn alle visuele elementen en berekeningen. Deze maakt ook gebruik van externe services, API's en libraries.
- **Server-side:** bevat alle data rondom de treinen, stations en perrons.

Via PHP bestanden (die op de publieke webserver staan) wordt een connectie gemaakt tussen de MySQL databank en de offline applicatie. Dit wordt verder besproken in sectie 5.1.

### 4.2.2 Navigatiestructuur

Om de verschillende schermen te kunnen bereiken, wordt gebruik gemaakt van een bepaalde navigatiestructuur. Dit wordt gedaan aan de hand van de 'react-navigation' library, die we in onze applicatie importeren. We maken gebruik van TabNavigator en StackNavigator.

#### 4.2.2.1 TabNavigator

Vanuit de App.js file zijn er 2 schermen te bereiken. Het HomeScreen (dat standaard wordt weergegeven) en het InfoScreen. Beiden zijn bereikbaar vanuit de bottomTabNavigator. Een horizontale balk die zich onderaan het scherm bevindt en hier aanwezig blijft gedurende de gehele periode dat de applicatie actief is. Volgende schermen zijn hiermee te bereiken:

- **HomeScreen:** bevat StackNavigator
- **InfoScreen:** bevat de logo's van het meewerkende bedrijf en de meewerkende universiteit, alsook de naam van de ontwikkelaar.

#### 4.2.2.2 StackNavigator

De StackNavigator zorgt voor een overgang tussen de schermen waarbij elk scherm bovenop de stack geplaatst wordt. Het zorgt ervoor dat men gemakkelijk terug kan gaan naar de vorige schermen, gezien deze zich nog op de stack bevindt. Omdat onze applicatie vooral een verzameling is van opeenvolgende opdrachten (kies bestemming - selecteer trein - kies rijtuig - navigeer naar rijtuig), zullen we naar de meeste schermen navigeren met behulp van de StackNavigator. Dit zijn volgende schermen:

- **DestinationScreen:** om je eindbestemming in te vullen
- **ChoiceScreen:** om trein te kiezen
- **NavigatieScreen:** om te navigeren naar een gekozen rijtuig
- **EndScreen:** om te vermelden dat je correct staat

#### 4.2.2.3 Modal screen

Naast de in navigatie geïmplementeerde schermen zijn er ook nog modal screens. Deze komen te voorschijn wanneer ze getriggerd worden door events. Geïmplementeerde modal screens zijn:

- **meldingVerkeerdePerron:** Deze wordt getriggerd wanneer blijkt dat de huidige locatie van de reiziger zich niet op het perron bevindt.
- **meerInfoScreen:** Geeft meer info meer over de trein waaronder de beschikbare faciliteiten en aantal rijtuigen.

#### 4.2.3 Custom Components

Naast de standaard geïmporteerde componenten als <View>, is er ook de mogelijkheid om componenten aan te maken. Dit kan interessant zijn om structuur aan te brengen of wanneer men eenzelfde component meerdere malen nodig heeft. De custom component die in deze applicatie werd aangemaakt is:

- **Train:** toont de gehele trein in een opeenvolging van rijtuigen, zie sectie 5.5.4.1.

#### 4.2.4 Expo platform

We kunnen onze applicatie gemakkelijk voor zowel iOS als Android maken dankzij het expo platform. Dit is een gratis en opensource platform dat rondom React Native gebouwd is en ervoor zorgt dat iOS en Android native tools niet geïnstalleerd moeten worden.

#### 4.2.5 Veelgebruikte variabelen en states

In de code en delen van de code die in volgende hoofdstukken worden beschreven, zijn er enkele variabelen en states vermeld die misschien meer uitleg vragen. Om deze reden lijsten we hier enkele veel gebruikte variabelen en states op:

- **totalHLengthTillEndNavigation:** Dit is het eindpunt van onze navigatie, dit is meestal de deur van het rijtuig waar we moeten instappen.
- **CurLocPixelX:** De x positie in pixels van de huidige locatie ten opzichte van de linkerbovenhoek van het perron.
- **CurLocPixelY:** De y positie in pixels van de huidige locatie ten opzichte van de linkerbovenhoek van het perron.
- **this.state.location.coords.longitude:** De lengtegraad van de huidige locatie in geografische coördinaten.
- **this.state.location.coords.latitude:** De breedtegraad van de huidige locatie in geografische coördinaten.
- **this.state.treinGeselecteerd:** Geeft aan of in de huidige toestand een trein is geslecteerd of niet.
- **this.state.listPlatformComponents:** bevat alle objecten die zich op het perron bevinden, informatie verkregen uit de databank. Properties zijn Longitude, Latitude, Type, Track.
- **this.state.listPlatformComponentsTypes:** bevat alle soorten objecten waarvan afmetingen zijn, dit is ook informatie verkregen uit de databank. Properties zijn Type, Width, Length.
- **ricoLengte:** richtingscoëfficiënt van de perronlengtes in geografische coördinaten.
- **ricoBreedte:** richtingscoëfficiënt van de perronbreedtes in geografische coördinaten.
- **this.state.listTrack [0][”LB-LONG”]:** De lengtegraad van de linkerbovenhoek van het perron in geografische coördinaten.

- **this.state.listTrack [0][“LB-LAT”]**: De breedtegraad van de linkerbovenhoek van het perron in geografische coördinaten.
- **this.state.listTrack [0][“RB-LONG”]**: De lengtegraad van de rechterbovenhoek van het perron in geografische coördinaten.
- **this.state.listTrack [0][“RB-LAT”]**: De breedtegraad van de rechterbovenhoek van het perron in geografische coördinaten.
- **this.state.listTrack [0][“RO-LONG”]**: De lengtegraad van de rechteronderhoek van het perron in geografische coördinaten.
- **this.state.listTrack [0][“RO-LAT”]**: De breedtegraad van de rechteronderhoek van het perron in geografische coördinaten.
- **this.state.listTrack [0][“LO-LONG”]**: De lengtegraad van de linkeronderhoek van het perron in geografische coördinaten.
- **this.state.listTrack [0][“LO-LAT”]**: De breedtegraad van de linkeronderhoek van het perron in geografische coördinaten.

#### 4.2.6 Vooraf gedefinieerde constanten en veelgebruikte functies

Doorheen de code wordt soms meerdere malen gebruik gemaakt van eenzelfde waarde. Denk bijvoorbeeld aan de omzetting van meter naar een bepaald aantal pixels op het scherm. Deze waarden zijn verzameld in één bestand. Ook voor code-efficiëntie en herbruikbaarheid worden hier enkele objecten met verschillende afbeeldingen in, aan toegevoegd. De constanten staan in bijlage A.1.1 beschreven. Ook werden enkele veelgebruikte functies in een apart bestand gezet (bijlage A.1.2).

### 4.3 Conclusie

In dit hoofdstuk legden we de algemene architectuur van onze applicatie vast. Om te navigeren gebruiken we een Stack- en TabNavigator. Ook kiezen we voor het expo platform om zo makkelijk een applicatie voor zowel iOS als Android te maken. Veel gebruikte constanten en functies worden gebundeld in een apart bestand.

# **Hoofdstuk 5**

## **Uitwerking**

Nu we de specificaties, de architectuur en het voorbereidend werk van onze applicatie hebben bestudeerd, zullen we onze finale applicatie in meer detail, component per component, beschrijven. In het volgende hoofdstuk zullen we eerst het online deel bespreken, vervolgens het offline deel dat scherm per scherm wordt overlopen, met subsecties voor de onderdelen en componenten.

### **5.1 Databank en PHP connectie**

Naast de offline applicatie is er ook een online server deel. Dit bestaat uit een databank en PHP code. Via de PHP code communiceert de applicatie met de databank en worden er gegevens opgevraagd.

#### **5.1.1 Databank**

De databank is een online MySQL-server die bereikbaar is onder een vroeger aangekochte domeinnaam. Met behulp van een online JSON convertor wordt JSON data uit de NMBS databank omgezet in MySQL waardoor een dummy databank tot stand komt. Uit de NMBS databank werden 3 tabellen gevormd die volgende informatie bevatten:

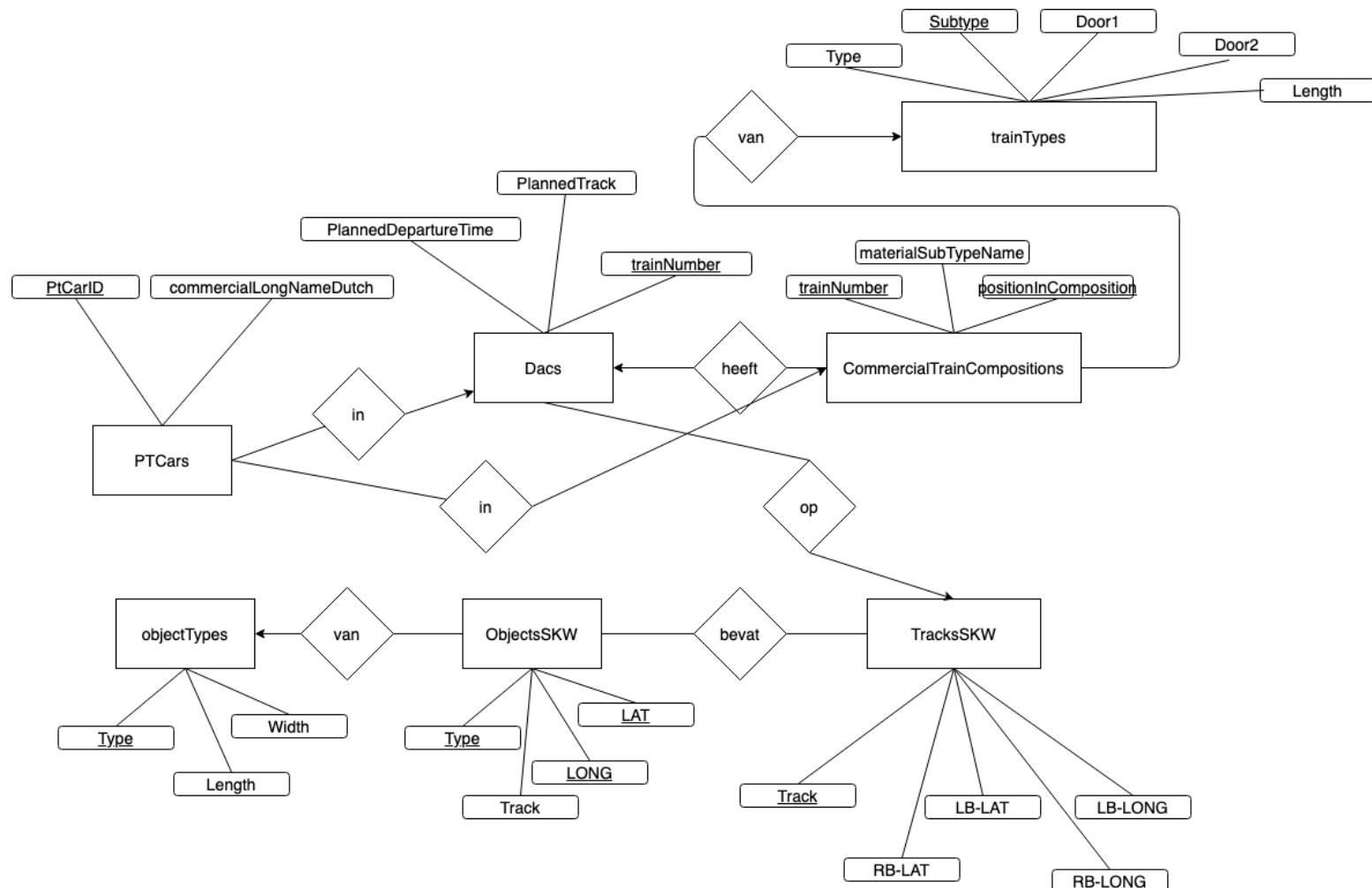
- **Dacs:** Bevat alle uren, treinen,... naar een bepaalde bestemming. De 'Dacs' tabel uit onze databank bevat alle uren en treinen naar Mechelen. Deze data werd uiteindelijk gebruikt in combinatie met de perrons van Sint-Katelijne-Waver. Alle treinsporen werden aangepast naar een spoor tussen 1 en 4, gezien dit het aantal sporen zijn dat het station van Sint-Katelijne-Waver bevat.
- **CommercialTrainCompositions:** Bevat per trein (die elk een eigen trainID heeft) inhoud over de samenstelling van de rijtuigen en de kenmerken hiervan.

- **PTCars:** Elke treinstation heeft een bepaald ptCarID, waarmee een station overeenkomt. In de andere tabellen wordt enkel verwezen naar deze ptCarID. Hierin staat welk ptCarID met welke stationsnaam overeenkomt

Wegens gebrek aan data werden volgende tabellen zelf aangemaakt:

- **TracksSKW:** waarin per spoor de geografische coördinaten van de linkerbovenhoek, rechterbovenhoek, linkeronderhoek en rechteronderhoek te vinden zijn. Deze 4 coördinaten vervullen een belangrijke rol in de hele applicatie. Ze zorgen namelijk dat lokalisatie, de positie van treinen, en de navigatie correct worden weergegeven.
- **trainTypes:** bevat per type voertuig een oplijsting van alle afstanden tot de deuren en de lengtes van de rijtuigen.
- **ObjectsSKW:** bevat per perron een oplijsting van alle objecten (trap, afdak,...) die zich op het perron bevinden.
- **objectTypes:** geeft de afmetingen van bepaalde types objecten weer.

Onderstaand ER diagram geeft de relatie tussen de tabellen weer.



**Figuur 5.1:** ER diagram die de connectie tussen de verschillende tabellen voorstelt. Per tabel zijn enkele interessante attributen hiervan weergegeven.

### 5.1.2 PHP

Via de PHP bestanden gebeurt de connectie tussen de databank en de applicatie. In de PHP code (te vinden in bijlage A.2) wordt een connectie met de databank gemaakt. Vervolgens wordt in het PHP bestand de data, die door de applicatie in JSON-codering hiernaartoe is verzonden, gedecodeerd en in variabelen opgeslagen. Met behulp van deze variabelen wordt een SQL query opgebouwd en uitgevoerd. De SQL queries die uitgevoerd worden, zijn dus dynamisch en worden vlak voor het versturen naar de databank samengesteld met de variabelen. De resulterende data uit de MySQL databank wordt in JSON geformateerd en teruggestuurd naar de applicatie. Volgende PHP bestanden worden gebruikt:

- **DBConfig.php:** Bevat de hostname, databasename, hostuser en hostpassword van de betreffende databank om verbinding te maken. Dit bestand wordt in elk ander PHP bestand ingevoegd.
- **get\_trains\_info.php:** Krijgt het gewenste eindstation vanuit de applicatie toegestuurd, en gaat met deze info in de 'PTCars' tabel zoeken naar het ptCarID waarmee dit station overeenkomt. Vervolgens gaat het in de 'Dacs' tabel zoeken naar alle treinen waarvan de bestemming gelijk is aan het ptCarID. Het stuurt het trainID, spoor en vertrekuur terug in JSON-formaat naar de applicatie.
- **get\_train\_info.php:** Na het selecteren van de gewenste trein, wordt het spoor en treinnummer aan dit PHP bestand doorgegeven. Uit de 'TrackSKW' tabel worden vervolgens de geografische coördinaten van het perron gehaald, uit de 'CommercialTrainCompositions' tabel worden gegevens over de rijtuigen gehaald, en indien beschikbaar wordt uit de 'trainTypes' tabel informatie over de afstanden en lengtes gehaald. Indien dit niet beschikbaar is, wordt de data over de lengtes uit de 'CommercialTrainCompositions' tabel gehaald. Uit de ObjectsSKW wordt data gehaald van de objecten die zich op het perron bevinden en ook de objectTypes worden opgehaald. Al deze data wordt teruggestuurd in JSON-formaat naar de applicatie.

## 5.2 Startscherf (App.js)

App.js is het vertrekende punt van de applicatie, ook te interpreteren als de main van de mobiele applicatie. Moeten we een kleine applicatie maken, zouden we alles hierin kunnen schrijven. We willen echter meerdere schermen en het overzicht kunnen bewaren, dus plaatsen we hier de react-navigation bibliotheek. Deze bibliotheek bevat onder meer een createAppContainer, waarmee de top-level navigator gelinkt wordt met de applicatie omgeving. De afgeleide bibliotheek react-navigation-tabs bevat createTabBottomContainer, waarmee de onderste menubalk van de smartphone wordt gecreëerd.

### 5.3 Invoer van bestemming (HomeScreen.js)

In HomeScreen.js staat de code van het eerste echte scherm. Hier geef je de locatie in waar je naartoe wil en klik je op zoeken waardoor je navigeert naar het volgende scherm. Wat de gebruiker niet ziet, is dat wanneer er op zoeken geklikt wordt, er op de achtergrond de functie getDataFromServer (te zien in listing 5.1) wordt opgeroepen, waarbij een connectie met get\_trains\_info.php wordt aangelegd en data wordt naartoe gestuurd. Het JSON antwoord (responseJSON) wordt onmiddellijk omgezet naar een array bestaande uit objecten. Deze wordt dan als argument meegegeven met de functie oproep om te navigeren naar het volgende scherm.

```
1 receivedJSON (possibleTrains)
2 {
3     possibleTrains = JSON.parse(possibleTrains);
4     let possibleTrainsAr=JSONtodata(possibleTrains) // omzetting naar array, te
        zien in bijlage A.1.2 op lijnen 98 - 118
5     this.setState({ listData : possibleTrainsAr });
6 }
7 getDataFromServer = () =>{
8     const { TextInputDestination } = this.state ;
9     fetch('http://'+ip+'/get_trains_info.php', { //IP is een constante dat ons
        webadres voorstelt
10    method: 'POST',
11    headers: {
12        'Accept': 'application/json',
13        'Content-Type': 'application/json',
14    },
15    body: JSON.stringify({
16        Destination: TextInputDestination,
17    })
18 }).then((response) => response.json())
19
20 .then((responseJson) => {
21     if((responseJson).length==2){
22         Alert.alert("Er zijn geen routes gevonden met deze bestemming.");
23     }
24     else
25     {
26         this.receivedJSON(responseJson);
27         this.props.navigation.navigate("ChoiceScreen", {dataLijst: this.state.
            listData}); //navigeren naar het volgende scherm met de verkregen
            data als argument meegegeven
28     }
29 }).catch((error) => {
30     console.error(error);
31 });
```

32

```
}
```

**Listing 5.1:** GetDataFromServer en receivedJSON functie, deze worden opgeroepen bij het versturen van data naar het PHP bestand.

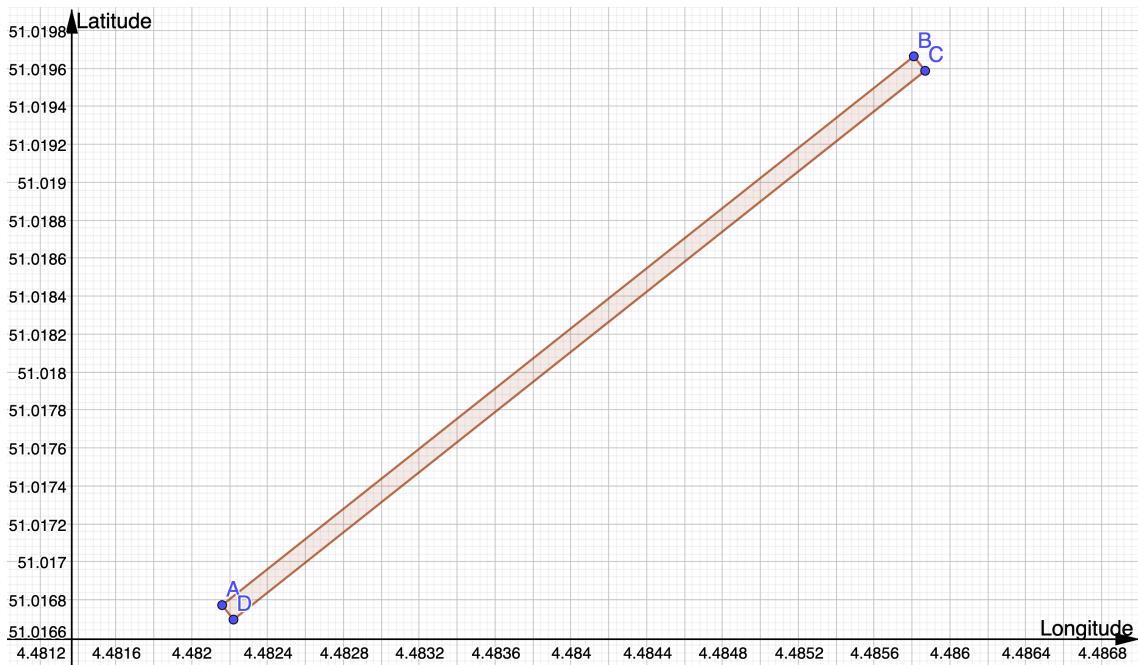
## 5.4 Keuzelijst van treinen (ChoiceScreen.js)

Het ChoiceScreen toont de beschikbare opties voor de gewenste treinbestemming. De data (treinen die stoppen aan de geselecteerde bestemming) die met het vorige scherm werd meegegeven als argument, wordt als een state in de constructor toegekend. De state wordt als data aan een FlatList gekoppeld. Deze lijst rendert vervolgens deze gehele data op het scherm. De data bevat alle uren en sporen waaruit je een trein kan kiezen. Bij het klikken op een element zal er het get\_train\_info.php bestand opgevraagd worden. Het antwoord wordt op een gelijkaardige manier als listing 5.1 behandeld: het JSON-formaat wordt omgezet in een objectarray en meegegeven naar het volgende scherm.

