Parkspot 3.0

Nawang Tendar
Jan Temmerman
Kevin Leemans

Inhoud

Discover

Briefing

Initial Idea/Assumptions

- Parkeren in het centrum van een stad is een groot probleem. Mensen die met de auto rijden hebben gewoonweg niet genoeg accurate informatie over de vrije parkeerplaatsen in het centrum. Auto's rijden onnodig rond opzoek naar parkeerplaats. Dit veroorzaakt extra verkeer en CO2 uitstoot in de stad. De dag van vandaag zijn de oplossing op de markt inaccuraat of te duur voor te installeren en te gebruiken. Nieuwe "Computer Vision" technologieën en kleine hardware CPU's zijn toegenomen over de laatste jaren en deze zullen blijven groeien. Dit zorgt ervoor dat er nieuwe goedkopere oplossing op de markt komen zodat er geen financiële limieten zijn voor het zoeken van nieuwe parkeerinformatie.
- "Privacy by design". Foto's worden niet doorgestuurd buiten de parkeersensoren. (Dit is goed voor de privacy en de GDPR-richtlijnen)
- Automatische kalibratie
- Het verbeteren van de verkeersituatie in stadscentra door beter parkeeradvies aan de bestuurders.

Challenge

- Hoofdprobleem Parkspot 3.0: vertrouwbare herkenning van open parkeerplaatsen.
- Het vinden van de beste IT Solution voor parkeerplekken.

Solution Direction

- Volledige automatische herkenning
- Semi-automatisch in combinatie met menselijke interactie (community, outsourced india service, games, end users)
- Volledig op mensen afhankelijk (community, outsourced india service, games, end users)

Participants

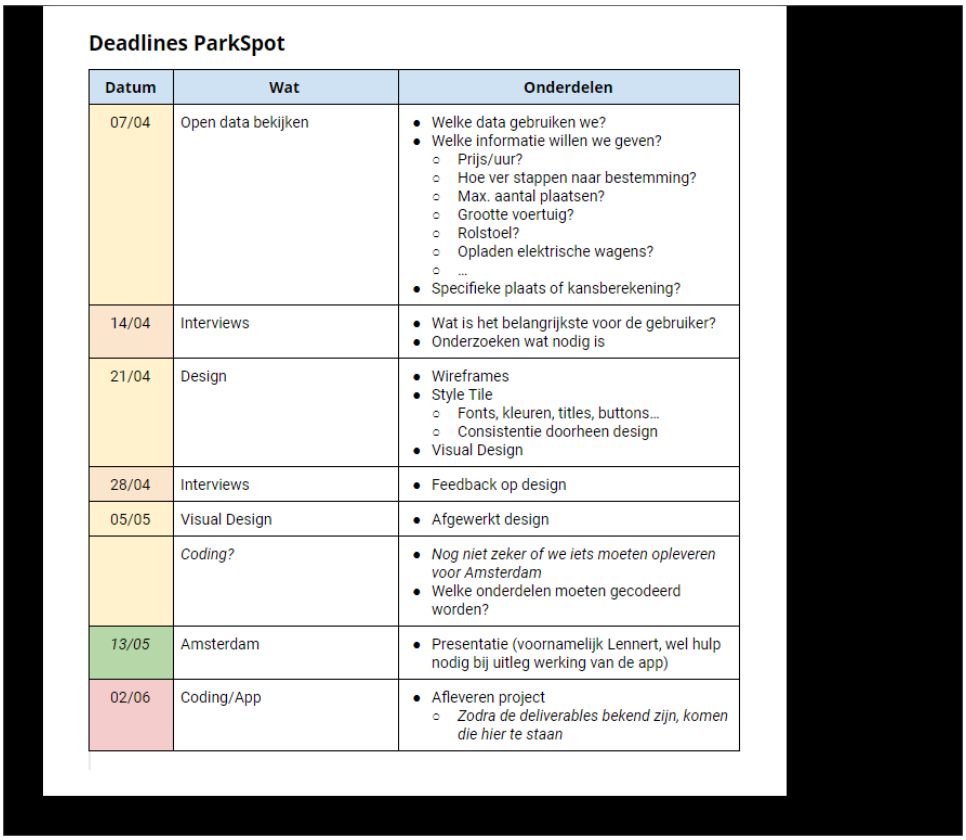
- Team Stuttgart: Making the Project Open Source
- Team Amsterdam: Hardware with back-end & computer vision integration via LoRa
- Team Ghent: UI & UX for the client in React Native with MERN-stack