## 5.5 Trein- en navigatiescherm (NavigatieScreen.js)

Het NavigatieScreen is waar de kern van de applicatie in deze masterproef zit: nadat de juiste trein geselecteerd wordt, zal hier de treinsamenstelling worden gevisualiseerd. Door het juiste rijtuig aan te duiden zal het perron tevoorschijn komen. Hierop wordt de huidige locatie van de gebruiker aangeduid, samen met een navigatie naar de plek waar men moet opstappen om op het rijtuig te geraken. Bij het arriveren aan de juiste locatie, zal er genavigeerd worden naar het eindscherm.

### 5.5.1 Geografisch coördinatenstelsel



**Figuur 5.2:** Geografisch coördinatenstelsel met een voorbeeldspoor.

Voor enkele van onze berekeningen, zullen we gebruikmaken van een xy-assenstelsel waarbij x de lengtegraad (longitude) en y de breedtegraad (latitude) coördinaat voorstelt. We gaan er dus vanuit dat een verplaatsing op de aarde lineair is. Dit is natuurlijk niet waar, wetende dat onze aarde bolvormig is, echter kunnen we dit zo toepassen gezien afstanden als de lengte en breedte van het perron ten opzichte van de gehele aarde relatief klein is. Andere benaderingen zijn te vinden in het perron: we gaan ervan uit dat dit perfect rechthoekig is. Dit is echter in de meeste perrons niet geheel correct, maar voor ons dummy model moeilijk implementeerbaar. Uit figuur 5.2 kunnen we opmaken dat we met twee richtingscoëfficiënten genoeg hebben, gezien we twee paar parallelle lijnen hebben.

### 5.5.2 Visualisatie van het perron

Om de geografische coördinaten van onze huidige locatie correct weer te geven op het perron, zullen we enkele berekeningen moeten doen. Dit omdat we niet werken met Google Maps-achtige API's, waar de positie automatisch juist gerenderd wordt op een kaart. We maken hier gebruik van een rechthoek dat ons perron voorstelt. De berekening van de correcte lengte en breedte zullen we in de volgende secties bespreken.

### 5.5.2.1 Grootteberekening van het perron

De grootteberekening van ons perron gebeurt met behulp van de Haversine formule. De Haversine formule berekent de korste afstand tussen 2 punten (in geografische coördinaten gegeven) die op een bol liggen. De berekening gaat wiskundig als volgt:

$$\Delta lon = lon_1 - lon_2 \quad (5.1)$$

$$\Delta lat = lat_1 - lat_2 \quad (5.2)$$

$$a = \sin^2 \frac{\Delta lat}{2} + \cos lat_1 * \cos lat_2 * \sin^2 \frac{\Delta lon}{2} \quad (5.3)$$

$$c = 2 * \text{atan2}(\sqrt{a}, \sqrt{1-a}) \quad (5.4)$$

$$d = R * c \quad (5.5)$$

De aardstraal (R) is ongeveer 6371 km, dit is een benadering, gezien de aarde geen perfecte bol is maar voldoende voor onze berekeningen. Door gebruik te maken van de geografische coördinaten van de linkerbovenhoek, rechterbovenhoek, linkeronderhoek en rechteronderhoek van het perron kunnen we de totale afstand van ons perron bepalen.

Omdat het perron niet wordt weergegeven in meter, maar in pixels bepalen we een verhouding van 27 meter is 500 pixels op het scherm. We leggen dit als constante vast in het bestand constants.js (zie lijn 1 in bijlage A.1.1). Op deze manier kunnen we ook de rijtuigen van de trein (die ook in meter in de databank staan) relatief correct weergeven ten opzichte van het perron.

### 5.5.2.2 Objecten op het perron



**Figuur 5.3:** Iconen van verschillende soorten objecten: (a) is een icoon voor een afdak, (b) een icoon voor trappen.

Meestal bevinden zich op het perron verscheidene objecten zoals een trap, een afdak om onder te schuilen,... Het is interessant om deze elementen mee te renderen op het perron, zodat de gebruiker aan de hand van meerdere referenties een beeld kan hebben waar hij staat. Alle objecten die op het perron staan, worden samen met een geografische coördinaat uit de 'ObjectsSKW' tabel gehaald. De geografische coördinaat is de linkerbovenhoek van het object.

Eerst wordt er gecontroleerd of het type zich in de 'objectTypes' tabel bevindt. Indien wel, haalt het hier de lengtes uit en maakt een `<View>` object aan op basis van de berekende lengtes in pixels (dit wordt uitgelegd in sectie 5.5.3.4). Door argumenten als breedte, lengte en beginpunt met de `<View>` tag mee te geven, wordt aan de hand van de `<View>` tag een rechthoek op het perron

getekend. In het midden van deze rechthoek wordt dan een icoon geplaatst. Dit is een afbeelding van wat het object voorstelt (zie figuur: 5.3). De perronObjecten worden in de objectPixels array bijgehouden om ze later te kunnen gebruiken voor de navigatiebepaling. In een 'PerronObject' bevinden zich de x en y pixels van alle hoeken van dit object ten opzichte van de linkerbovenhoek van het perron.

```

1 function PerronObject(lbpixelx,lbpixely,rbpixelx,rbpixely,lopixelx,lopixely,
2   ropixelx,ropixely) {
3   this.LBPixelX = lbpixelx;
4   this.LBPixelY = lbpixely;
5   this.RBPixelX = rbpixelx;
6   this.RBPixelY = rbpixely;
7   this.LOPixelX = lopixelx;
8   this.LOPixelY = lopixely;
9   this.ROPixelX = ropixelx;
10  this.ROPixelY = ropixely;
11 }
```

**Listing 5.2:** PerronObject met alle properties gedeclareerd

Indien het object zich niet in de databank bevindt wordt enkel op de geografische coördinaat een icoon geplaatst. Volgende code geeft de implementatie weer. Perronobjecten zijn statisch en dus onveranderlijk. Deze functie wordt dan ook maar 1 keer opgeroepen.

```

1 perronObjecten()
2 {
3   // Ga alle objecten af van het perron
4   for (let i=0; i < (this.state.listPlatformComponents).length; i++)
5   {
6     // Ga alle soorten objecten af
7     for (let j=0;j < (this.state.listPlatformComponentsTypes).length; j++)
8     {
9       //Zit dit type object in de databank van met alle types?
10      if(this.state.listPlatformComponentsTypes[j]["Type"]==this.state.
11          listPlatformComponents[i]["Type"])
12      {
13        // Geografische coordinaat van het object (linkerbovenhoek) omzetten
14        // naar aantal pixels vanaf de linkerbovenhoek van het perron
15        [Obj1X, Obj1Y]=this.geoToPixelBerekening(this.state.
16          listPlatformComponents[i]["LONG"],this.state.listPlatformComponents
17          [i]["LAT"]);
18        // Breedte in pixels van het object berekenen, meternrPixelOmzetting is
19        // een constante
20        let widthObj=meterNrPixelOmzetting*parseFloat(this.state.
21          listPlatformComponentsTypes[j]["Width"])
22        // Hoogte in pixels van het object berekenen, meternrPixelOmzetting is
23        // een constante
24      }
25    }
26  }
27}
```

```

17 let heightObj=meterNrPixelOmzetting*parseFloat(this.state.
    listPlatformComponentsTypes[j]["Length"])
18 let objAfbeelding;
19 //Bestaat een icoon van dit type object
20 if (objImages[this.state.listPlatformComponentsTypes[j]["Type"]])
21 {
22     //Maak een afbeelding aan van dit icoon
23     objAfbeelding=<Image source={objImages[this.state.
        listPlatformComponentsTypes[j]["Type"]]} style={{flex: 1,
        resizeMode: 'contain' }} />;
24 }
25 //Zet het icoon in het midden van de View-tag, wat een rechthoek van de
26     grootte van het object is
27 objObject[i]=<View style={{width:widthObj, backgroundColor: 'grey',
    height: heightObj,position: 'absolute',left: Obj1X, top:Obj1Y,
    alignItems:'center', justifyContent:'center'}}>{objAfbeelding}</
    View>;
28 //in PerronObject vinden zich de x en y pixels van alle hoeken van dit
    object ten opzichte van de linkerbovenhoek van het perron, voor de
    navigatiebepaling later.
29 objectPixels[nr]=new PerronObject(Obj1X,Obj1Y,Obj1X+widthObj,Obj1Y,
    Obj1X,Obj1Y+heightObj,Obj1X+widthObj,Obj1Y+heightObj);
30 nr++;
31 }
32 else
33 {
34     //Geen afstanden bekend van het object? Toon dan gewoon een icoon
35     [Obj1X, Obj1Y]=this.geoToPixelBerekening(this.state.
        listPlatformComponents[i]["LONG"],this.state.listPlatformComponents
        [i]["LAT"]);
36     //Bestaat een icoon van dit type object
37     if (objImages[this.state.listPlatformComponents[i]["Type"]])
38     {
39         //Maak een afbeelding aan van dit icoon
40         objObject[i]=<Image source={objImages[this.state.
            listPlatformComponents[i]["Type"]]} style={{position: 'absolute',
            left:Obj1X, top:Obj1Y, flex: 1,resizeMode: 'contain' }} />;
41     }
42 }
43 }
44 return [objObject] //Bevat alle objecten in tag vormen, dit moet gewoon nog
    opgeroepen worden in de render() om dit op het scherm weer te geven.
45 }

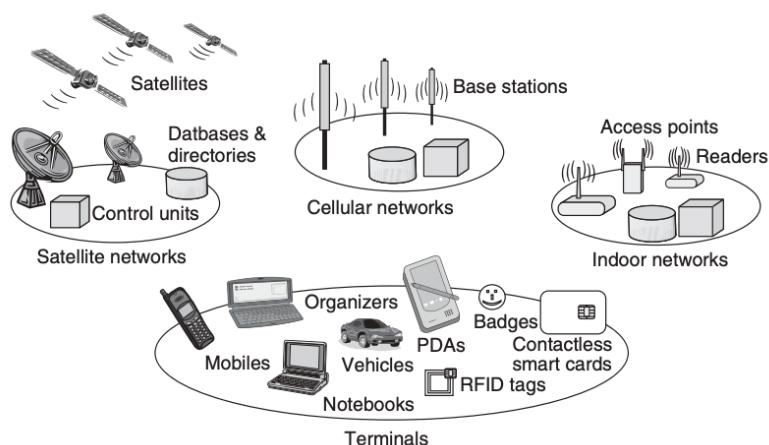
```

**Listing 5.3:** Initialisatie van perronobjecten

### 5.5.3 Locatiebepaling en visualisatie

De locatiebepaling is heel belangrijk, zo moet de nauwkeurigheid goed genoeg zijn om een correct resultaat te verkrijgen. Ook moeten alle wiskundige berekeningen correct zijn: zo moet er correct berekend worden waar de huidige locatie wordt gevisualiseerd op het perron, aangezien we niet met een Google Maps API werken. In volgende secties zullen we dan ook bespreken hoe locatiebepaling uitgevoerd wordt, en hoe deze locatie dan wordt gevisualiseerd op het scherm.

#### 5.5.3.1 Locatiebepaling op een smartphone



**Figuur 5.4:** Technologieën waarmee locatiebepaling wordt gedaan [1].

Locatiebepaling wordt op een smartphone met 3 technologieën geïmplementeerd: CELL ID, WLAN en GPS. In figuur 5.4 worden de mogelijke infrastructuren gevisualiseerd alsook de terminals (de toestellen waarvan je de locatie wil bepalen). Infrastructuren waarmee onze smartphone werkt zijn:

- **Satelliet infrastructuur:** Meest bekende voorbeeld hiervan is de bij smartphone gebruikte GPS technologie, die een wereldwijde dekking heeft en een relatieve hoge nauwkeurigheid. Satellietsignalen zijn echter heel gevoelig voor schaduweffecten en kunnen o.a. gemakkelijk geabsorbeerd worden door muren. Daarom worden deze signalen het best buitenshuis gebruikt.
- **Mobiele infrastructuur:** Dit is de infrastructuur die ook wordt gebruikt voor communicatieve doeleinden. Het is ook meestal een onnauwkeurigere methode: de draadloze signalen zijn moeilijk te schalen op een landoppervlak. Doordat operators meestal landelijke dekking hebben, werkt het ook binnen. Bij smartphone is dit de GSM technologie.

- **Binnenshuis infrastructuur:** Dit is gebaseerd op technologieën met een gelimiteerd bereik. Bij smartphones gebruikt men hier de WLAN technologie [1].

### 5.5.3.2 Locatiebepaling in React Native

In React Native wordt gebruikgemaakt van de expo-Location en expo-Permissions API's. Deze libraries werken op zowel Android en iPhone dankzij het expo platform. Voordat we connectie moeten maken, moeten we eerst toegang krijgen tot de locatievoorziening van de smartphone. Dit doen we met

```
await Permissions.askAsync(Permissions.LOCATION)
```

, waarbij de gebruiker een melding zal krijgen of hij locatietoegang wil geven aan de applicatie. Door de asynchrone await zal de applicatie niet verder renderen tot er een toestemming (of geen toestemming) wordt gegeven. Na geven van de toestemming wordt de asynchrone functie

```
await Location.getCurrentPositionAsync({});
```

opgeroepen. Waarbij er asynchroon wordt gemonitord met de functie watchPositionAsync wanneer de locatie wordt veranderd. Deze functie ziet er als volgt uit:

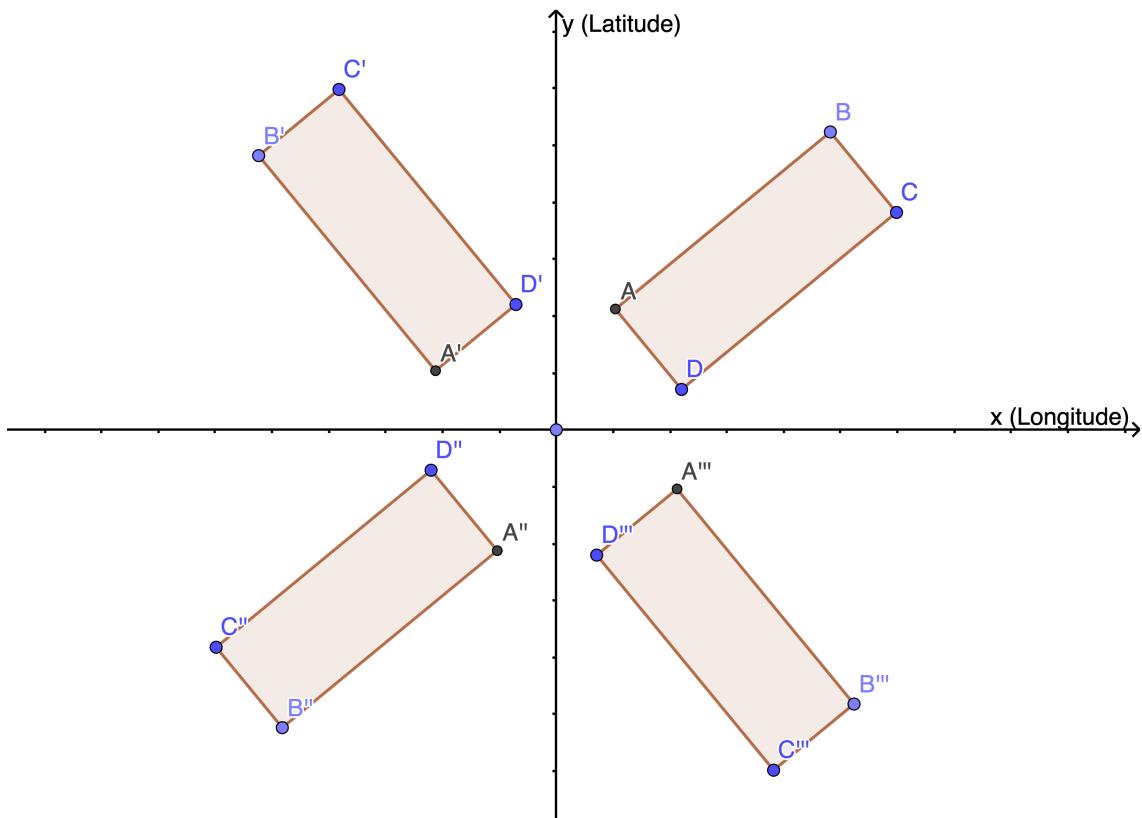
```

1 _getLocationAsync2 = async () => {
2     this.location = Location.watchPositionAsync({
3         enableHighAccuracy:true,
4         timeInterval: 10,
5         distanceInterval: 1,
6         location => {
7             this.setState({location}),
8             // Oproep van functie voor navigatieberekening:
9             this.maakNavigatie(geselecteerd,this.state.treinGeselecteerd,false)
10            if (netIngedrukt==false)
11            {
12                // Oproep van animaties
13                this.animatie()
14            }
15        });
16    };
17 }
```

**Listing 5.4:** De getLocationAsync2 functie, waarbij de functie watchPositionAsync wordt opgeroepen.

De functies die worden opgeroepen bij verandering van locatie, herberekenen onder andere de huidige locatie en wijzigen de navigatie. In de volgende secties worden deze oproepen verder verduidelijkt.

### 5.5.3.3 Randvoorwaarden voor visualisatie van de locatie



**Figuur 5.5:** Mogelijke perronoriëntaties van een perron in het geografisch coördinatenstelsel.

Omdat onze huidige locatie enkel op het perron mag verschijnen, en we de reiziger willen waarschuwen indien deze zich niet op het juiste perron bevindt, implementeren we enkele randvoorwaarden voor onze geolocatie. Dit kunnen we doen aan de hand van enkele eenvoudige wiskundige regels. Zoals te zien in figuur 5.5, heeft ons probleem vier mogelijke gevallen:

- Perron ABCD, waarbij de richtingscoëfficiënten van de perronbreedtes positief zijn, en de perronlengtes negatief.
- Perron A'B'C'D', geografisch zo gelegen dat de richtingscoëfficiënten van de perronbreedtes negatief zijn.
- Perron A''B''C''D'', dit is geografisch gezien exact hetzelfde perron als ABCD, echter komt de trein op het andere spoor aan en dus de andere kant van het perron.
- Perron A'''B'''C'''D''', ook hier geografisch gezien exact hetzelfde perron als A'B'C'D', echter komt de trein ook hier aan de andere kant van het perron aan.

Om te weten of onze huidige locatie tussen de lengtes van het perron ligt, beschouwen we de 2 lijnstukken (AB en CD in figuur 5.5) die de breedte bepalen als rechte. Dan testen we of lijnstuk AB (de kant waarlangs de trein komt) bovenaan ligt. Dit doen we door de latitude coördinaat te bekijken: als punt A in latitude hoger is dan punt D, dan ligt de treinkant bovenaan. Vanaf het moment dat we dit weten, weten we welke variabelen we kunnen gebruiken als bovenkant.

Waarom doen we dit? In de databank is voor elk spoor de linkerbovenhoek, rechterbovenhoek, linkeronderhoek en rechteronderhoek van het perron ingegeven in functie van het spoor. Dit wil zeggen dat bijvoorbeeld linkerbovenhoek ( $X_a$ ) overeenkomt met de linkerkant van de treinkant. Voor 1 perron waar 2 treinen aankomen is dan bijvoorbeeld de linkerbovenhoek (punt A) van het ene spoor gelijk aan de rechteronderhoek (punt C) van het andere spoor. Dit is best te begrijpen met figuur 5.5, waar ABCD ( $A'B'C'D'$ ) en  $A''B''C''D''$  ( $A'''B'''C'''D'''$ ) hetzelfde perron is, en  $A=C''$  ( $A'=C'''$ ),  $B=D''$  ( $B'=D'''$ ),  $C=A''$  ( $C'=A'''$ ) en  $D=B''$  ( $D'=B'''$ ).

Vanaf het moment dat we dit weten is het een kwestie van de juiste vergelijking te maken. We passen volgende wiskundige logica toe op rechthoek ABCD:

$$rico = \frac{y_B - y_A}{x_B - x_A} \quad (5.6)$$

$$y - y_B = rico * (x - x_B) \quad (5.7)$$

$$rico * x - y = rico * (x_B) - y_B \quad (5.8)$$

Wanneer we de huidige locatie in x en y invullen, moet deze in waarde tussen de boven- en onderkant van het perron liggen.

$$rico * x_B - y_B < rico * (x_{CL}) - y_{CL} \quad (5.9)$$

$$rico * x_D - y_D > rico * (x_{CL}) - y_{CL} \quad (5.10)$$

Idem voor de lengtezijden van het perron, enkel de richtingscoëfficiënt is hier anders:

$$rico = \frac{y_D - y_A}{x_D - x_A} \quad (5.11)$$

We implementeren deze vergelijkingen in listing 5.5, waar  $x_A$  overeenkomt met "this.state.listTrack[0]"["LB-LONG"]"(longitude van linkerbovenhoek in functie van een bepaald spoor) en  $x_{CL}$  met "this.state.location.coords.longitude". ParseFloat wordt altijd toegepast om data uit de databank (waar stringify functies werden op toegepast) om te vormen in floating points. Als aan deze statements voldaan wordt, wordt de navigatie en locatie berekend in pixels.

```

1 let vergelijking=ricoLengte*parseFloat(this.state.location.coords.longitude)-
   parseFloat(this.state.location.coords.latitude)
2 let vergelijking2=ricoLengte*parseFloat(this.state.listTrack[0][ "LB-LONG "])-
   parseFloat(this.state.listTrack[0][ "LB-LAT "])
3 let vergelijking3=ricoLengte*parseFloat(this.state.listTrack[0][ "RB-LONG "])-
   parseFloat(this.state.listTrack[0][ "RB-LAT "])

```

```

4 let statement, statement2;
5 if (parseFloat(this.state.listTrack[0]["LB-LONG"]) - parseFloat(this.state.
    listTrack[0]["LO-LONG"]) > 0)
6 {
7     statement = (vergelijking) > (vergelijking2)
8     statement2 = (vergelijking3) > (vergelijking)
9 }
10 if (parseFloat(this.state.listTrack[0]["LB-LONG"]) - parseFloat(this.state.
    listTrack[0]["LO-LONG"]) < 0)
11 {
12     statement = (vergelijking) < (vergelijking2)
13     statement2 = (vergelijking3) < (vergelijking)
14 }
```

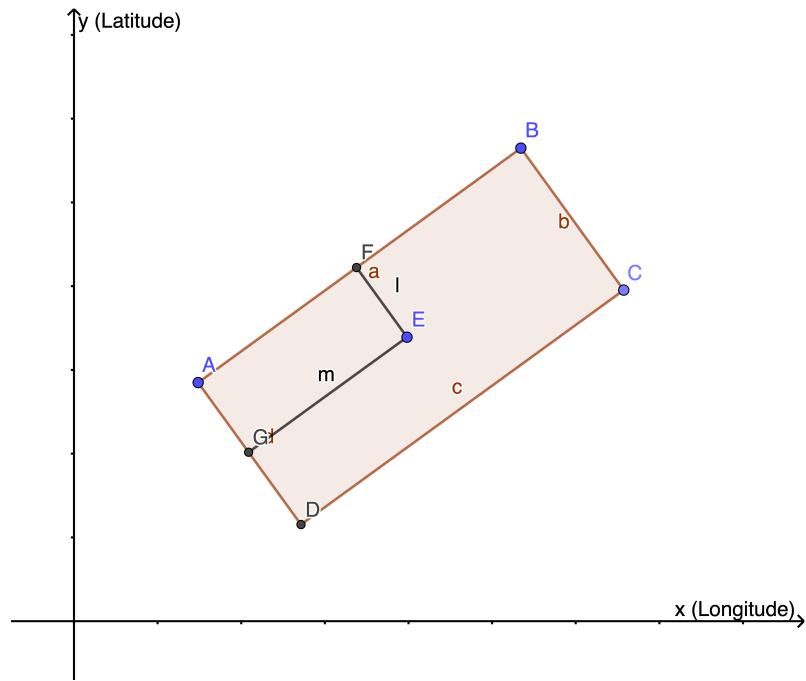
**Listing 5.5:** Vergelijkingen, opgesteld in overeenkomst met bovenstaande vergelijkingen.

#### 5.5.3.4 Positionering huidige locatie op het perron

De positionering van huidige locaties wordt berekend met de wiskundige eigenschap om de loodrechte afstand van een punt tot een rechte te berekenen:

$$d(P,l) = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}} \quad (5.12)$$

We zullen dus de huidige locatie berekenen door de loodrechte afstand van dit punt tot de breedte en lengte van het perron te berekenen. We verduidelijking dit met de figuur 5.6 waar punt E de huidige locatie voorstelt en lijn l en m de afstanden zijn die we moeten berekenen. Hiervoor hebben we weer de richtingscoëfficiënten nodig, berekend in 5.6.



**Figuur 5.6:** Visualisatie van de locatie op het perron. Punt E stelt hier de huidige afstand voor, lijnstuk l is de loodrechte afstand van punt E tot lijnstuk AB en lijnstuk m is de loodrechte afstand van punt E tot lijnstuk AD.

Op de figuur passen we 5.12 toe:

$$y - y_B = rico * (x - x_B) \quad (5.13)$$

$$0 = rico * (x) - rico * (x_B) + y_B - y \quad (5.14)$$

$$0 = rico * (x) - y - rico * (x_B) + y_B \quad (5.15)$$

$$a = rico \quad (5.16)$$

$$b = -1 \quad (5.17)$$

$$c = -rico * (x_B) + y_B \quad (5.18)$$

$$l = \frac{|(rico * x_E) - y_E - (rico * x_B) + y_B|}{\sqrt{rico^2 + (-1)^2}} \quad (5.19)$$

Dit is de afstand in het geografisch coördinatenstelsel. We willen echter de afstand in pixels t.o.v. de linkerbovenhoek van het perron hebben. Dit doen we met de volgende berekening:

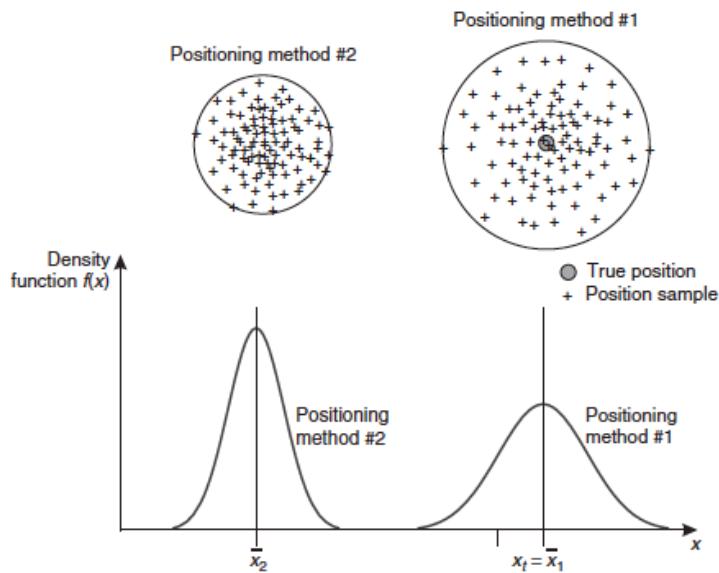
$$|AD| = \sqrt{(x_A - x_D)^2 + (y_A - y_D)^2} \quad (5.20)$$

Hiermee hebben we de afstand van het perron in het geografisch coördinatenstelsel. Door lijnstuk l nu te delen door lijnstuk AD, weten we de verhouding ten opzicht van de bovenkant van het perron waar de huidige locatie zich bevindt.

$$V = \frac{l}{|AD|} \quad (5.21)$$

Door deze verhouding nu te vermenigvuldigen met het aantal pixels van het perron (berekend in sectie 5.5.2.1) krijgen we het aantal pixels waar we onze huidige locatie moeten plaatsen ten opzichte van de bovenkant. Idem voor de berekening van lijn m.

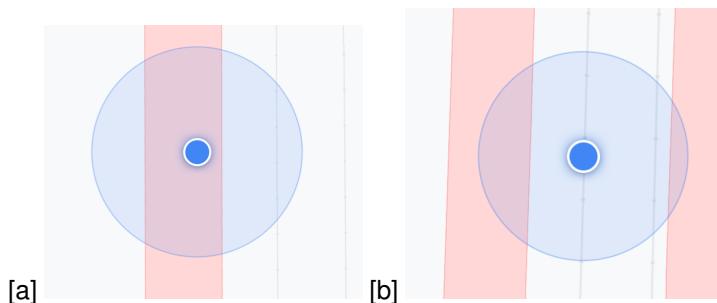
### 5.5.3.5 Nauwkeurigheid



**Figuur 5.7:** Verschil in nauwkeurigheid tussen twee positioneringsmethodes. Positioneringsmethode 1 heeft een hogere nauwkeurigheid, maar lagere precisie [1].

Nauwkeurigheid kan vaak verward worden met precisie, er is echter een groot verschil in beide termen. Een hogere precisie kan verkregen worden als meerdere samples dicht op elkaar worden gesampled, dat betekent echter niet dat de nauwkeurigheid klopt. Dit wordt aangetoond in figuur 5.7 waar positioneringsmethode 2 een veel hogere precisie heeft, dan positioneringsmethode 1, echter is positioneringsmethode 1 veel nauwkeuriger gezien het gemiddelde op de true position ligt. Het nauwkeurigheidscijfer wordt door React Native in aantal meters meegegeven. Het is dus hoe lager het nauwkeurigheidscijfer, des te beter de nauwkeurigheid.

### 5.5.3.6 Implementatie van nauwkeurigheid in de applicatie

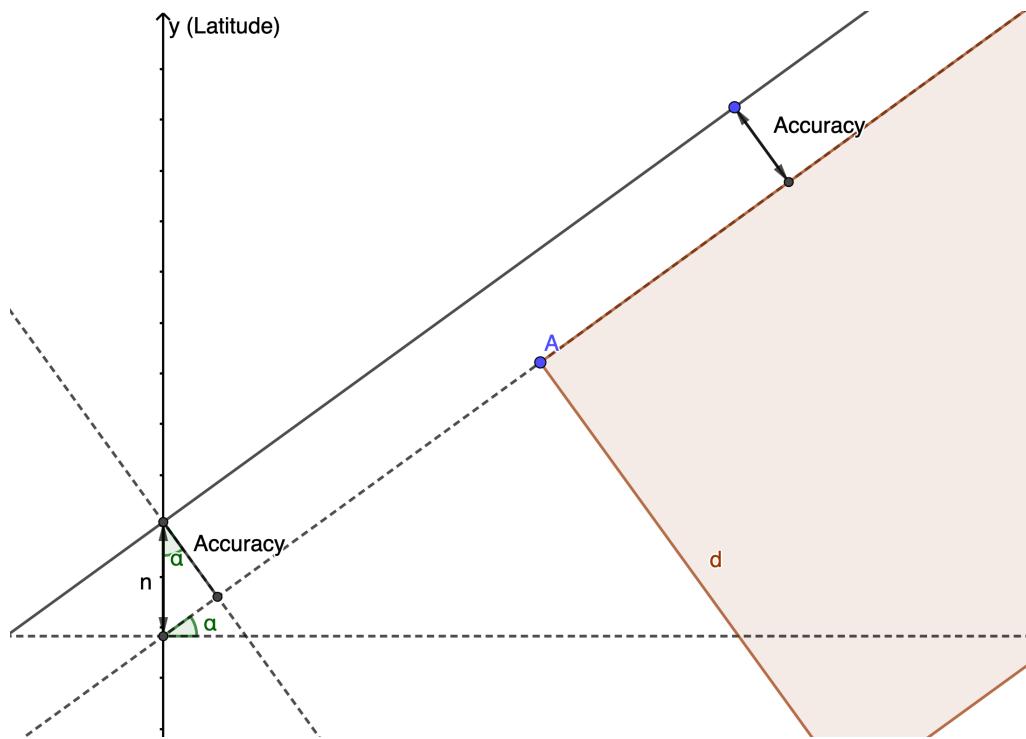


**Figuur 5.8:** 2 situaties van locatiebepaling. De blauwe bol is de berekende huidige locatie, de grote blauwe cirkel is de nauwkeurigheidscirkel, de rode rechthoeken zijn de perrons.

Voor onze applicatie speelt nauwkeurigheid een grote rol. Wanneer de smartphone fout lokaliseert en niet op het perron zit, werkt de applicatie niet. In bovenstaande figuur 5.8 zien we 2 mogelijke situaties van locatiebepaling op het perron. in situatie (a) ligt de locatie op het perron, dus hebben we geen probleem om de navigatie naar een rijtuig te starten. In geval (b) staan we echter op het perron, maar door een slechte locatiebepaling is onze locatie naast het perron berekend. Aan de vooropgestelde randvoorwaarden van sectie 5.5.3.6 wordt niet voldaan en de navigatie wordt niet gestart. De nauwkeurigheidscirkel reikt echter wel tot dit perron, waarmee de smartphone aangeeft dat hij mogelijk fout is en de locatie wel op het perron ligt. Doordat we dankzij deze nauwkeurigheidscirkel de foutenmarge weten, kunnen we er wel voor zorgen om de locatie zo weer te geven dat het doel van onze applicatie zolang mogelijk kan werken. Dit doen we door te controleren of de huidige locatie + nauwkeurigheidsgetal (gelijk aan de straal van de nauwkeurigheidscirkel) mogelijks nog op het perron zou kunnen zitten. Indien dit zo is, veranderen we de huidige locatiebol ( (a) in figuur 5.9) in een verticale balk. Deze verticale balk geeft weer waar we ongeveer op het perron zouden zitten ( (b) in figuur 5.9). Op deze manier kan men ervoor zorgen dat men toch naar de juiste richting kan navigeren.



**Figuur 5.9:** Wanneer de huidige locatie zich op het perron bevindt, zal dit verschijnen als een bolletje (a). Wanneer deze zich niet op het perron bevindt, maar wel nog in de nauwkeurigheidscirkel ligt, verschijnt dit als een verticale balk (b) die de hoogte heeft van de hoogte van het perron.



**Figuur 5.10:** Omzetting van het nauwkeurigheidsgetal in zijn y-coördinaat.

We zetten de nauwkeurigheid om in geografische coördinaten, aan de hand van een constante die we kunnen afleiden uit de verhouding perron in meter tot geografische coördinaten/ (benadering).

$$\text{Accuracy}(geo) = \text{Accuracy}(meter) * \text{meterToGeo} \quad (5.22)$$

Nu hebben we onze nauwkeurigheid in een afstand in het geografisch coördinatenstelsel omgezet. Deze afstand staat loodrecht op onze perronbreedtes. We zoeken de y-coördinaat ( $n$  op figuur 5.10).

$$\alpha = \arctan(rico) \quad (5.23)$$

$$\cos(\alpha) = \frac{\text{Accuracy}}{n} \quad (5.24)$$

$$n = \frac{\text{Accuracy}}{\cos(\alpha)} \quad (5.25)$$

Nu we dit weten, kunnen we de ongelijkheden licht aanpassen.

$$rico * x_B - y_B - n < rico * (x_{CL}) - y_{CL} \quad (5.26)$$

$$rico * x_D - y_D + n > rico * (x_{CL}) - y_{CL} \quad (5.27)$$

We lossen deze vergelijkingen op als er aan de vergelijkingen 5.9 en 5.10 niet werd voldaan.

### 5.5.4 Visualisatie van de trein

In de volgende secties zullen we bespreken hoe we de trein visualiseren. We zullen bespreken hoe de visualisatie afhangt van de verschillende soorten rijtuigen en hoe we bepalen waar de trein op het perron stopt.

#### 5.5.4.1 Train component (Train.js)

De train component is een custom component. Dit is een deel code verhult in een component die je gemakkelijk zou kunnen hergebruiken en zorgt ook voor het behoud van overzichtelijkheid. Deze roepen we op in ons NavigatieScreen.js aan de hand van de <Train> tag. In de train component bevinden zich de volgende functies:

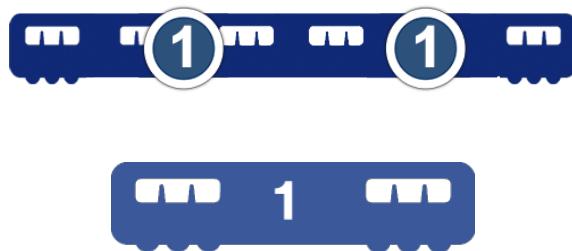
- Rendering van de rijtuigen en optiemogelijkheden (te vinden in sectie 5.5.4.2)
- Berekening voorkeur (te vinden in sectie 5.5.6.1)

Argumenten die worden meegegeven met de train tag zijn:

- **data={this.state.listTrainSpecifics}**: data die via de get\_train\_info.php uit de 'commercialtraincompositions' tabel van de databank zijn gehaald en bevat info over de rijtuigen van de specifieke trein.
- **onPress={this.makeNavigation}**: de functie die vanuit deze train component wordt opgeroepen wanneer een rijtuig wordt aangeklikt.
- **offset={offsetpx}**: offset waar de trein moet beginnen ten opzichte van het perron, wordt berekend in NavigatieScreen (zie sectie 5.5.4.3).
- **voordeur={this.state.voordeur}**: wanneer in het keuzemenu een bepaalde categorie wordt geselecteerd, wordt dit doorgegeven naar de train component, aangezien deze ervoor zorgt dat het juiste rijtuig wordt belicht.
- **CurLocPixelX={CurLocPixelX}**: dit wordt meegegeven omdat in de train component wordt gezocht naar het rijtuig dat de opgegeven voorkeur heeft en zich het dichtst bij de locatie bevindt.

#### 5.5.4.2 Rendering van de rijtuigen en optiemogelijkheden





**Figuur 5.11:** Een greep uit mogelijke rijtuigafbeeldingen die gerenderd kunnen worden: bovenaan een voorbeeld van een specifiek subtype trein (AM08M-a), in het midden een algemene afbeelding met een bepaalde lengte (hier 27 meter) en onderaan een algemene afbeelding.