Define

Planning

Agenda

- Start: Ma 01-04
- Milestone 1 (+feedback): Ma 29-04
- Milestone 2 (+feedback): Ma 13-05
- Presentation in Amsterdam: eind mei
- Deadline: Zo 16-06-2019 om 20u
- Examens: Ma 17-06-2019 of Di 18-06-2019



Deadlines ParkSpot

Datum	Wat	Onderdelen

07/04	Open data bekijken	• Welke data gebruiken we?
		• Welke informatie willen we geven?
		  ○ Prijs/uur?
		  ○ Hoe ver stappen naar bestemming?
		  ○ Max. aantal plaatsen?
		  ○ Grootte voertuig?
		  ○ Rolstoel?
		  ○ Opladen elektrische wagens?
		  ○ ...
		• Specifieke plaats of kansberekening?

14/04	Interviews	• Wat is het belangrijkste voor de gebruiker? • Onderzoeken wat nodig is

21/04	Design	• Wireframes
		• Style Tile
		  ○ Fonts, kleuren, titles, buttons...
		  ○ Consistentie doorheen design
		• Visual Design

28/04	Interviews	• Feedback op design
05/05	Visual Design	• Afgewerkt design
	Coding?	• *Nog niet zeker of we iets moeten opleveren voor Amsterdam* • Welke onderdelen moeten gecodeerd worden?
13/05	Amsterdam	• Presentatie (voornamelijk Lennert, wel hulp nodig bij uitleg werking van de app)
02/06	Coding/App	• Afleveren project ○ *Zodra de deliverables bekend zijn, komen die hier te staan*

Specificaties

Technische Specificaties

Server

De server wordt geschreven met JavaScript (esnext) in Node.js met het Express.js framework. Voor authenticatie en autorisatie gebruiken we het Passport framework. Binnen Passport zullen we minimaal een local-strategy, jwt-strategy en een extra strategy (bv: facebook-strategy, google-strategy, twitchtv-strategy, ...) aanbieden.

De server is voorzien van een API (version controlled). Voor ieder collectie uit de database maken we een corresponderende controller, schema, validatie en routes. De volgende actiemethoden zijn minimaal in de controller aanwezig: index (alle documenten uit een collectie tonen, inclusief pagination), show (GET), create, store (POST), edit, update (PUT), delete (DELETE), softdelete (PATCH) en softundelete (PATCH).

Alle API-endpoints worden gedocumenteerd via de OpenAPI-specificatie m.b.v swagger-jsdoc en swagger-ui-express (of ReDoc). Beveilig de swagger-ui-express en dus ook de API-endpoints met JWT. De API is afgeschermd voor niet-bevoegden en voldoet aan de normale eisen voor beveliging.

Zorg voor een globale foutafhandeling, logging en custom 404 en Error pages (m.b.v. ejs template engine).

De client wordt geïnjecteerd in de root route: http://{your domain}:{your port}/. De client wordt niet gerenderd op de server, maar wel de informatie in de <head>. Op deze manier wordt de app SEO-friendly en ook om te bookmarken via OpenGraph e.d.. SSR is ook mogelijk, maar combineren met PWA is niet eenvoudig.

Database

We gebruiken MongoDb als databasesysteem. Deze databank moet gehost worden op mLab of MongoDb. Gebruik op de campus of binnen het netwerk van de Arteveldehogeschool de wifi "ArteveldeHS Open".

Voor een team dat bestaat uit:

- 1 lid, moeten minimaal 7 collecties (tabellen) aanwezig zijn, exclusief users en roles.
- 2 leden, moeten minimaal 9 collecties (tabellen) aanwezig zijn, exclusief users en roles..
- 3 leden, moeten minimaal 11 collecties (tabellen) aanwezig zijn, exclusief users en roles.

De volgende relaties tussen documenten (en dit zonder embedded documents) moeten minstens eenmaal voorkomen:

- 1..1 (één op één)
- 1..N (één op veel)
- N..N (veel op veel)

De volgende Unit Tests (Mocha, Chai, Sinon) moeten minimaal aanwezig zijn voor een team dat bestaat uit:

- 1 lid: tests voor een minimaal 1 controller, tests voor de auth controller, tests voor minimaal 1 mongoose model
- 2 leden: tests voor een minimaal 2 controllers, tests voor de auth controller, tests voor minimaal 2 mongoose models
- 3 leden: tests voor een minimaal 3 controllers, tests voor de auth controller, tests voor minimaal 3 mongoose models

Client

De cliënt die geïnjecteerd wordt in de server is een PWA React-client met gedeeltelijke SSR (volledig kan ook). De cliënt bevat ook state management via Redux met Redux middleware (Thunk, Sagas of Epics). Het router-systeem dat we zullen gebruiken binnen React is react-router.