Wanneer we rijtuigen renderen, doen we dit door deze één voor één in de train component te doorlopen. Op basis van type, lengte van het voertuig, aanwezigheid van 1e of 2e klasse,... wordt de juiste afbeelding getoond. De meeste van deze afbeeldingen hebben we hier speciaal voor gecreëerd. In figuur 5.11 zijn enkele mogelijkheden te zien. De rendering van de optiemogelijkheden gebeurt in de constructor van de train component aangezien deze onveranderlijk is. De opties die we in onze applicatie verwerken zijn:

- Aanwezigheid van airco
- Faciliteitsvoorziening voor fietsers
- Faciliteitsvoorziening voor personen met een beperking
- Beschikbaarheid van stopcontacten
- Aanwezigheid van klaptafels
- Aanwezigheid van toiletten

Deze worden als volgt voorgesteld in de applicatie:



**Figuur 5.12:** Pictogrammen die in onze applicatie zullen worden gerenderd.

#### 5.5.4.3 Offsetberekening van de trein

Wanneer een trein op het perron arriveert, stopt deze niet aan het begin van het perron maar op een bepaalde plaats ten opzichte van het begin van het perron. Om te weten waar treinen van een bepaalde lengte arriveren, heeft de NMBS bepaalde data hierover opgeslagen. Omdat we deze data niet ter beschikking konden krijgen, is een vervangfunctie geschreven, die gaat als volgt:

```

1 treinOffsetBerekening = () => {
2   let breedtemeter=(perronBreedtePixels/meterNrPixelOmzetting) //  

    perronBreedtePixels is berekend , meterNrPixelOmzetting is een constante,  

    resultaat is breedte van het perron in meter
3   let halfperron=parseFloat(breedtemeter)/2 //afstand van de breedte van een  

    half perron in meter
4   let totalelengte=0
5   for (let i=0;i<(this.state.listTrainSpecifics).length;i++) //Ga over alle  

    rijtuigen
6   {
7     totalelengte+=parseFloat(this.state.listTrainSpecifics[i].Length)/1000 //  

      Totale lengte van de rijtuigen
8   }
9   let halvelengte=totalelengte/2 //Afstand tot de helft van de rijtuigen
10  let offsetmeter=halfperron-halvelengte //Verschil=waar trein begint op het  

    perron in meter
11  treinOffsetPixels=(offsetmeter*meterNrPixelOmzetting) //omzetting hiervan in  

    pixels
12  if (parseFloat(treinOffsetPixels) < 0)
13  {
14    treinOffsetPixels=0; //Indien het perron groter is dan de trein, begint de  

      trein gewoon vanaf het begin van het perron
15  }
16
17 }

```

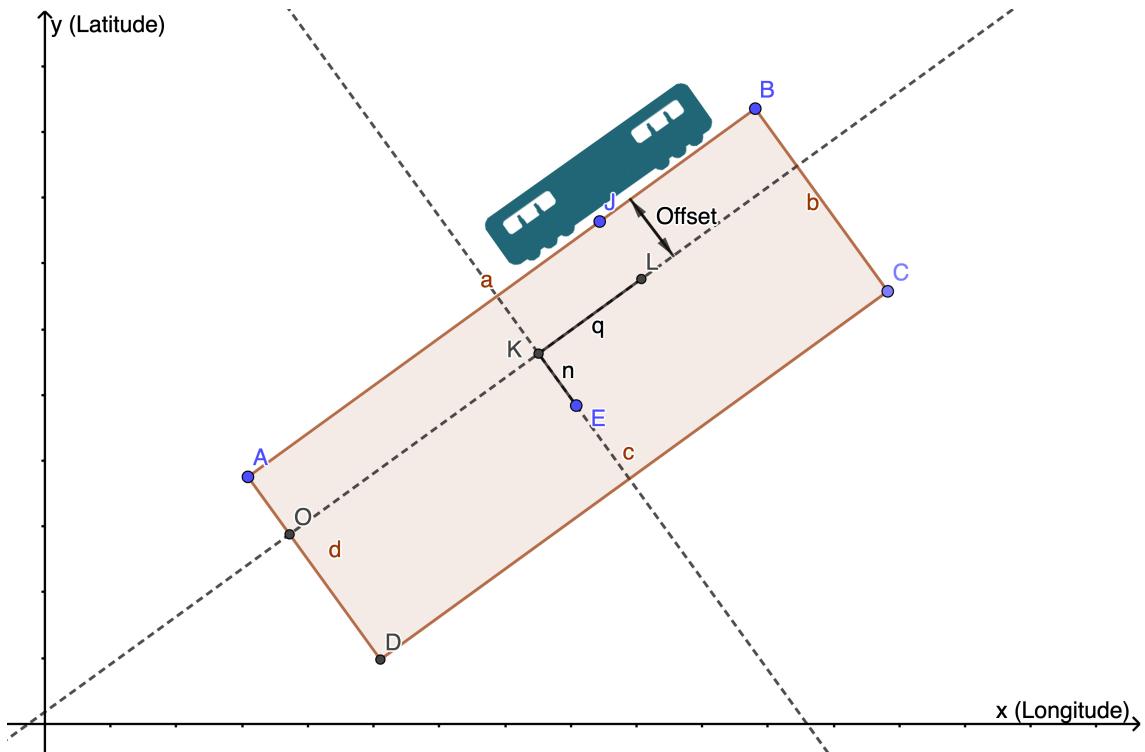
**Listing 5.6:** Offsetberekeningsfunctie voor een trein

We zorgen er hiermee voor dat onze trein mooi in het midden van ons perron komt. Indien de trein langer is dan het perron, is de offset 0 (zie regel 12 - 15 in listing 5.6). Doordat de enige output de offset is, kan deze functie makkelijk omgevormd worden indien er data van de NMBS zou gebruikt worden.

### 5.5.5 Navigatiebepaling

Het hoofddoel van de applicatie is om te navigeren naar de juiste locatie, daarom is het navigatie-onderdeel natuurlijk heel belangrijk voor onze applicatie. In de volgende sectie bespreken we 2 opties hoe we dit kunnen realiseren, beide zijn doorheen de implementatiefase geprogrammeerd.

### 5.5.5.1 Eerste optie: berekening met geografische coördinaten



**Figuur 5.13:** Navigatiebepaling eerste optie in het geografisch coördinatenstelsel.

We maken onze navigatie zo dat vanuit de huidige locatie (punt E in figuur 5.13) een navigatielijn wordt getrokken loodrecht richting de treinkant (lijn n in figuur 5.13) tot een bepaalde offset t.o.v. de spoorkant. Daarna wordt de horizontale lijn gevuld (lijn q) tot men aan het eindpunt (L) is. We krijgen dus een navigatiepad dat bestaat uit 1 horizontale (q) en 1 verticale (n) lijn. Als we aan punt L staan, staan we correct want dit is het punt waar men het rijtuig moet opstappen.

Eerst willen we de geografische coördinaten van punt K te weten komen, dit doen we op de volgende manier:

- Eerst berekenen we de twee richtingscoëfficiënten, één is gelijk aan 5.6, de andere staat hier loodrecht op, waardoor we deze op volgende manier kunnen berekenen:

$$rico2 = \frac{-1}{rico} \quad (5.28)$$

- Gelijkaardig aan hoe we in sectie 5.5.3.6 de y-coördinaat berekenden voor de nauwkeurigheid, doen we dit voor de offset:

$$\alpha = \arctan(rico) \quad (5.29)$$

$$\cos(\alpha) = \frac{Offset}{y_{Offset}} \quad (5.30)$$

$$y_{Offset} = \frac{Offset}{\cos(\alpha)} \quad (5.31)$$

- De twee rechten die K snijden zijn:

$$y - y_A = rico(x - x_A) - y_{Offset} \quad (5.32)$$

$$y - y_E = rico2(x - x_E) \quad (5.33)$$

- Omvormend naar y:

$$y = rico(x - x_A) - y_{Offset} + y_A \quad (5.34)$$

$$y = rico2(x - x_E) + y_E \quad (5.35)$$

- Aan elkaar gelijkstellen:

$$rico(x - x_A) - y_{Offset} + y_A = rico2(x - x_E) + y_E \quad (5.36)$$

- Dit lossen we op naar onbekende x:

$$rico * x - rico2 * x = rico2(-x_E) + y_E + rico * x_A + y_{Offset} - y_A \quad (5.37)$$

$$x_K = x = \frac{rico2(-x_E) + y_E + rico * x_A + y_{Offset} - y_A}{rico - rico2} \quad (5.38)$$

Nu weten we de longitude van dit punt. Om nu de latitude te berekenen vullen we dit in:

$$y_K = y = rico2(x_K - x_E) + y_E \quad (5.39)$$

We kunnen te weten komen waar in pixels dit op ons perron terechtkomt, op identieke wijze waarmee we onze huidige locatie berekenen (zie sectie 5.5.3.4). Met de bekomen geografische coördinaten kunnen we de lengte van n te weten komen, dit doen we op dezelfde manier als de perronberekening (zie sectie 5.5.2.1), waar onze twee coördinaten het verkregen punt E en onze huidige locatie zijn.

Nu zullen we ervoor zorgen dat we onze horizontale navigatielijn correct weergeven. Eerst moeten we hiervan bepalen waar ons punt L moet terechtkomen, dit wordt gedaan aan de hand van data uit de databank. Elke rijtuiglengte wordt samen met deuroffset en treinoffset (berekend in sectie 5.5.4.3) opgeteld. Dit wordt omgezet in pixels.

$$|AD| = \sum_{i=1}^n rijtuig_i + treinoffset + deuroffset \quad (5.40)$$

$$V (= \frac{|AD|(px)}{Breedteperron(px)}) \quad (5.41)$$

Deze verhouding vermenigvuldigen we nu met de afstand van de breedtelijn van het perron in geografische coördinaten om zo de afstand in het geocoördinatensysteem te krijgen. Om nu

te weten welke geografische coördinaten L heeft, doen we volgende afleiding: We kunnen 2 vergelijkingen opstellen:

$$\begin{cases} |KL| = \sqrt{(x_L - x_K)^2 + (y_L - y_K)^2} \\ y_L - y_K = rico(x_L - x_K) \end{cases} \quad (5.42)$$

We vullen de tweede vergelijking in de eerste, hierdoor krijgen we:

$$|KL|^2 = (x_L - x_K)^2 + rico^2 * (x_L - x_K)^2 \quad (5.43)$$

$$|KL|^2 = (x_L - x_K)^2 (1 * +rico^2) \quad (5.44)$$

$$(x_L - x_K)^2 = \frac{|KL|^2}{(1 * +rico^2)} \quad (5.45)$$

$$(x_L - x_K) = \pm \sqrt{\frac{|KL|^2}{(1 * +rico^2)}} \quad (5.46)$$

$$(x_L) = \pm \sqrt{\frac{|KL|^2}{(1 * +rico^2)}} + x_K \quad (5.47)$$

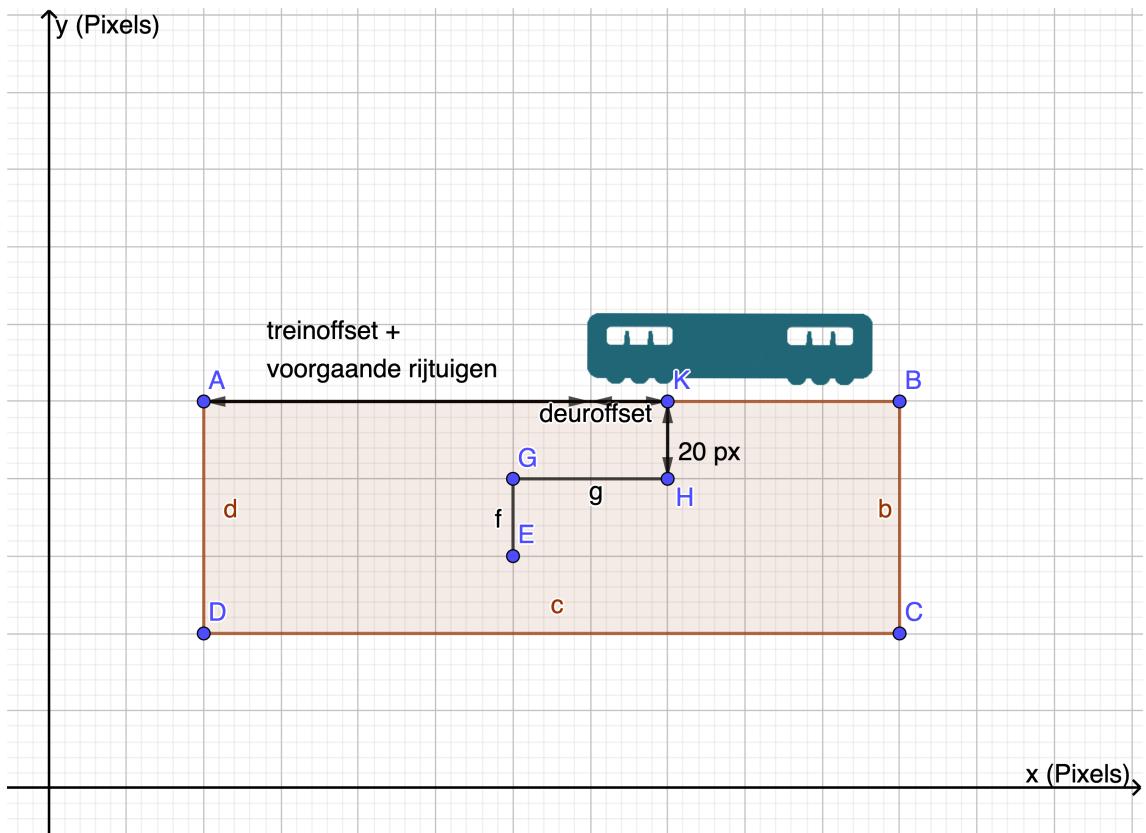
Hiermee vinden we de x-coördinaat op positie L. Om de y-coördinaat te vinden vullen we dit in de vergelijking in. Bij een negatieve afstand en positieve richtingscoëfficiënt moeten we de negatieve vierkantswortel gebruiken.

$$y_L = rico * (x_L - x_K) + y_K \quad (5.48)$$

Hiermee hebben we de geografische coördinaten van punt L. Tussen de bekomen coördinaten kunnen we nu weer een grootberekening doen (zie sectie 5.5.2.1) en een coördinatenberekening om de coördinaten in pixels te weten te komen waar de horizontale en verticale lijn van ons navigatiepad moet beginnen (zie sectie 5.5.3.4)).

We tekenen de horizontale en verticale lijn aan de hand van de `<View>` tag. Aan deze tag geven we een left, top, width en height argument mee. Deze tekent dan een rechthoek vanaf de left en top coördinaat met grootte width en height. Indien  $|KL|$  positief is, betekent dit dat we naar rechts moeten bewegen op het perron, en zal de left coördinaat waar de horizontale lijn moet uit vertrekken gelijk zijn aan punt K. Indien  $|KL|$  negatief is, moeten we naar links gaan en zal de left coördinaat waaruit de horizontale lijn moet vertrekken gelijk zijn aan punt L.

### 5.5.5.2 Tweede optie: berekening met pixels



**Figuur 5.14:** Navigatiebepaling tweede optie in het pixel coördinatenstelsel.

Een andere optie is dat we enkel met de geolocatie van onze huidige locatie werken en vanaf dan alles renderen in pixels. We berekenen het eindpunt van de navigatie:

$$x_H(\text{pixels}) = \text{offset} + \text{tot begin trein} + \text{alle voorgaande rijtuigen} + deuroffset \quad (5.49)$$

Voor  $y_H$  nemen we dan een vaste afstand in pixels.

$$y_H = 20 \quad (5.50)$$

We nemen het punt G zodanig dat het loodrecht boven punt E zit en op dezelfde lijn als H, waardoor:

$$x_G = x_E \quad (5.51)$$

$$y_G = y_H \quad (5.52)$$

Op basis van deze berekeningen kunnen we dan volgende afstanden berekenen:

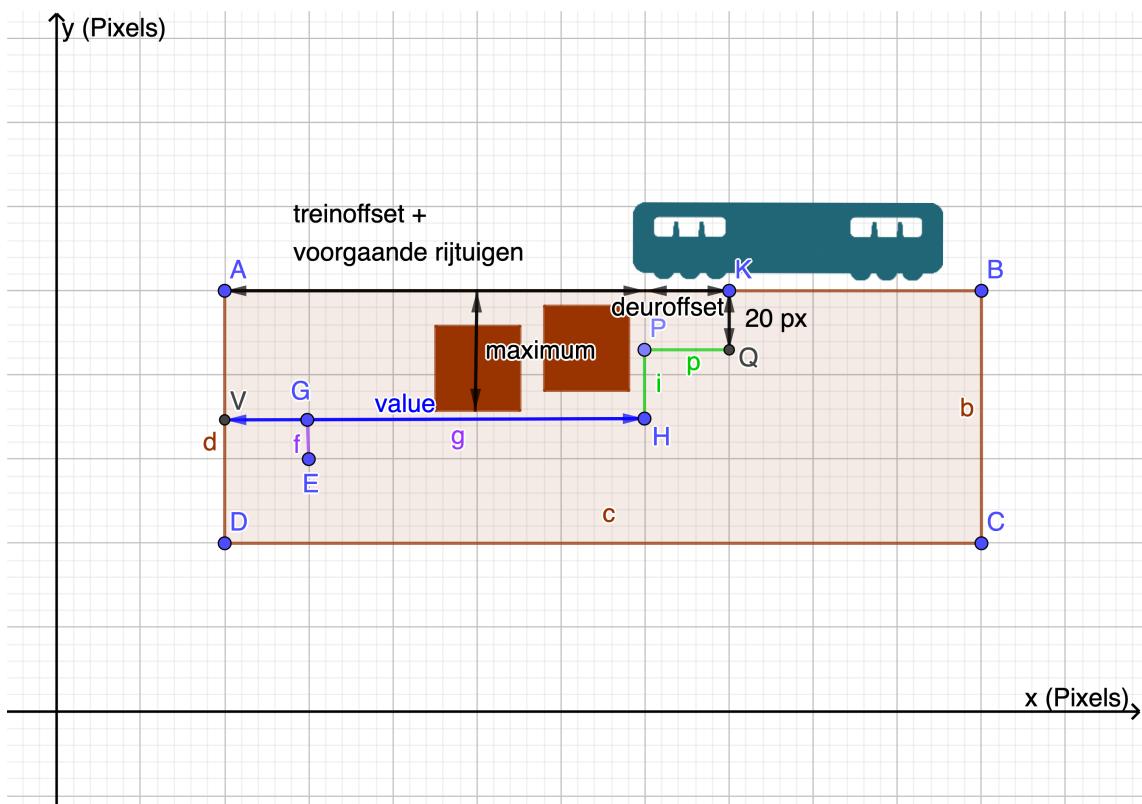
$$f = \gamma_G - \gamma_E \quad (5.53)$$

$$g = x_H - x_G \quad (5.54)$$

Ook hier geldt dezelfde voorwaarde als in optie 1 want ook hier wordt de horizontale en verticale lijn getekend met de `<View>` tag. Indien  $|GH|$  positief is, betekent dit dat we naar rechts moeten bewegen op het perron, en zal de left coördinaat waar de horizontale lijn moet uit vertrekken gelijk zijn aan punt G. Indien  $|GH|$  negatief is, moeten we naar links gaan en zal de left coördinaat waaruit de horizontale lijn moet vertrekken gelijk zijn aan punt H.

**Conclusie** Omdat de navigatieberekening aan de hand van pixels minder rekenwerk van de applicatie vraagt en nauwkeuriger is (men neemt geen kwadraten van kleine kommagetallen zoals geografische coördinaten waardoor nauwkeurigheid verminderd), zullen we deze gebruiken in onze applicatie. De volgende secties zijn dan ook toegepast op de pixelberekeningsmethode.

#### 5.5.5.3 Navigatie bij objecten op het perron



Figuur 5.15: Navigatiebepaling met perronobjecten

De mogelijkheid bestaat dat er één of meerdere objecten op het perron staan waardoor men aan 2 lijnen voor het navigatiepad niet genoeg heeft. Dit wordt opgelost met enkele toevoegingen aan de navigatieberekening.

In sectie 5.5.2.2 werd de objectPixels array al gemaakt, met alle hoeken per object in pixelafstand ten opzichte van de linkerbovenhoek van het perron. Om te weten of we met deze objecten rekening moeten houden bij de navigatie, zullen we eerst in een for-loop over alle objecten gaan en controleren of deze zich tussen het eindpunt van de navigatie (punt Q) en de huidige locatie (Punt E) bevindt. Vervolgens wordt gecheckt of in de y-richting de objecten de horizontale lijn van het navigatiepad (rechte p) doorkruisen. Indien dit zo is, wordt deze in de perronObjectenErin array gezet.

```

1 //Ga over alle objecten
2 for (let i=0; i < (objectPixels).length; i++)
3 {
4     //Voorwaarde als de navigatie vanuit huidige locatie punt E naar links gaat
5     if (naarLinks)
6     {
7         //Als we naar links gaan testen we of de x-coordinaat van de linkerzijde
9             van het object groter is dan de x-coordinaat van eindpunt Q en
10            kleiner is dan de x-coordinaat van locatiepunt E. Dat betekent dat
11            het object tussen punt E en punt Q zit. Een andere mogelijkheid is
12            dat het object groter is dan punt Q en punt E.
13            statement=((X_Q<objectPixels[i]["LOPixelX"]) && (objectPixels[i]["
14                "LOPixelX"]<X_E) || (objectPixels[i]["LOPixelX"]<X_Q && objectPixels[i]
15                ["ROPixelX"]>X_E))
16        }
17        //Voorwaarde als de navigatie vanuit huidige locatie punt E naar rechts
18        //gaat
19        else
20        {
21            //Als we naar rechts gaan testen we of de x-coordinaat van de rechterzijde
22            van het object kleiner is dan de x-coordinaat van eindpunt Q en groter
23            is dan de x-coordinaat van locatiepunt E. Dat betekent dat het object
24            tussen punt E en punt Q zit. Een andere mogelijkheid is dat het object
25            groter is dan punt Q en punt E.
26            statement=((X_Q>objectPixels[i]["ROPixelX"]) && (objectPixels[i]["
27                "ROPixelX"]>X_E) || (objectPixels[i]["LOPixelX"]<X_Q && objectPixels[i]
28                ["ROPixelX"]>X_Q))
29        }
30        if (statement)
31        {
32            // Controleer of de y coordinaat van rechte p zich tussen de boven en
33            onderkant van het object bevindt.
34            if ((objectPixels[i]["LOPixelY"]>Y_P) && (Y_P>objectPixels[i]["
35                "LBPixely"]))
36            {
37                perronObjectenErin.push(i)
38            }
39        }
40    }
41 }

```

**Listing 5.7:** Navigatie met perronobjecten: condities aangepast in overeenstemming met figuur 5.15.

We weten nu welke objecten ons navigatiepad versturen. We gaan de horizontale lijn van ons navigatiepad bijgevolg verlagen. Dit door te controleren welk van de objecten zich het verstuert verticaal over het perron uitstrekt. Vanaf deze waarde (maximum) doen we 5 pixels bij, dit is waar de horizontale lijn van ons navigatiepad zich zal bevinden in de y-richting. In figuur 5.15 is de maximumwaarde de waarde van het linkse object.

```

1     if (objectPixels[i]["LOPixelY"]>maximum)
2     {
3         maximum=objectPixels[i]["LOPixelY"]
4     }

```

**Listing 5.8:** Controleer welk object zich het verstuert uitstrekt in de y-richting.

Vervolgens gaan we testen tot waar ons navigatiepad wordt verstoord door de objecten. Wanneer we vanaf de huidige locatie naar rechts moeten gaan (zoals in onze figuur, gezien het eindpunt rechts t.o.v. de huidige locatie ligt) is dit tot (de rechterkant van) het laatste object. Als we naar links gaan is dit (de linkerzijde van) het eerste object. We bewaren de waarde van de x-coordinaat in variabele value. Hierna (vanaf punt H in figuur 5.15) wordt de navigatie hervat zoals in bovenstaande secties beschreven.

```

1     //Als ons eindpunt eindigt in een object, wordt onze value X_Q
2     if (objectPixels[i]["LOPixelX"]<X_Q && objectPixels[i]["ROPixelX"]>X_Q)
3     {
4         value=X_Q;
5         eindigtTussenIn=true;
6     }
7     else if (naarLinks)
8     {
9         //Value start met een groot getal, eindigt met de kleinste pixelwaarde
10        van een object, tot hier moet onze horizontale navigatielijn lager
11        liggen.
12        if(objectPixels[i]["LOPixelX"]<value)
13        {
14            value=objectPixels[i]["LOPixelX"];
15        }
16        // Value start met een klein getal, eindigt met de grootste pixelwaarde
17        van een object, tot hier moet onze horizontale navigatielijn lager
18        liggen.
19        else if(objectPixels[i]["ROPixelX"]>value)
20        {
21            value=objectPixels[i]["ROPixelX"];
22        }
}

```

---

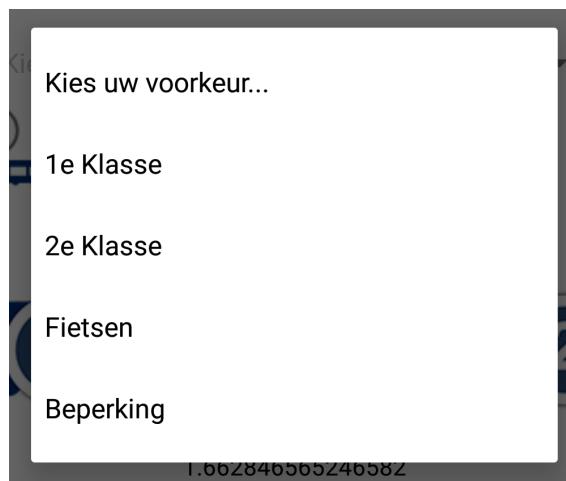
**Listing 5.9:** We zoeken het eindpunt tot waar ons navigatiepad wordt verstoord door de objecten.

Nu hebben we alles om ons volledige navigatiepad te creëren. We creëren 2 extra navigatiepaden (i en p in figuur 5.15). Een uitzondering hierin is wanneer het eindpunt van onze navigatie zich bevindt tussen de linker- en rechterzijde van eenzelfde object: dan hebben we maar 2 navigatiebalken nodig, gezien we nooit naar de navigatiebalken i en p gaan. Om hergebruik van de code te kunnen waarborgen, schrijven we de navigatiebepaling van sectie 5.5.5.2 in een aparte functie. We voeren deze sectie 5.5.5.2 dan ook 2 keer uit:

- Navigatiepad EGH in figuur 5.14 wordt vervangen door navigatiepad EGH in figuur 5.15. We weten dat  $x_G$  gelijk is aan de huidige locatie  $x_E$ . We doen hier alsof ons eindpunt H is.  $y_h$  in formule 5.50 wordt gelijk aan de maximum value + 5.
- Navigatiepad EGH in figuur 5.14 wordt vervangen door navigatiepad HPQ in figuur 5.15. We doen hier alsof H onze 'huidige locatie' is en dat we naar eindpunt Q moeten navigeren.

## 5.5.6 Overige visuele elementen

### 5.5.6.1 Keuzemenu



**Figuur 5.16:** Het keuzemenu in werking met de mogelijke voorkeuren getoond.

Er zijn 2 opties om het ideale rijtuig te selecteren: via de rendering van de rijtuigafbeeldingen of via het keuzemenu element. Dit is een klein submenu (zie figuur 5.16) waarbij er een voorkeur wordt opgegeven en de train component automatisch het ideale rijtuig voor u selecteert. Omdat we de rijtuigen renderen in een aparte train component wordt de bepaling van het ideale rijtuig echter niet in NavigatieScreen.js gedaan, maar in het bestand van de train component. Het keuzemenu wordt

gemaakt aan de hand van een voorgeschreven RNPickerSelect API. Onze code ziet er als volgt uit:

```

1 <RNPickerSelect onValueChange={(value) => this.setState({voordeur: value})}
2   placeholder={{label: 'Kies uw voorkeur...', value: 0,}} style={pickerStyle}
3   items={[
4     { label: '1e Klasse', value: 'lastPlanned_materialUnits_isFirstClass' },
5     { label: '2e Klasse', value: 'lastPlanned_materialUnits_isSecondClass' },
6     { label: 'Fietsen', value: 'lastPlanned_materialUnits_hasBikeSection' },
7     { label: 'Beperking', value: 'lastPlanned_materialUnits_hasPrmSection' }
8   ]}
9 />

```

**Listing 5.10:** Het keuzemenu met de mogelijke opties.

**Berekening voorkeur** Wanneer er via het keuzemenu een voorkeur geselecteerd is, moet men ervoor zorgen dat er een navigatie is naar het rijtuig die én deze specificatie bevat én het dichtst bij de huidige locatie is. Dit doen we in de train component op volgende manier in pseudocode:

```

1 sorteerDichtbijCL()
2 afstanden[]
3 for (j = (1,2, ..., laatste rijtuig+1)
4 {
5   afstand = afstand tot het begin van de trein
6   for (i = 0, 1 ... , j)
7   {
8     if (j == i+1)
9     {
10       if ((afstand+rijtuig[i].deur - huidige locatie x) > (afstand + rijtuig[
11         i].deur2 - huidige locatie x))
12       {
13         afstand=(afstand+rijtuig[i].deur2 - huidige locatie x)
14       }
15       else
16       {
17         afstand=(afstand+rijtuig[i].deur - huidige locatie x)
18       }
19       break;
20     }
21   }
22   afstand+=rijtuig[i].lengte

```

```

23     }
24 }
25 rijtuig[j-1].afstandTotHuidigeLocatie=afstand
26 rijtuig[j-1].Rijtuig=j-1
27 }
```

**Listing 5.11:** Berekening dichtsbijzijnde rijtuig, algoritme in pseudocode.

We tellen alle rijtuigen op tot het rijtuig  $j$ . We moeten natuurlijk de afstand tot de deur nemen die zich het dichtst bij de huidige locatie bevindt, waardoor er hier een extra conditie bijkomt. We vullen de afstanden in een nieuw id element 'AfstandTotLoc' in en dit voor elk rijtuig ( $j$ ).

Nu we alle afstanden berekend hebben, sorteren we de lijst van kortste naar langste afstand. Dit doen we op aan de hand van de volgende functie:

```

1 function compare( a, b ) {
2   if ( a.AfstandTotLoc < b.AfstandTotLoc ) {
3     return -1;
4   }
5   if ( a.AfstandTotLoc > b.AfstandTotLoc ) {
6     return 1;
7   }
8   return 0;
9 }
```

**Listing 5.12:** Het sorteeralgoritme, de bedoeling is om te sorteren van kortste naar langste afstand tot de huidige locatie.

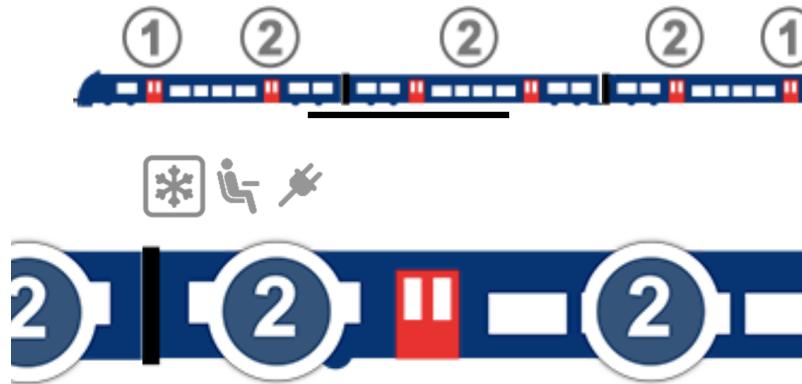
Dit is een bubblesort sorteeralgoritme: men vergelijkt twee naast elkaar gelegen elementen, indien afstand  $a$  groter is dan  $b$  is, wordt 1 teruggegeven, wat betekent dat element  $b$  een lagere index krijgt. Bij een terugkeerwaarde 0 is de positie onveranderd, en bij -1 krijgt  $a$  een lagere index. Dit is logisch, gezien deze dan een grotere afstand heeft.

Na deze sortering, wordt in een for-loop gecontroleerd welke rijtuigen de voorkeur hebben, de gesorteerde lijst wordt afgelopen van kortste naar langste afstand tot de locatie, en vanaf het moment dat er een match gevonden wordt, zal de for-loop afgebroken worden en de navigatie starten naar het gekozen rijtuig.

### 5.5.6.2 Meldingsvenster verkeerde perron

Wanneer er niet aan de randvoorwaarden van sectie 5.5.3.3 en 5.5.3.6 wordt voldaan, tonen we een meldingsvenster. Dit venster toont de boodschap dat de reiziger zich niet op het correcte perron bevindt. Voor dit modal Screen maken we gebruik van een Dialog component uit de 'React Native-popup-dialog' API.

### 5.5.6.3 Minitatuurweergave



**Figuur 5.17:** Miniatuurtrein (bovenaan) en trein in verhouding met het perron (onderaan).

Omdat meestal maar een klein deel van een trein kan worden weergegeven op het scherm, willen we ervoor zorgen dat de gebruiker in bepaalde mate een overzicht van de trein kan hebben. Dit doen we aan de hand van een miniatuurweergave. Hierin renderen we een kleine versie van de rijtuigafbeeldingen, net zoals de grote rijtuigen op basis van type, aanwezigheid van 1e klasse, ... Omdat we de gebruiker willen aantonen waar de getoonde versie van de trein zich bevindt ten opzichte van de miniatuurversie, geven we dit weer aan de hand van een zwarte lijn. Deze wordt gemaakt met behulp van de `<View>` tag. Hierdoor is er nood aan een correcte verhouding tussen de miniatuurtrein en de getoonde trein. We nemen een verhouding van 127/500 om elke pixel naar een pixel van de miniatuurtrein om te zetten. Deze verhouding is ook vastgelegd in constants.js (te vinden in bijlage A.1.1).

```

1 <ScrollView
2   scrollEventThrottle={16}
3   onTouchStart={() => {ScrollToMiniatuur=true; ScrollTo=false}}
4   horizontal={true}
5   ref={(ref) => { this.scrollreferentieMiniatuur = ref;  }}
6   scrollEnabled={true}
7   showsHorizontalScrollIndicator={false}
8   onScroll={this.handleMiniatuurScroll}
9 >

```

**Listing 5.13:** ScrollView tag van de miniatuurtrein.

Ook de miniatuurweergave heeft net als de train component een verticale scrollView, omdat de trein te klein zou uitvallen wanneer we de gehele trein op 1 verticaal scherm zouden tonen. We willen implementeren dat wanneer we met de miniatuurtrein scrollen, de trein meebeweegt. Dit doen we door een functie bij beide verticale scrollViews te implementeren in de onScroll eigenschap (te zien in listing 5.13). Wanneer dus in een van de 2 miniatuurweergaves wordt gescrold, zal dit deze

functie activeren. Deze functies zien er voor de trainScrollView en miniatuurtainScrollView als volgt uit:

```

1 handleScroll = event => { //Wordt opgeroepen als er gescrold wordt met de grote
2   trein
3   if (ScrollTo == true)
4   {
5     this.scrollreferentieMiniatuur.scrollTo({x:(event.nativeEvent.contentOffset
6       .x-treinOffsetPixels)*pixelNrMiniatuurPixelOmzetting,y:0,animated:false
7       });
8   }
9 }
10 handleMiniatuurScroll = event => { //Wordt opgeroepen als er gescrold wordt met
11   de miniatuurtrein
12   if (ScrollToMiniatuur==true)
13   {
14     this.scrollreferentie.scrollTo({x:(event.nativeEvent.contentOffset.x/(
15       pixelNrMiniatuurPixelOmzetting))+treinOffsetPixels,y:0,animated:false})
16   }
17 }
```

**Listing 5.14:** Functies die worden opgeroepen vanuit de onScroll eigenschap van de scrollView.

Wanneer er met de grote trein wordt gescrold, zal dit de functie handleScroll oproepen, deze roept dan de scrollReferentieMiniatuur op, wat een referentie is naar het verticale scrollView van de miniatuurtrein. Omdat de miniatuurtrein enkel de trein weergeeft en niet het perron, moeten we de ContentOffset (de offset van de scrollview, vanaf dit punt wordt een deel van de scrollview op het scherm getoond) verminderen met de treinoffset. Om nu de pixels om te vormen naar de pixels in miniatuurvorm, vermenigvuldigen we dit met de pixelNrMiniatuur constante (127/500). De werkwijze is idem in de omgekeerde richting voor wanneer er met de miniatuurtrein wordt gescrold. Er kan een oneindige lus ontstaan wanneer er met de miniatuurtrein wordt gescrold, en dit doorgegeven wordt aan de grote trein. Deze merkt vervolgens dat ook hier de positie veranderd is en geeft dit op zijn beurt door aan de miniatuurtrein,... Om dit te vermijden geven we een extra 'onTouchStart' argument mee (te zien in 5.13). Dit zet de juiste variabelen (scrollTo of scrollToMiniatuur) op true, zodanig dat we niet in een oneindige lus komen.

#### 5.5.6.4 Extra informatiescherm

Bij wijze van uitbreiding hebben we nog een extra informatiescherm gemaakt. Wanneer men op de info button van het navigatiescherm drukt, komt hier info over de trein waaronder de lengte, de voorzieningen en het type trein. De voorzieningen in de trein worden in de constructor al aangemaakt, op dezelfde wijze als de opties boven elk rijtuig worden weergegeven.

## 5.6 Bevestiging correctie positie (EndScreen.js)

Wanneer de gebruiker in de buurt komt van de correcte positie, zal er genavigeerd worden naar het eindscherm. Hierop verschijnt een tekst met de boodschap dat de gebruiker zich op het correcte perron bevindt.

## 5.7 Uitbreidings: Animaties

Om de applicatie gebruiksvriendelijker te maken en om alles visueel mooi en vloeiend weer te geven, implementeren we enkele animaties. Hiervoor maken we gebruik van de 'Animated' library. Om waarden te kunnen animeren hebben we nood aan een Animated Value. Animated Values zijn variabelen die van waarde kunnen veranderen en pixel per pixel naar deze nieuwe waarde gaan. Als we bijvoorbeeld naar een nieuwe huidige locatie gaan, betekent dit dat we (snel) pixel per pixel omschuiven naar de nieuwe locatie, zodat deze vloeiend lijkt. Volgende elementen zullen we animeren:

- Navigatiebepaling:
  - De verplaatsing van de huidige locatie naar de nieuwe, geüpdatete plaats.
  - De grootte van het horizontale lijn van het navigatiepad naar het eindpunt, die afhankelijk is van de positie van de huidige locatie ten opzichte van de eindlocatie.
  - De grootte van het verticale lijn van het navigatiepad dat gaat van de huidige locatie naar het horizontale navigatiepad, omwille van dezelfde reden.
  - De verplaatsing van het horizontale lijn van het navigatiepad wanneer de huidige locatie zich verplaatst.
  - De verplaatsing van het verticale lijn van het navigatiepad wanneer de huidige locatie zich verplaatst.
- Treinafbeeldingen:
  - De selectie van een rijtuig zal ervoor zorgen dat andere rijtuigen transparanter worden.
- Perron:
  - Bij de eerste selectie van een rijtuig zal het perron verschijnen. Dit was voorheen nog niet zichtbaar.

### 5.7.1 Navigatiebewegingen vloeiend laten verlopen

Voor we onze animatie voor de navigatie starten, initialiseren we de Animated Values van de verplaatsing met een initiële waarde van 0. Deze waarden worden elke keer geherinitialiseerd met waarde 0 wanneer een nieuw rijtuig wordt geselecteerd, omwille van twee redenen:

1. Op deze elementen die van plaats veranderen (de lijnen van het navigatiepad en de huidige locatie) wordt een translatie uitgevoerd. Dit betekent dus geen absolute positionering ten opzichte van de linkerbovenhoek van het perron, maar een relatieve positie ten opzichte van waar de huidige locatie en navigatiepaden zich op dat moment bevinden. Een translatie kan dus negatief (verplaatsing naar links) of positief (verplaatsing naar rechts) zijn. In de stijlelementen (bv. 'left:' en 'top') van de navigatiepaden en van de huidige locatie zijn dus initiële (niet-Animated Value) waarden die ingegeven. Initieel wordt in left en top de eerste waarde (juist voor het perron verschijnt) ingevuld.
2. We doen dit telkens bij het aanduiden van een nieuw rijtuig omdat we onze applicatie niet onnodig willen beladen. Zo zullen we animaties pas activeren wanneer het perron verschijnt op het scherm. Op de achtergrond zal de huidige locatie echter wel altijd geüpdatet worden zodat deze onmiddellijk beschikbaar is voor navigatiebepaling.

```

1   this.state = {
2     CLanimx: new Animated.Value(0),
3     CLanimy: new Animated.Value(0),
4     NAVanimx1: new Animated.Value(0),
5     NAVanimy1: new Animated.Value(0),
6     NAVanimx2: new Animated.Value(0),
7     NAVanimy2: new Animated.Value(0)
8
9   };

```

**Listing 5.15:** Initiële Animated Values voor de states waarop translaties zal worden uitgevoerd.

Er moet dus voor gezorgd worden dat wanneer het perron verschijnt eenmalig de huidige waarde in de (niet-Animated Value) variabelen wordt bewaard, en nadien de animatie wordt geactiveerd. De niet-Animated value variabelen die de initiële waarde bevatten zijn:

- **initCurrentLocationLeft:** de initiële huidige positie in pixels in de x-richting. Bij animatie wordt hierop een translatie uitgevoerd door de this.state.CLanimx.
- **initCurrentLocationTop:** de initiële huidige positie in pixels in de y-richting. Bij animatie wordt hierop een translatie uitgevoerd door de this.state.CLanimY.
- **initHorizontalLineLeft:** de initiële positie waar het horizontale navigatiepad begint in

pixels in de x-richting. Bij animatie wordt hierop een translatie uitgevoerd door de this.state.NAVanimx1.

- **initHorizontalLineTop:** de initiële positie waar het horizontale navigatiepad begint in pixels in de y-richting. Bij animatie wordt hierop een translatie uitgevoerd door de this.state.NAVanimy1
- **initVerticalLineLeft:** de initiële positie waar het verticale navigatiepad begint in pixels in de x-richting. Bij animatie wordt hierop een translatie uitgevoerd door de this.state.NAVanimy1.
- **initVerticalLineTop:** de initiële positie waar het verticale navigatiepad begint in pixels in de y-richting. Bij animatie wordt hierop een translatie uitgevoerd door de this.state.NAVanimy2

Deze variabelen worden dus niet meer geüpdatet nadat het perron verschenen is.

Al deze initialisaties worden gedaan wanneer de navigatieberekeningen beginnen, aan de hand van de variabele netIngedrukt. Deze variabele wordt op 'true' gezet wanneer er net een rijtuig is geselecteerd. Andere elementen die worden opgeroepen zijn een scrollTo functie, die na het selecteren van een rijtuig automatisch scrollt naar een plek op het perron (offsetCL in listing 5.16) zodanig dat de huidige locatie in het midden van het scherm staat. Hierdoor kan de reiziger makkelijk de navigatie naar het rijtuig starten.

```

1 if (netIngedrukt)
2 {
3     this.perronInitialisatie() //Laat het perron verschijnen
4     var offsetCL = (CurLocPixelX) - (windowWidth/2)
5     this.scrollreferentie.scrollTo({x:offsetCL,y:0,animated:true});
6     netIngedrukt=false;
7 }
```

**Listing 5.16:** Functies die worden uitgevoerd wanneer een rijtuig juist is geselecteerd.

Aan de hand van de functie perronInitialisatie, worden in de initiële variabelen de huidige toestand toegewezen. Nadat dit gebeurd is, wordt netIngedrukt op false gezet en vanaf dit moment wordt elke verandering van grootte of positie geanimeerd.