Styling wordt binnen React toegevoegd via scss en / of via styled components.

De volgende Unit Tests (Jest, fetch-mock, expect, Sinon) moeten minimaal aanwezig zijn voor een team dat bestaat uit:

- 1 lid: tests voor een minimaal 1 component, tests voor 1 redux store en middelware
- 2 leden: tests voor een minimaal 2 componenten, tests voor 2 redux stores en middelware
- 3 leden: tests voor een minimaal 3 componenten, tests voor 3 redux stores en middelware

Functionele Specificaties

- React Native front-end met connectie naar de API van de backoffice. Deze moet:
    - users kunnen aanmelden, registreren.
    - Open data API's aanspreken van de steden
- Routing tussen de verschillende screens.
- Je favoriete navigatie app gebruiken als routing voor uw bestemming. Onze app is een tussen station voor open parkeerplaatsen te vinden.
- User moet zijn voorkeuren kunnen aanduiden en aan de hand van deze voorkeuren te zien krijgen waar er nog open plaatsen zijn voor te parkeren.
- User moet favorieten hebben volgens CRUD
- User moet kunnen kiezen tussen de plaatsen die het dichts bij zijn eindbestemming zijn volgens de opgegeven voorkeuren.

Scenario's

1. Ik wil naar Gent maar ik wil op de Park & Ride staan. Naar Gent wil ik gaan met behulp van mijn persoonlijke geprefereerde navigatie. Ik wil de dichtste bijliggende Park & Ride van mijn eindpositie. De Park & Ride mag niet vol zijn. Vanuit de Park & Ride wil ik genavigeerd worden naar mijn eindbestemming.
2. Ik wil naar Gent maar ik wil liefst zo dicht mogelijk bij mijn eindpositie staan zodat ik niet te veel moet stappen. Ik wil hiervoor mijn geprefereerde navigatiesysteem gebruiken.
3. Ik wil naar Gent maar ik wil ergens staan waar het goedkoop is. Dan wil ik van daar met het openbaar vervoer naar mijn eindbestemming. Ik wil hiervoor mijn geprefereerde navigatiesysteem gebruiken. Ik wil gemakkelijk kunnen betalen. Ik wil van mijn parkeerplaats de juiste dichts bij zijnde tram of bus nemen.
4. Ik wil naar Gent en ik sta het liefst op de ondergrondse parking. Dan wil ik genavigeerd worden te voet naar mijn eindebestemming. Ik wil hiervoor mijn geprefereerde navigatiesysteem gebruiken.