De animatie wordt in gang gezet wanneer bemerkt wordt dat de locatie verandert (via de asynchrone watchPosition) en de netIngedrukt-variabele een waarde false heeft. Omdat we willen dat alle bovengenoemde animaties tegelijk uitgevoerd worden, worden ze in een speciale array van de 'Animated' library bewaard die ervoor zorgt dat alle animaties tegelijk worden uitgevoerd.

```

1 Animated.parallel([
2     Animated.timing(
3         this.state.CLanimx,
4         {
5             toValue: ((CurLocPixelX)-initCurrentLocationLeft),
```

```
6         duration: 500,
7     }
8 ),
9 Animated.timing(
10     this.state.CLanimy,
11     {
12         toValue: ((CurLocPixelY)-initCurrentLocationTop),
13         duration: 500,
14     }
15 ),
16 Animated.timing(
17     this.state.NAVanimx1,
18     {
19         toValue: ((navHPixelX)-initHorizontalLineLeft),
20         duration: 500,
21     }
22 ),
23 Animated.timing(
24     this.state.NAVanimy1,
25     {
26         toValue: ((navHPixelY)-initHorizontalLineTop),
27         duration: 500,
28     }
29 ),
30 Animated.timing(
31     this.state.animationHorizontalLineWidth,
32     {
33         toValue : horizontalLineWidth+10, //Grootte berekening horizontale balk,
34             deze mag absoluut zijn
35         duration: 500,
36     })
37 ),
38 Animated.timing(
39     this.state.animationVerticalLineHeight, //Grootte berekening verticale
40         balk, deze mag absoluut zijn
41     {
42         toValue : verticalLineHeight,
43         duration: 500,
44     })
45 ),
46 Animated.timing(
47     this.state.NAVanimx2,
48     {
49         toValue: ((navVPixelX))-initVerticalLineLeft,
50         duration: 500,      // 500ms
51     }
52 ),
53 Animated.timing(           // Animate over time
54     this.state.NAVanimy2, // The animated value to drive
```

```

51    {
52      toValue: ((navVPixelY))-initVerticalLineTop,
53      duration: 500,           // 500ms
54    }
55
56  )
57
58 ]).start();

```

**Listing 5.17:** Parallelle uitvoering van de verschillende animaties.

Wegens de translatie worden alle positiewaarden verminderd met de initiële waarden. De variabelen CurLocPixelX, CurLocPixelY, navHPixelX, navHPixelY, navVPixelX, navVPixelY zijn namelijk variabelen die absoluut ten opzichte van de linkerbovenhoek van het perron in positie berekend worden. De initiële variabelen worden éénmalig de waarden van bovenvermelde variabelen toegekend (zie code 5.19).

De vermindering is dus de waarde die zegt hoeveel pixels getransleerd wordt ten opzichte van de initiële waarde,. Onderstaande code geeft weer hoe de navigatieobjecten opgeroepen worden in de rendering:

```

1 // Huidige locatie
2   navigatieObjects[0]=<Animated.View style={{height:this.state.
3   animationValue2, borderRadius:5, backgroundColor: '#7e879f', width: 10,
4   position: 'absolute', left: initVerticalLineLeft, top:
5   initVerticalLineTop, transform: [{ translateX: this.state.NAVanimx2 },{ 
6   translateY: this.state.NAVanimy2}]}} />;
7 // Horizontale navigatie
8   navigatieObjects[1]=<Animated.View style={{width:this.state.animationValue,
9   borderRadius:5, backgroundColor: '#7e879f', height: 10,position: '
10  absolute',left: initHorizontalLineLeft, top:initHorizontalLineTop,
11  transform: [{ translateX: this.state.NAVanimx1 },{ translateY: this.
12  state.NAVanimy1}]}} />;
13 // Verticale Navigatie
14   navigatieObjects[2]=<Animated.Image source={require('../images/icon.png')} 
15  style={{position: 'absolute',left:initCurrentLocationLeft, top:
16  initCurrentLocationTop,transform: [{ translateX: this.state.CLanimx },{ 
17  translateY: this.state.CLanimy}], flex: 1, width: 20, height: 20,
18  resizeMode: 'contain' }} />;

```

**Listing 5.18:** Navigatieobjecten, volledig gereed voor rendering.

### 5.7.2 Verschijnen van het perron op het scherm

Naast een mooie navigatie, is het voor de overzichtelijkheid ook handig om het perron niet te tonen, en enkel de rijtuigen te tonen. Wanneer vervolgens een trein geselecteerd wordt, zal dit

perron tevoorschijn komen. Dit gebeurt in de functie perroninitialisatie die opgeroepen wordt in het begin van de navigatie (te zien in code 5.16). Op een identieke manier verdwijnt het perron weer wanneer er opnieuw op het aangeduidde rijtuig gedrukt wordt.

```

1  perronInitialisatie ()
2  {
3      initVerticalLineLeft=(navVPixelX);
4      initVerticalLineTop=(navVPixelY);
5      initHorizontalLineLeft=(navHPixelX);
6      initHorizontalLineTop=(navVPixelY);
7      initCurrentLocationTop=(CurLocPixelY);
8      initCurrentLocationLeft=(CurLocPixelX);
9      initHorizontalLineWidth=horizontalLineWidth;
10     this.setState({animationHorizontalLineWidth:new Animated.Value(
11         initHorizontalLineWidth+10)})
12     this.setState({animationVerticalLineHeight:new Animated.Value(
13         verticalLineHeight)})
14     this.forceUpdate() //Forceer update en zorg er hiermee dat de
15     variabelen correct zijn voor het perron verschijnt
16
17     //Perronverschijning
18     Animated.timing(
19         this.state.fadeAnim,
20         {
21             toValue: 1,
22             duration: 500,
23             useNativeDriver: true
24         }
25     ).start();
26     // Als het perron tevoorschijn komt, moet de grootte van de scrollview
27     // vergroten met de grootte van het perron
28     scrollViewHeight=scrollViewInitValue+perronLengtePixels;
29 }
```

**Listing 5.19:** Initialisatie van het perron en de navigatievariabelen

### 5.7.3 Accentueren van het geselecteerde rijtuig



**Figuur 5.18:** Situatie waarin het linker rijtuig is geselecteerd, en de ander rijtuigen vervaagd worden.

Wanneer we een rijtuig aanduiden, willen we duidelijk maken welk rijtuig er geselecteerd is. Dit doen we door een animatie te starten die alle andere rijtuigen buiten het geselecteerde rijtuig

transparanter maken. We willen echter dat dit ook omgekeerd werkt, wanneer we terug op ditzelfde rijtuig klikken, moet alle rijtuigen terug 0% transparant zijn. Dit doen met behulp van enkele hulpvariabelen:

- Ingedrukt: meegegeven argument, zegt welk rijtuig geselecteerd is.
- vorigedrukt: nodig om te vergelijken of de huidige ingedrukte overeenkomt met het rijtuig dat voordien was geselecteerd.

```

1  fade = (ingedrukt, data) =>
2  {
3      // Indien er nog niets in ingedrukt (this.preseed false)
4      if (!this.pressed)
5      {
6          this.pressed=true;
7          //Stek het rijtuig dat ingedrukt moet worden in vorigeIngedrukt.
8          vorigeIngedrukt=ingedrukt;
9          for (let p=0;p<(data).length;p++)
10         {
11             if (p==ingedrukt)
12             {
13                 continue //DIT RIJTUIG IS INGEDRUKT EN WORDT NIET TRANSPARANT
14             }
15             Animated.timing(
16                 imageopacity[p],
17                 {
18                     toValue: 0.5, //ANDERE RIJTUIGEN WORDEN TRANSPARANT
19                 }
20             ).start ();
21         }
22         this.props.onPress(ingedrukt,this.pressed,false)
23     }

```

**Listing 5.20:** Eerste deel van de functieoproep met condities om het perron te laten verschijnen wanneer er geen rijtuig is ingedrukt.

Wanneer een ander rijtuig ingedrukt is, moet het juiste rijtuig oplichten, en het vorige transparanter worden.

```

1
2     else if (this.pressed && ingedrukt!=vorigeIngedrukt)
3     {
4         Animated.parallel([
5             Animated.timing(
6                 imageopacity[vorigeIngedrukt],
7                 {
8                     toValue: 0.5, //TRANSPARANT

```

```

9         }
10        ),
11        Animated.timing(
12          imageopacity[ingedrukt],
13          {
14            toValue: 1.0, //NIET TRANSPARANT
15          }
16        )].start();
17        vorigeIngedrukt=ingedrukt
18        this.props.onPress(ingedrukt,this.pressed,true)
19        //Het derde argument van this.props.onPress wijst erop dat het perron
20        zichtbaar moet blijven.
21      }

```

**Listing 5.21:** Tweede deel van de functieoproep, met condities wanneer er al een rijtuig was ingedrukt.

In andere gevallen is er geen enkel rijtuig geselecteerd en zijn alle rijtuigen niet-transparent.

```

1   else
2   {
3     this.pressed=false;
4     for (let p=0;p<(data).length;p++)
5     {
6       Animated.timing(
7         imageopacity[p],
8         {
9           toValue: 1.0, //NIET TRANSPARANT
10          }
11        ).start();
12      }
13      this.props.onPress(ingedrukt,this.pressed,false)
14    }
15  }
16 }

```

**Listing 5.22:** Derde deel van de functieoproep met condities voor het correct transparant maken en doorgeven naar NavigatieScreen.

Bij het keuzemenu element krijgen we nog een andere situatie. Het kan namelijk zijn dat, wanneer een nieuwe voorkeur wordt ingesteld, opnieuw hetzelfde rijtuig het dichtst bij de huidige locatie staat en deze voorkeur heeft. Omwille van deze reden zal bij dergelijk voorkomen this.props.onPress(ingedrukt,this.pressed,true) onmiddellijk opgeroepen worden, zonder de fade functie op te roepen. Dit is omdat er geen ander rijtuig moet oplichten, aangezien dit hetzelfde rijtuig betreft.

## 5.8 Uitbreidbaarheid

Onze code is zo geïmplementeerd dat deze makkelijk uitbreidbaar is. Zo controleren we bij het renderen van objecten op het perron of deze in beide tabellen (ObjectsSKW en objectTypes) voorkomen. Op deze manier kan er makkelijk nieuwe soorten objecten in de databank toegevoegd worden, zonder onze code hiervoor te moeten aanpassen. Vervolgens wordt er gecontroleerd of er een afbeelding bestaat van dit type object om als icoon te tonen. Hier moet wel een kanttekening gemaakt worden: dynamische toekenning van een imagebron aan een image tag is niet mogelijk in React Native. Daarom schrijven we de iconen van alle objecten in het objImages object in het constants.js bestand (zie bijlage A.1.1). Op deze manier zal men enkel dit bestand moeten aanpassen om nieuwe objecten toe te voegen waarvan men een icoon wil.

Een gelijkaardige implementatie is te vinden bij het renderen van rijtuigen en zijn miniatuurweergave. Er wordt gekeken of er afbeeldingen van het specifieke subType zijn. Door te checken of deze zich bevinden in een constante met alle subtypes en zijn afbeeldingen in (zie bijlage A.1.1). Indien niet, wordt er gecontroleerd of er specifieke afbeeldingen zijn van de lengte van het rijtuig, en indien niet, wordt gebruikgemaakt van algemene afbeeldingen. Zo kan zelfs wanneer er geen op maat gemaakte afbeeldingen van het rijtuig zijn, er toch een visualisatie verschijnen.

Ook voor de rendering van de opties afbeeldingen is er uitbreidbaarheid toegepast. Eerst kijkt men welke opties er zijn door in het object van het rijtuig de functie matchKey op te roepen met keyToFind='lastPlanned\_materialUnits', die zoekt vervolgens naar properties die starten met 'lastPlanned\_materialUnits'. Dan wordt er voor elk van deze gecheckt of deze zich in het featureImages object (te vinden in bijlage A.1.1) bevindt.

```

1 function matchKey (objectToSearch, keyToFind) {
2   let echteArray=[];
3   let i=0;
4   for (let k in objectToSearch) {
5     if ( k.startsWith(keyToFind))
6     {
7       echteArray [i]=k;
8       i++;
9     }
10   }
11   return echteArray;
12 }
```

**Listing 5.23:** Functie waarbij properties van het object ObjectToSearch worden doorzocht op het argument keyToFind, om zo te weten over welke opties een rijtuig bezit.

## 5.9 Conclusie

In dit hoofdstuk hebben we scherm per scherm overlopen wat de geschreven code van onze applicatie inhield. We moeten eerst verschillende componenten renderen voor we de navigatiebepaling beginnen. Om een correcte navigatiebepaling te creëren, moeten we ook randvoorwaarden opleggen die controleren of we wel op het perron staan. We controleren of de locatie op het perron ligt of de nauwkeurigheidscirkel hiervan tot op het juiste perron komt. De navigatiebepaling wordt vervolgens gedaan in pixelcoördinaten aan de hand van 2 loodrechte lijnen die een navigatiepad voorstellen, met mogelijks 2 extra lijnen wanneer er objecten van het perron op ons pad liggen. Voor animaties moeten we deze zodanig implementeren dat er geen animaties worden gedaan wanneer ze niet zichtbaar zijn.

# **Hoofdstuk 6**

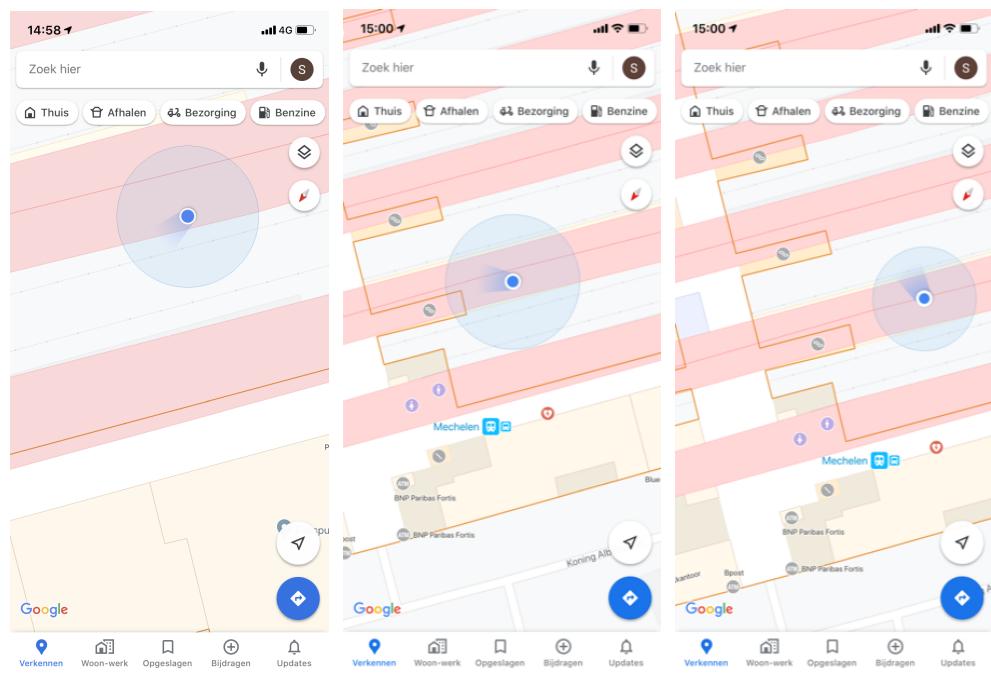
## **Resultaten**

### **6.1 Testing**

Om de goede werking van onze applicatie te verzekeren, hebben we voortdurend testen uitgevoerd. We beschrijven hier de belangrijkste testen.

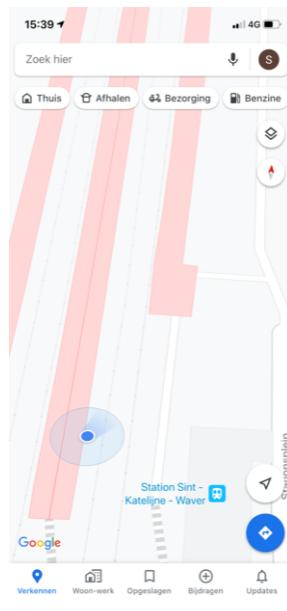
#### **6.1.1 Reallife test: Mechelen**

Oorspronkelijk was onze applicatie bedoeld voor het station in Mechelen. Bij uittesten bleek echter dat de locatiebepaling hier niet ideaal is. Onderstaande afbeeldingen tonen de locatiebepaling bij het perron van spoor 1 (onderste perron), we zien dat zelfs de nauwkeurigheids cirkel hier niet tot aan het perron geraakt. De locaties werden uitgetest met 3 smartphones, waarvan 2 iPhones en 1 Android gsm, met gelijkaardige resultaten. Dit bevestigt dat de oorzaak niet aan een slecht gekalibreerde smartphone ligt. Mogelijke oorzaak hiervan kan de verstoring van het signaal door de omgeving en weercondities zijn [34].



**Figuur 6.1:** Uittesten van de applicatie in Mechelen, 8 mei 2020, om 15:00. We bevonden ons op dit moment op het onderste perron (rode rechthoek).

Hierna werden real time testen gedaan op het perron van Sint-Katelijne-Waver, met meer succes. Op onderstaande foto is te zien hoe het locatiesignaal correct in de buurt is van het perron van spoor 2/3. Het locatiepunt bevindt zich echter juist wel of niet op het perron.



**Figuur 6.2:** Uittesten van de applicatie in Sint-Katelijne-Waver, 8 mei 2020, om 15:39. We bevonden ons op dit moment op het middelste perron.

Uit de testen bleek dat onze locatiebepaling en navigatie van onze applicatie zelf werkte, maar dat de locatiebepaling van de omgeving door de smartphone de applicatie verhinderde correct te werken. Volgende aanpassingen werden hiervoor gedaan:

- De perrons van Sint-Katelijne-Waver werden vanaf dit moment gebruikt als testperrons voor de praktische realisatie. De tabel 'TracksSKW' met geografische coördinaten van deze perrons werd gecreëerd als vervanging van de tabel 'TracksMechelen'.
- Er werd een extra if-conditie en vergelijking toegevoegd dat ervoor zorgt, wanneer de huidige locatie zich niet op het perron bevindt (maar de nauwkeurigheidscirkel wel tot hier reikt), er een verticale balk wordt weergegeven die de huidige locatie voorstelt.

### 6.1.2 Reallife test: Sint-Katelijne-Waver

Na bovenstaande aanpassingen, werden meerdere testen op de perrons van Sint-Katelijne-Waver gedaan. Hier bleek echter dat soms de nauwkeurigheid nog altijd te wensen overliet, zelfs met een if-conditie waar de nauwkeurigheidscirkel in verwerkt zit. Om te voorkomen dat door slechte nauwkeurigheid de navigatie soms zou wegvalLEN, werd de code aangepast. Wanneer er eenmalig een locatie op het perron wordt gevonden (al dan niet met de vergelijking voor de nauwkeurigheidscirkel), wordt de vergelijking van de nauwkeurigheidscirkel overgeslagen. Dit betekent dat na deze eerste keer, wanneer de locatie niet op het perron berekend is, er automatisch naar de verticale balk wordt overgegaan. Er wordt dus niet meer getest of de huidige locatie +

nauwkeurigheidscirkel zich op het perron zou kunnen bevinden. Dit maakt de locatiebeweging vloeinder, maar kan ook wel voor meer fouten zorgen. Uit de testen bleek echter dat de locatieberekeningen die zich niet op het perron bevinden, wel parallel meelopen als we stappen. De kans dat men zich op het verkeerde perron bevindt zonder dat hier een melding van werd gegeven, wordt echter wel groter.

### 6.1.3 Overige testen

Verder werden volgende testen uitgevoerd:

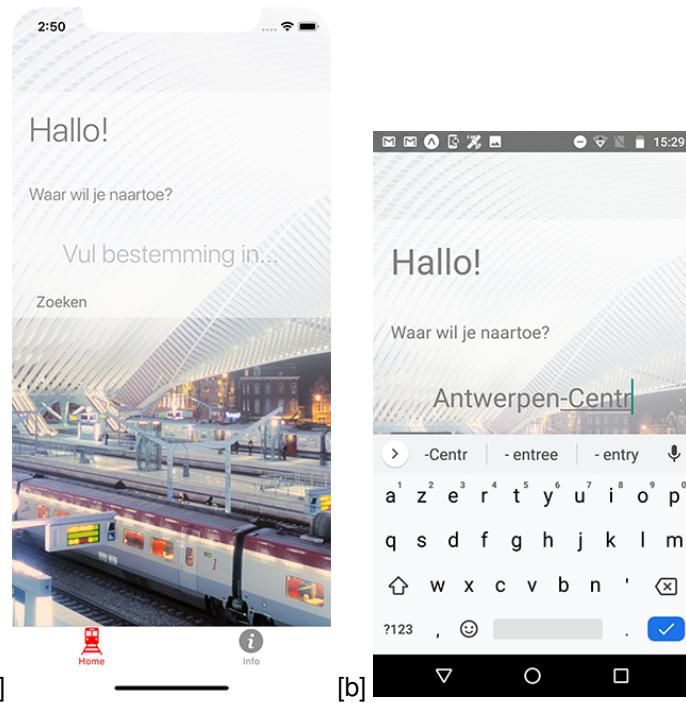
- Correcte overeenkomst van beginpunt en eindpunt perron op het scherm.
- Overeenkomst perronobjecten met de objecten die op het perron staan.
- Foutmeldingen bij locatie op het perron, verkeerde bestemmingsinvoer, ...
- Correcte werking scrollview.
- ...

## 6.2 Visueel eindresultaat van de applicatie

Nu we al onze elementen specifiek besproken hebben, is het tijd om het resultaat van onze applicatie te bewonderen. Onze applicatie werkt zowel voor Android als iOS en we zullen van beide schermafbeeldingen plaatsen.

### 6.2.1 HomeScreen

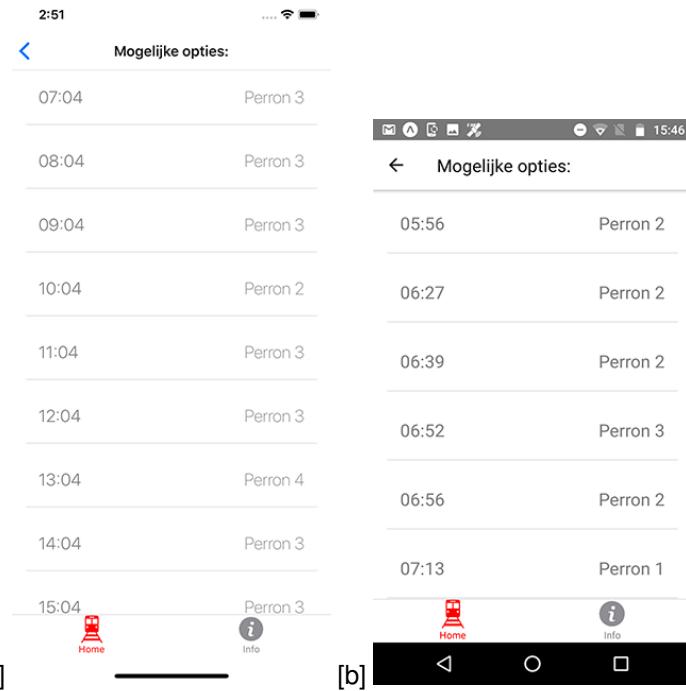
Het homescreen is eenvoudig gehouden. Omdat we enkel data van 1 station hebben, is het vertrekpunt altijd hetzelfde. De eindbestemming is echter vrij te kiezen.



**Figuur 6.3:** Welkomstscherm in een (a) iOS omgeving en (b) Android omgeving, waar iets wordt ingetypt als eindbestemming.

### 6.2.2 ChoiceScreen

In het keuzescherm wordt elke mogelijke treinoptie naar de eindbestemming per vertrekuur en vertrekperron weergegeven, geordend op vertrekuur.



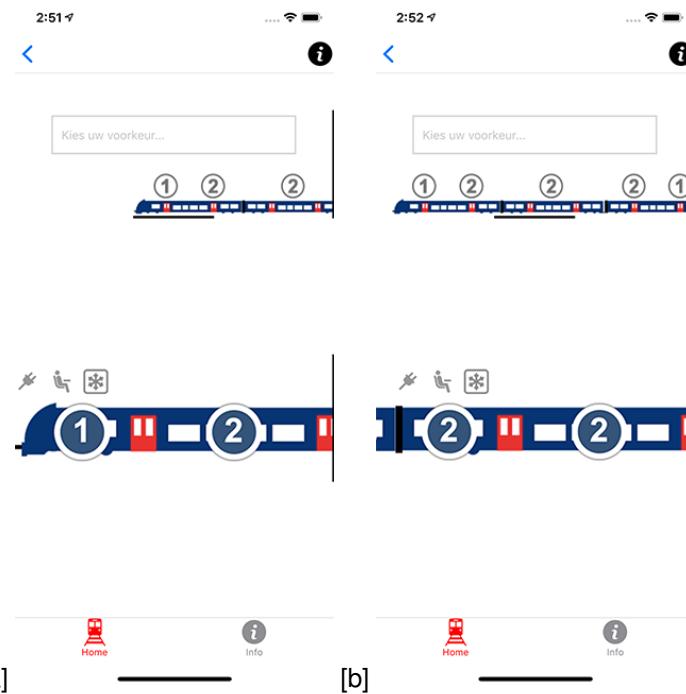
**Figuur 6.4:** Keuzescherm in een (a) iOS omgeving en (b) Android omgeving.

### 6.2.3 NavigationScreen

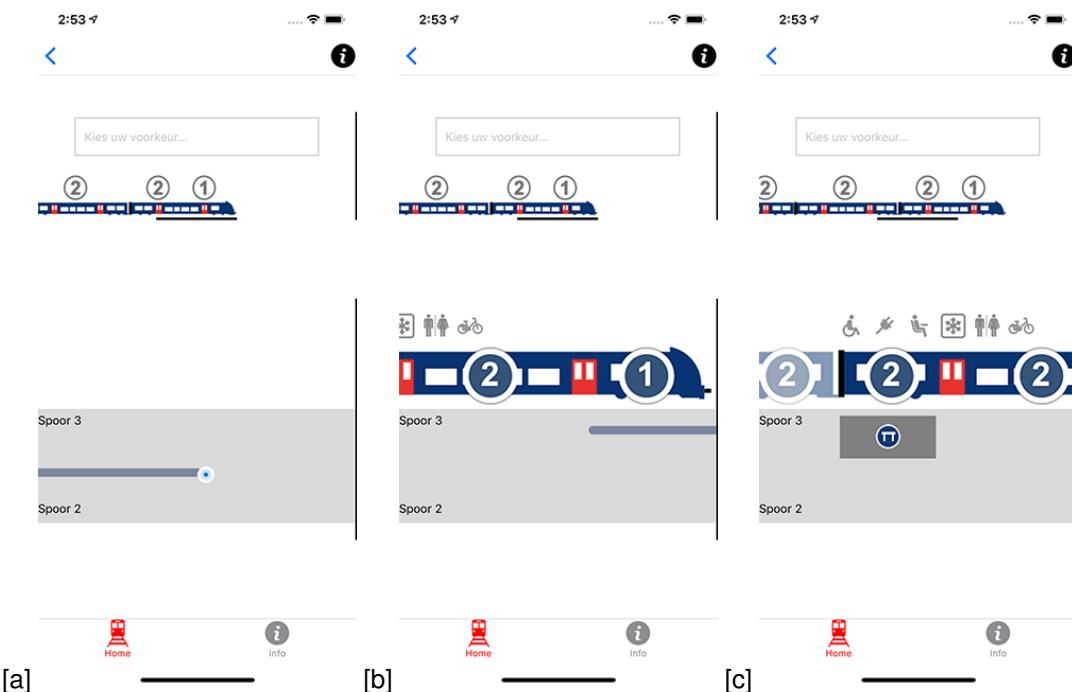
Het navigatiescherm begint met de vertoning van het keuzemenu, de miniatuurweergave van de trein en de grotere weergave van de trein. Boven de trein worden opties van elk rijtuig weergegeven. Bij aanduiden licht het rijtuig op en wordt er gescrolld waar de huidige locatie zich bevindt.

Er is ook de optie om een voorkeur op te geven via het keuzemenu bovenaan. Verder wordt, wanneer de locatie zich niet op het perron bevindt, maar mogelijk toch nog op het perron ligt (uitgerekend met de nauwkeurigheidscirkel), de locatie weergegeven met een verticale balk die de positie op het perron ongeveer weergeeft. Wanneer de locatie binnen de nauwkeurigheidscirkel zich onmogelijk op het perron kan bevinden, komt de melding tevoorschijn dat de locatie zich niet op het perron bevindt.

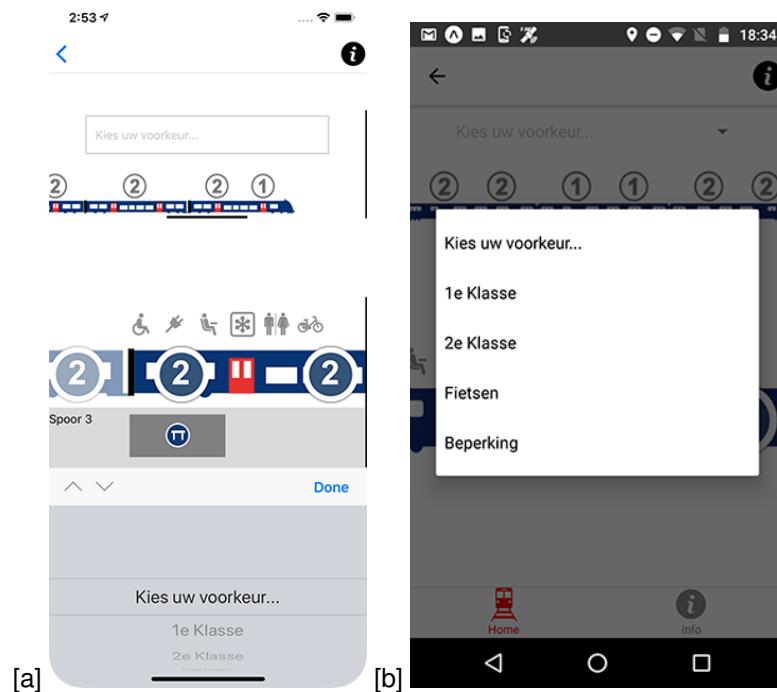
Door op de info button rechts bovenaan te klikken wordt het modal screen geopend.



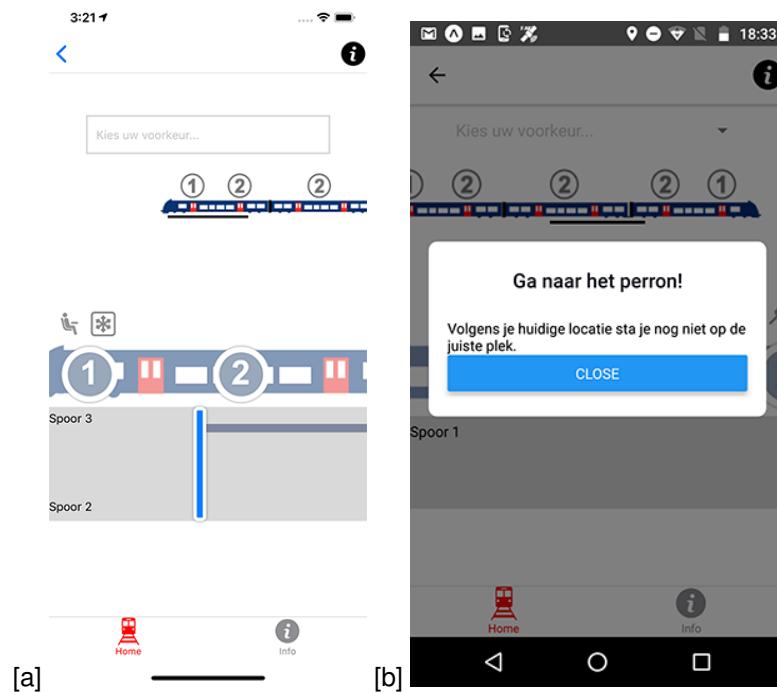
**Figuur 6.5:** Bij het navigeren naar dit scherm wordt het begin van de trein weergegeven (a), als we scrollen, scrollt de miniatuurweergave mee, idem in de omgekeerde richting (b).



**Figuur 6.6:** Na selectie wordt er automatisch naar een locatie op het perron gescrolld die bepaald is zodanig dat de huidige locatie in het midden staat (a). Het eindpunt van de locatie is de deur dat het dichtst bij de huidige locatie staat.



**Figuur 6.7:** Het keuzemenu in (a) iOS en in (b) Android.



**Figuur 6.8:** (a) Locatiebalk wanneer de locatie zich net niet op het perron bevindt en dit mogelijk een nauwkeurigheidsfout is. (b) Melding dat de gebruiker zich niet op de juiste positie op het perron bevindt.

2:52 4G ... ☰

Meer info

Lengte: 3 rijtuigen

Type trein: AM08M\_a

In deze trein:

airco  
voorziening voor invaliden  
toiletvoorziening  
fietsenvoorziening

Sluiten

**Figuur 6.9:** Geopende modalscreen met informatie over wat de lengte en wat de algemene voorzieningen zijn in de trein.

#### 6.2.3.1 EndScreen

Het eindscherm komt tevoorschijn wanneer we op de juiste positie van het perron terechtkomen.



**Figuur 6.10:** Het eindscherm, dat tevoorschijn komt bij correcte positonering.

### 6.2.3.2 InfoScreen

Het InfoScreen bevat de logo's van het meewerkende bedrijf en de meewerkende universiteit, alsook de naam van de ontwikkelaar.

16:46 ↗  
◀ Zoek ⌂



Stijn De Bels



**Figuur 6.11:** Het infoscreen.

# **Hoofdstuk 7**

## **Besluit**

In deze masterproef werd gekeken naar de mogelijkheid om navigatiebepaling op een perron te implementeren in een mobiele applicatie en in welke mate dit de bediening van de reiziger verbetert. Op volgende stellingen werd een antwoord gegeven:

1. Is het mogelijk om aan de hand van een smartphonelocatie de reiziger naar de juiste plaats op het perron te begeleiden?
2. Is het mogelijk om de locatie van een treinstel en rijtuig aan het perron vooraf te bepalen?
3. Is de huidige locatie van de gebruiker exact genoeg om de applicatie te laten werken?
4. Bevat de databank genoeg informatie over de exacte samenstelling van de trein om de verschillende categorieën rijtuigen weer te geven?

Om de eerste stelling te kunnen beantwoorden, werd eerst onderzocht welke elementen nodig zijn om een navigatie op het perron weer te geven. Hiervoor werden verschillende prototypes gemaakt. Hieruit bleek dat er nood is aan referentiepunten zodat de gebruiker correct kan navigeren. In de databank werd daarom dummy data van de geografische coördinaten van de hoeken van het perron en de objecten gecreëerd. Omzettingen van geografische coördinaten naar pixels werden toegevoegd om zo het perron en de objecten correct op de applicatie weer te geven. Ook werden verschillende opties om de navigatie te realiseren besproken en werd hier een uitbreiding op gemaakt om te vermijden dat het navigatiepad objecten kruist. We kunnen dus besluiten dat het inderdaad mogelijk is.

Aan de hand van data uit de databank kon bepaald worden waar rijtuigen zich bevinden ten opzichte van de trein. Ook werd een functie geschreven die de locatie van het volledige treinstel aan het perron bepaalde. Echter was de data om deze locatie te bepalen niet beschikbaar en werd een vervangfunctie geschreven. We kunnen de tweede stelling dus positief beantwoorden, op voorwaarde dat we toegang hebben tot deze data.

Voor de derde stelling bleek na het testen van de applicatie in Mechelen dat locatiebepaling niet correct genoeg is om de applicatie te laten werken. Er werd om deze reden een extra functie aan de navigatiebepaling toegevoegd, die navigatie blijft weergeven indien de nauwkeurigheidscirkel van de locatie zich tot op het perron begeeft. We konden in Mechelen echter vaststellen dat zelfs met de nauwkeurigheidscirkel de locatie er hopeloos naast zat. We kunnen dus besluiten dat er in sommige stations nood zal zijn aan voorzieningen die de locatiebepaling verbetert. Zo is een veelgebruikte verbeteringsmogelijkheid het installeren van Bluetooth Low Energy (BLE) beacons. BLE beacons zijn kleine draadloze toestellen die op verschillende plaatsen geïnstalleerd kunnen worden. De positie van deze beacon is bekend. Wanneer een smartphone hiermee verbindt, wordt een schatting gemaakt van de afstand tot deze beacon. Op deze manier wordt een locatie bepaald. Deze techniek wordt al in drukbezochte plaatsen toegepast, zoals bijvoorbeeld London's Gatwick Airport [35].

Aan de hand van de commercialTrainCompositions tabel en de plannen die de NMBS ter beschikking stelde, konden er verschillende afbeeldingen gecreëerd worden voor de verschillende soorten rijtuigen. Er werden afbeeldingen gecreëerd gebaseerd op het type rijtuig, de aanwezigheid van 1e of 2e klasse en de lengte van het rijtuig. Omdat wegens tijdsgebrek niet alle types en lengtes werden aangemaakt, werden ook algemene rijtuig afbeeldingen van trainMap opgenomen in de afbeeldingscollectie. We kunnen dus besluiten dat aan de vierde stelling is voldaan.

## 7.1 Future work

Hoewel aan de meeste stellingen beantwoord werd, is er zeker nog ruimte voor verbetering alvorens de applicatie publiek zal worden gesteld. Zo werd er in deze applicatie weinig tot geen rekening gehouden met beveiliging van data en applicatie. Een interessante onderzoeksroute zou kunnen zijn in welke mate onze applicatie kwetsbaar is voor hacking.

Ook programmeertechisch zijn er enkele verbeteringen mogelijk: zo is communicatie tussen de train component en het navigatiescherm weinig fraais. De keuze om de train component en navigatiescherm te splitsen zorgde ervoor dat er voortdurend argumenten heen-en-weer moeten worden gestuurd. Een mogelijkheid om het implementeren van een globaal geheugen (zoals Redux) kan geopperd worden. Onze keuze om dit niet te doen werd genomen uit de filosofie dat we werken met een StackNavigator waardoor onze data enkel naar het opeenvolgende scherm moet worden doorgegeven. De voorgaande schermen maken dus nooit gebruik van data van latere schermen. De train component gooide naarmate de tijd vorderde echter roet in het eten.

Hoewel er wel degelijk werd nagedacht over uitbreidbaarheid in onze applicatie, zijn ook hier nog enkele zwakke punten: om nieuwe types afbeeldingen toe te voegen moeten we de constants.js file aanpassen. Dit is omdat React Native geen dynamische toekenning van local files aan de <Image> tag toelaat. Het is een interessante piste om te zien in welke mate de mogelijkheid er is

om afbeeldingen op een online server te plaatsen voor dynamische toekenning, en wat hierin het effect is op de performantie. Ook kan onderzocht worden of er andere programmeeromgevingen zijn die wel dynamische toekenningen toelaten.

De weergave van het perron wordt enkel rechthoekig gegeven, echter zijn sommige perrons in stations niet rechthoekig. We kunnen onderzoeken in welke mate dit invloed zal hebben op het herschrijven van onze functie en welke extra data we hiervoor zullen moeten implementeren.

In de applicatie werken we met dummy data, het zou interessant zijn om met realtime data te werken, om zo meldingen te kunnen geven indien het spoor veranderd is of er een vertraging is.

# Bibliografie

- [1] Van Looveren L. Applicatieontwikkeling in cross-platform en platform-specifieke frameworks. master thesis, KU Leuven Campus Geel, 2017 - 2018.
- [2] Hermes D. *Xamarin Mobile Application Development*. Apress Media LLC, 2015.
- [3] Abhishek N. Ashkat P. *React Native for Mobile Development*. Apress Media LLC, 2019.
- [4] Frachet M. Understanding the react native bridge concept, 2017.
- [5] WeAreSocial. Digital in 2018 in belgium, 2018.
- [6] MobielVlaanderen. Het gebruik van vervoermiddelen, 2018.
- [7] Sheppard D. *Beginning Hybrid Mobile Application Development*. Apress Media LLC, 2016.
- [8] Wikipedia. Windows phone — Wikipedia, the free encyclopedia, 2004.
- [9] Armour B. 5 key benefits of native mobile app development, 2018.
- [10] Clayton J. Securing hybrid mobile applications, 2019.
- [11] Gillis A. Rouse M. 5 key benefits of native mobile app development, 2018.
- [12] Charland A. and Leroux B. Mobile application development: Web vs. native. *Communications of the ACM*, 54(5):49–53, 2011.
- [13] Stringfellow A. What is web application architecture? how it works, trends, best practices and more, 2017.
- [14] Verbeeck G. Voordelen van webgebaseerde applicaties, 2017.
- [15] Giona C. Experience and comparison between native and hybrid development approaches for mobile devices. master thesis, Politecnico Di Milano, 2013 – 2014.
- [16] Fox L. Advantages and disadvantages – web apps, 2017.
- [17] Wikipedia. Cross-site scripting — Wikipedia, the free encyclopedia, 2004. [Online; accessed 29-November-2019].

- [18] Crookes D. Progressive web apps. *Web User*, (452):38–39, 2018.
- [19] Gaunt M. Google developers. service workers: an introduction — web fundamentals. [Online; accessed 29-November-2019].
- [20] Gaunt M. and Kinlan P. Google developers. the web app manifest — web fundamentals. [Online; accessed 29-November-2019].
- [21] Sheppard D. *Beginning Progressive Web App Development*. Apress Media LLC, 2017.
- [22] Kho N. Everything you need to know about: Progressive web apps. *EContent*, 41(2):20–24, 2018.
- [23] Oragui D. Should you consider investing in a progressive web app?, 2018.
- [24] Hoober S. Mobile apps: Native, hybrid, and webviews, 2018.
- [25] Korolev S. Mobile app development approaches explained, 2019.
- [26] Rickard C. Choosing the right mobile app for your project: Native vs cross-platform vs hybrid, 2016.
- [27] Wikipedia. Xamarin — Wikipedia, the free encyclopedia, 2004. [Online; accessed 12-December-2019].
- [28] Zammetti F. *Practical React Native*. Apress Media LLC, 2018.
- [29] Gackenheimer C. *Introduction to React*. Apress Media LLC, 2015.
- [30] Cool T. The virtual dom, 2019.
- [31] Crauwels H. Vennekens J. *Objectgericht Programmeren 2*. KU Leuven, 2014.
- [32] Zammetti F. *Practical Flutter*. Apress Media LLC, 2019.
- [33] Kopec D. *Dart for Absolute Beginners*. Apress Media LLC, 2014.
- [34] Sensolus. Four geolocation technologies compared: How can they improve your operational efficiency?, 2019.
- [35] Marcel J. A new way to navigate, 2017.

# Bijlage A

## Broncode

### A.1 Javascript bestanden

Hierin bevinden zich enkele Javascript bestanden waarheen verwezen is in de tekst. De volledige code is te vinden op de volgende link: <https://tinyurl.com/MPStijnDeBels>.

#### A.1.1 constants.js

```
1 export const meterNrPixelOmzetting = (500/27);
2 export const pixelNrMiniatuurPixelOmzetting = (127/500);
3
4 export const images = {
5     AM08M_a: require('./images/AM08M_a.png'),
6     AM08M_b: require('./images/AM08M_b.png'),
7     AM08M_c: require('./images/AM08M_c.png'),
8     AM08P_a: require('./images/AM08M_a.png'),
9     AM08P_b: require('./images/AM08M_b.png'),
10    AM08P_c: require('./images/AM08M_c.png')
11};
12
13 export const imagesSMALL = {
14     AM08M_a: require('./images/AM08M_aSMALL.png'),
15     AM08M_b: require('./images/AM08M_bSMALL.png'),
16     AM08M_c: require('./images/AM08M_cSMALL.png'),
17     AM08P_a: require('./images/AM08M_aSMALL.png'),
18     AM08P_b: require('./images/AM08M_bSMALL.png'),
19     AM08P_c: require('./images/AM08M_cSMALL.png')
20};
21
22 export const objImages = {
23     STAIRS: require('./images/ObjSTAIRS.png'),
```

```
24     ROOF: require('./images/ObjROOF.png')
25 };
26
27 export const featureImages = {
28     lastPlanned_materialUnits_hasBikeSection: require('./images/Pictogram_fiets.
29         .png'),
30     lastPlanned_materialUnits_hasToilets: require('./images/Pictogram_toilet.
31         .png'),
32     lastPlanned_materialUnits_hasAirco: require('./images/Pictogram_airco.png')
33     ,
34     lastPlanned_materialUnits_hasTables: require('./images/Pictogram_tafel.png'
35         ),
36     lastPlanned_materialUnits_hasSecondClassOutlets: require('./images/
37         Pictogram_stopcontact.png'),
38     lastPlanned_materialUnits_hasPrmSection: require('./images/
39         Pictogram_rolstoel.png')
40 };
41
42
43
44 export const featureText = {
45     lastPlanned_materialUnits_hasBikeSection: "fietsenvoorziening",
46     lastPlanned_materialUnits_hasToilets: "toiletvoorziening",
47     lastPlanned_materialUnits_hasAirco: "airco",
48     lastPlanned_materialUnits_hasPrmSection: "voorziening voor invaliden"
49 };
50
51
52
53
54
55
56
57 };
58
59
60
61
62
63
64 
```

```

65 "train-lastC1": require('./images/train-lastC1-27.png'),
66 "train-firstC12": require('./images/train-firstC12-27.png'),
67 "train-firstC21": require('./images/train-firstC21-27.png'),
68 "train-lastC21": require('./images/train-lastC21-27.png'),
69 "train-lastC12": require('./images/train-lastC12-27.png'),
70 "train-lastnone": require('./images/train-lastnone.png'),
71 "train-firstnone": require('./images/train-firstnone.png')
72 };
73 export const imageLengths = {
74   27000: images27,
75   general: imagesGeneral
76 };

```

## A.1.2 functions.js

```

1  export function matchKey(objectToSearch, keyToFind) {
2    let echteArray=[];
3    let i=0;
4    for (let k in objectToSearch) {
5      if ( k.startsWith(keyToFind))
6      {
7        echteArray[i]=k;
8        i++;
9      }
10    }
11    return echteArray;
12  }
13
14 export function rijtuigBepaling(prop, data, key, typeAfbeelding,
15   rijtuigLengteAfbeelding)
16 {
17   let filename=0;
18   let noTouch=false
19   if (typeAfbeelding[prop.lastPlanned_materialUnits_materialSubTypeName])
20   {
21     filename=typeAfbeelding[prop.
22       lastPlanned_materialUnits_materialSubTypeName]
23   }
24   else
25   {
26     let argument="";
27
28     if (prop.lastPlanned_materialUnits_isFirstClass=='WAAR' && prop.
29       lastPlanned_materialUnits_isSecondClass=='ONWAAR' && (key==0 || ((key
30       !=(data).length && key!=0) && (prop.

```

```
        lastPlanned_materialUnits_tractionPosition!=data[key-1].
        lastPlanned_materialUnits_tractionPosition)))))
27    {
28
29        argument="train-firstC1"
30    }
31    else if (prop.lastPlanned_materialUnits_isFirstClass=='WAAR' && prop.
            lastPlanned_materialUnits_isSecondClass=='ONWAAR' && (key==(data).
            length-1 || ((key+1!=(data).length) && (prop.
            lastPlanned_materialUnits_tractionPosition!=data[key+1].
            lastPlanned_materialUnits_tractionPosition))))
32    {
33
34        argument='train-lastC1'
35    }
36    else if (prop.lastPlanned_materialUnits_isFirstClass=='ONWAAR' && prop.
            lastPlanned_materialUnits_isSecondClass=='WAAR' && (key==0 || ((key!=(
            data).length && key!=0) && (prop.
            lastPlanned_materialUnits_tractionPosition!=data[key-1].
            lastPlanned_materialUnits_tractionPosition))))
37    {
38        console.log("hoi")
39        argument='train-firstC2'
40
41    }
42    else if (prop.lastPlanned_materialUnits_isFirstClass=='ONWAAR' && prop.
            lastPlanned_materialUnits_isSecondClass=='WAAR' && (key==(data).length
            -1 || ((key+1!=(data).length) && (prop.
            lastPlanned_materialUnits_tractionPosition!=data[key+1].
            lastPlanned_materialUnits_tractionPosition))))
43    {
44        argument='train-lastC2'
45    }
46    else if (prop.lastPlanned_materialUnits_isFirstClass=='ONWAAR' && prop.
            lastPlanned_materialUnits_isSecondClass=='ONWAAR' && (key==0 || ((key
            !=(data).length && key!=0) && (prop.
            lastPlanned_materialUnits_tractionPosition!=data[key-1].
            lastPlanned_materialUnits_tractionPosition))))
47    {
48        argument='train-firstnone'
49        noTouch=true;
50    }
51    else if (prop.lastPlanned_materialUnits_isFirstClass=='ONWAAR' && prop.
            lastPlanned_materialUnits_isSecondClass=='ONWAAR' && (key==(data).
            length-1 || ((key+1!=(data).length) && (prop.
            lastPlanned_materialUnits_tractionPosition!=data[key+1].
            lastPlanned_materialUnits_tractionPosition))))
```

```
52  {
53      argument='train-lastnone'
54      noTouch=true;
55  }
56  else if (prop.lastPlanned_materialUnits_isFirstClass=='WAAR' && prop.
57      lastPlanned_materialUnits_isSecondClass=='WAAR' && (key==0 || ((key!=
58      data).length && key!=0) && (prop.
59      lastPlanned_materialUnits_tractionPosition!=data[key-1].
60      lastPlanned_materialUnits_tractionPosition)))
61  {
62      argument='train-firstC12'
63  }
64  else if (prop.lastPlanned_materialUnits_isFirstClass=='WAAR' && prop.
65      lastPlanned_materialUnits_isSecondClass=='WAAR' && (key==(data).length
66      -1 || (key+1!=(data).length) && (prop.
67      lastPlanned_materialUnits_tractionPosition!=data[key+1].
68      lastPlanned_materialUnits_tractionPosition)))
69  {
70      argument='train-lastC21'
71  }
72  else if (prop.lastPlanned_materialUnits_isFirstClass=='WAAR')
73  {
74      argument='train-C1'
75  }
76  else if (prop.lastPlanned_materialUnits_isSecondClass=='WAAR')
77  {
78      argument='train-C2'
79  }
80  if (argument)
81  {
82      //console.log(argument)
83      if((rijtuigLengteAfbeelding[((data[key].Length))]!=undefined) && (
84          argument in rijtuigLengteAfbeelding[((data[key].Length))])
85      {
86          //console.log("hier")
87          //console.log(rijtuigLengteAfbeelding[((data[key].Length))][
88              argument])
89          filename=rijtuigLengteAfbeelding[((data[key].Length))][argument];
90      }
91  else
92  {
93      filename=rijtuigLengteAfbeelding["general"][argument]
94  }
```

```

89     }
90   else
91   {
92     return;
93   }
94 }
95 return [filename, noTouch]
96 }
97
98 export function JSONtodata (specifics)
99 {
100 let i=0;
101 var specificsA;
102 var specificsAr = [specifics.length];
103 for(let i = 0; i < specifics.length; i++)
104 {
105   //console.log(i);
106   if (specifics === null) {
107     specifics = [ ];
108   }
109   else
110   {
111     specificsA=specifics[i];
112     specificsA = JSON.parse(specificsA);
113     specificsAr[i]=specificsA;
114     //console.log(specificsAr);
115   }
116 }
117 return specificsAr
118 }
```

### A.1.3 Navigatieberekening eerste optie: berekening met geografische coördinaten

```

1 berekenNavigatie(j)
2 {
3   /* hieruit komt: navVPixelX,navVPixelY,navHPixelX,navVPixelY,
4   horizontalLineWidth,verticalLineHeight*/
5   navVGeoX=(parseFloat(ricoLengte)*parseFloat(this.state.location.coords.
6     longitude)-parseFloat(ricoBreedte)*parseFloat(this.state.listTrack
7     [0]["LB-LONG"])-parseFloat(this.state.location.coords.latitude)+
8     parseFloat(this.state.listTrack[0]["LB-LAT"])-parseFloat(
9     offsetwalkperron))/(parseFloat(ricoLengte)-parseFloat(ricoBreedte))
10  navVGeoY=ricoLengte*parseFloat(navVGeoX)-ricoLengte*parseFloat(this.
11    state.location.coords.longitude)+parseFloat(this.state.location.
12    coords.latitude);
13  [navVPixelX, navVPixelY]=this.geoToPixelBerekening(navVGeoX,navVGeoY);
```

```
7     verticalLineHeight=this.geoToPixelAfstandsBerekening(navVGeoY, navVGeoX
8         , this.state.location.coords.latitude, this.state.location.coords.
9             longitude) //in pixels
10    totalHLengthTillEndNavigation=0;
11    for(let i = 0; i < (this.state.listTrainSpecifics).length; i++)
12    {
13        if (this.state.listTrainOffsets[i])
14        {
15            if (j==i)
16            {
17                totalHLengthTillEndNavigation+=parseFloat(this.state.
18                    listTrainOffsets[i].Door1)/1000
19                break;
20            }
21        }
22    }
23    else
24    {
25        if (j==i)
26        {
27            totalHLengthTillEndNavigation+=2
28            break;
29        }
30        else
31        {
32            totalHLengthTillEndNavigation+=parseFloat(this.state.
33                listTrainSpecifics[i].lastPlanned_materialunits_lengthInMeter
34                )
35        }
36    }
37    totalHLengthTillEndNavigation=(parseFloat(totalHLengthTillEndNavigation
38        )/27)*500
39    totalHLengthTillEndNavigation+=parseFloat(treinOffsetPixels) //afstand
40        in pixels
41    //console.log(treinOffsetPixels)
42    verhoudingPuntTotPerron=(totalHLengthTillEndNavigation/
43        perronBreedtePixels)
44    horizontalLineGeoWidth=(verhoudingPuntTotPerron)*perronBreedteGeo
45    horizontalLineGeoWidth-=(navVPixelX/perronBreedtePixels)*
46        perronBreedteGeo //afstand in geocoordinaatn
```

```

43     if (horizontalLineGeoWidth<0 & ((this.state.listTrack[0]["LB-LAT"])-
44         parseFloat(this.state.listTrack[0]["LO-LAT"]))>0)
45     {
46         navHGeoX=-Math.sqrt((Math.pow(parseFloat(horizontalLineGeoWidth),2)-
47             (Math.pow(parseFloat(ricoBreedte),2)+1))+parseFloat(navVGeoX))
48     }
49     else
50     {
51         navHGeoX=Math.sqrt((Math.pow(parseFloat(horizontalLineGeoWidth),2))/(
52             Math.pow(parseFloat(ricoBreedte),2)+1))+parseFloat(navVGeoX)
53     }
54     navHGeoY=ricoBreedte*(navHGeoX-navVGeoX)+navVGeoY
55     horizontalLineWidth=this.geoToPixelAfstandsBerekening(navHGeoY,navHGeoX
56         ,navVGeoY,navVGeoX);
57     [navHPixelX,navHPixelY]=this.geoToPixelBerekening(navHGeoX,navHGeoY);
58     if (horizontalLineGeoWidth>0)
59     {
60         navHPixelX=navVPixelX
61     }
62     else{
63         navHPixelX=navHPixelX
64     }
65 }
```

## A.2 PHP bestanden

### A.2.1 DBConfig.php

Enkele elementen zijn hier vervangen door het woord 'AFGESCHERMD'.

```

1 <?php
2
3 //Define your host here.
4 $HostName = "AFGESCHERMD.be.mysql";
5
6 //Define your database name here.
7 $DatabaseName = "betrainnnmbs";
8
9 //Define your database username here.
10 $HostUser = "betrainnnmbs";
11
12 //Define your database password here.
13 $HostPass = "AFGESCHERMD";
```

```
14
15    ?>
```

## A.2.2 get\_trains\_info.php

```
1 <?php
2
3     // Importing DBConfig.php file.
4     include 'DBConfig.php';
5
6     // Creating connection.
7     $con = mysqli_connect($HostName,$HostUser,$HostPass,$DatabaseName);
8     // Getting the received JSON into $json variable.
9     $json = file_get_contents('php://input');
10
11    // decoding the received JSON and store into $obj variable.
12    $obj = json_decode($json,true);
13
14    $destination = $obj['Destination'];
15
16    // Creating SQL query and insert the record into MySQL database table.
17    $Sql_Query = "select list_id from ptcars where list_commercialLongNameDutch
18        ='$destination';";
19    $result = mysqli_query($con, $Sql_Query);
20    $json=[];
21    $nr=0;
22    $row = mysqli_fetch_assoc($result);
23    $id=$row['list_id'];
24
25    $Sql_Query = "select DISTINCT entries_PlannedTrack, cast(
26        entries_PlannedDepartureTime as time) AS 'PlannedDepartureTime',
27        entries_TrainNumber from dacs where entries_Destination1PtcarId=' $id'
28        and entries_PlannedDepartureTime is not null and entries_TrainNumber is
29        not null and entries_PlannedTrack < '11' and entries_ReferenceStatus='
30        Planned' ORDER BY 'dacs'. 'entries_PlannedDepartureTime' ASC
31    ";
32    $result = mysqli_query($con, $Sql_Query);
33    $json=[];
34    $nr=0;
35    while ($row = mysqli_fetch_assoc($result))
36    {
37        $json[$nr] = json_encode($row);
38        $nr=$nr+1;
```

```
36     }
37     $total=json_encode($json);
38     echo json_encode($total);
39
40
41     mysqli_close($con);
42 ?>
```

### A.2.3 get\_train\_info.php

```
1 <?php
2
3     // Importing DBConfig.php file.
4     include 'DBConfig.php';
5
6     // Creating connection.
7     $con = mysqli_connect($HostName,$HostUser,$HostPass,$DatabaseName);
8     // Getting the received JSON into $json variable.
9     $json = file_get_contents('php://input');
10
11    // decoding the received JSON and store into $obj variable.
12    $obj = json_decode($json,true);
13
14    $trainnr = $obj['Trainnr'];
15    $track = $obj['Track'];
16
17    // Creating SQL query and insert the record into MySQL database table.
18    $Sql_Query = "SELECT * FROM 'TracksSKW' where 'Track'='".$track."'";
19
20    $result = mysqli_query($con, $Sql_Query);
21    $json=[];
22    $nr=0;
23    while ($row = mysqli_fetch_assoc($result))
24    {
25
26        $json[$nr] = json_encode($row);
27        $nr=$nr+1;
28    }
29
30
31    $Sql_Query = "select DISTINCT commercialtraincompositions.
32                  lastPlanned_materialunits_positionInComposition,
33                  commercialtraincompositions.
34                  lastPlanned_materialUnits_tractionPosition,
35                  commercialtraincompositions.
36                  lastPlanned_materialUnits_materialSubTypeName,
```

```
commercialtraincompositions.lastPlanned_materialUnits_inDirection,
commercialtraincompositions.lastPlanned_materialUnits_hasPrmSection,
commercialtraincompositions.
lastPlanned_materialUnits_hasFirstClassOutlets,
commercialtraincompositions.
lastPlanned_materialUnits_hasSecondClassOutlets,
commercialtraincompositions.lastPlanned_materialUnits_hasTables,
commercialtraincompositions.lastPlanned_materialUnits_hasAirco,
commercialtraincompositions.lastPlanned_materialUnits_hasToilets,
commercialtraincompositions.lastPlanned_materialUnits_hasBikeSection,
commercialtraincompositions.lastPlanned_materialUnits_isFirstClass,
commercialtraincompositions.lastPlanned_materialUnits_isSecondClass,
IF(trainTypes.Length IS NOT null,trainTypes.Length,
commercialtraincompositions.lastPlanned_materialunits_lengthInMeter
*1000) AS 'Length',IF(trainTypes.Door1 IS NOT null,trainTypes.Door1
,8000) AS 'Door1' ,trainTypes.Door2 from commercialtraincompositions
LEFT JOIN trainTypes ON trainTypes.Subtype=
commercialtraincompositions.
lastPlanned_materialUnits_materialSubTypeName where
commercialtraincompositions.trainKey_trainNumber='$trainnr' and '
lastPlanned_materialUnits_positionInComposition' is not null
32 ORDER BY 'commercialtraincompositions.'.
lastPlanned_materialUnits_tractionPosition' ASC, '
commercialtraincompositions.'.
lastPlanned_materialUnits_positionInComposition' ASC";
33
34 $result = mysqli_query($con, $Sql_Query);
35 $json2=[];
36 $nr=0;
37 while ($row = mysqli_fetch_assoc($result))
38 {
39     $json2[$nr] = json_encode($row);
40     $nr=$nr+1;
41 }
42 $Sql_Query = "SELECT * FROM `objectTypes` ";
43 $result = mysqli_query($con, $Sql_Query);
44 $json3=[];
45 $nr=0;
46 while ($row = mysqli_fetch_assoc($result))
47 {
48     $json3[$nr] = json_encode($row);
49     $nr=$nr+1;
50 }
51 $Sql_Query = "SELECT * FROM `ObjectsSKW` where 'Track'='$track'";
52 $result = mysqli_query($con, $Sql_Query);
53 $json4=[];
54 $nr=0;
```

```
55  while ($row = mysqli_fetch_assoc($result))  
56  {  
57      $json4[$nr] = json_encode($row);  
58      $nr=$nr+1;  
59  }  
60  $total=json_encode([$json,$json2,$json3,$json4]);  
61  echo json_encode($total);  
62  // Echo the message.  
63  mysqli_close($con);  
64  ?>
```

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN  
CAMPUS DE NAYER SINT-KATELIJNE-WAVER  
J. De Nayerlaan 5  
2860 SINT-KATELIJNE-WAVER, België  
tel. + 32 15 31 69 44  
[iw.denayer@kuleuven.be](mailto:iw.denayer@kuleuven.be)  
[www.iw.kuleuven.be](http://www.iw.kuleuven.be)