Design

Sitemap

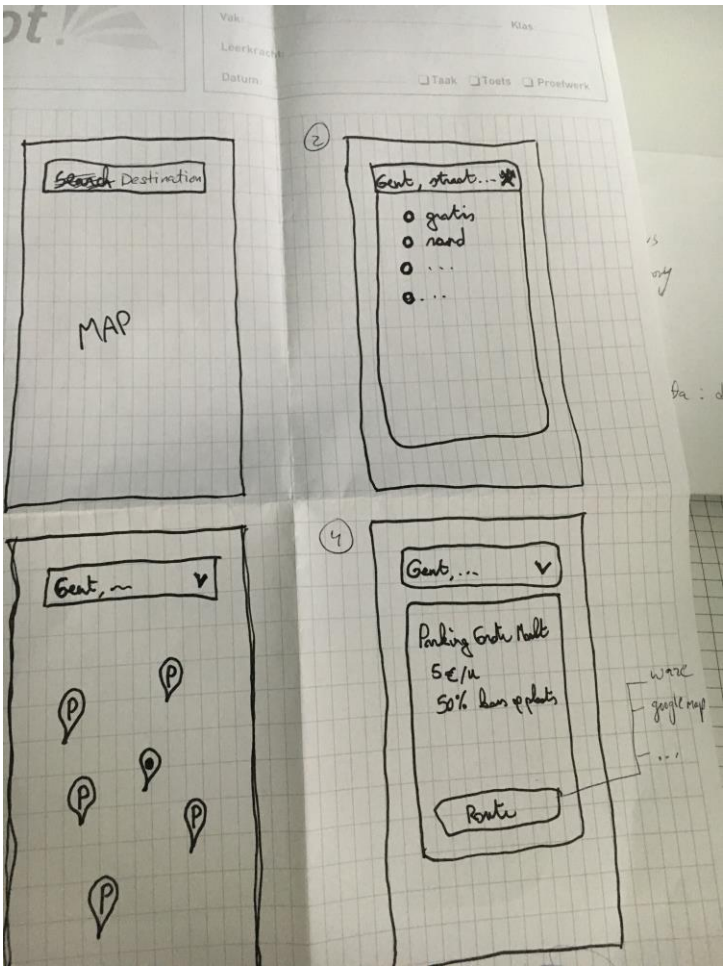
React Native

Back-Office

https://www.gloomaps.com/QvWTkyjwtT

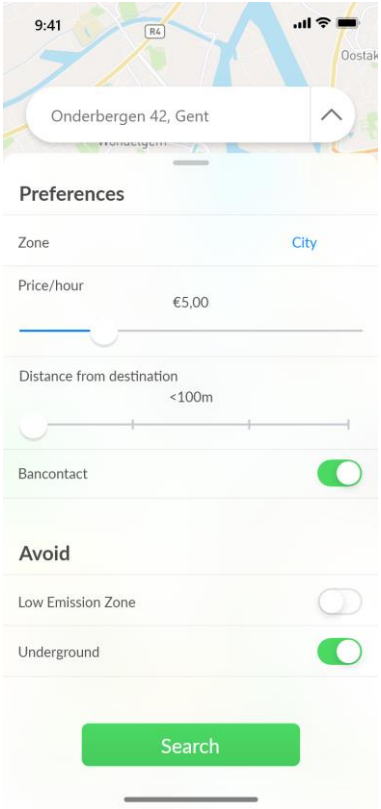
Wireframe

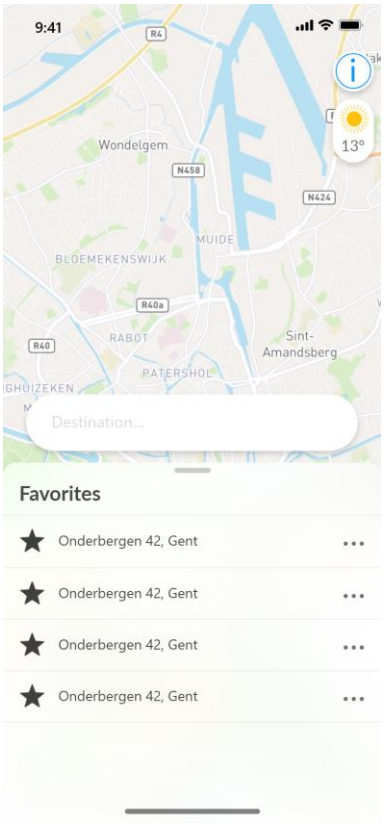
React Native App

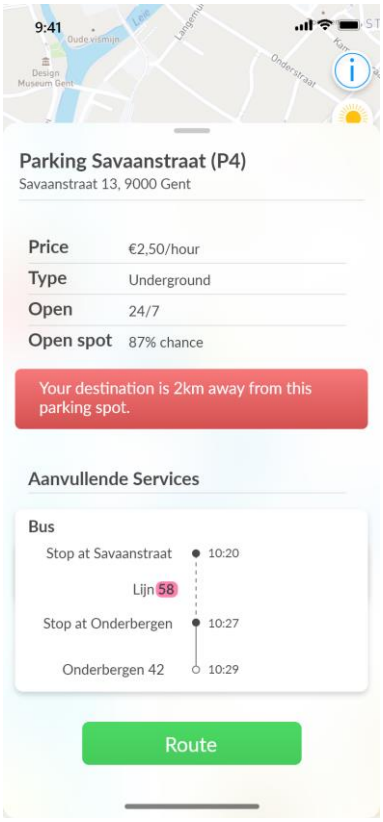


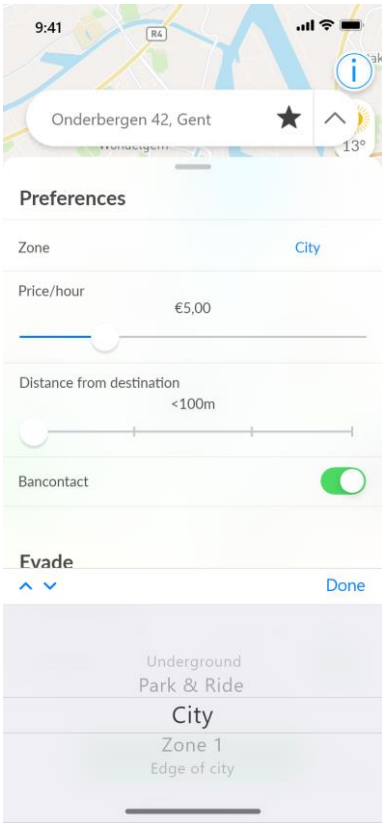
Visual Designs

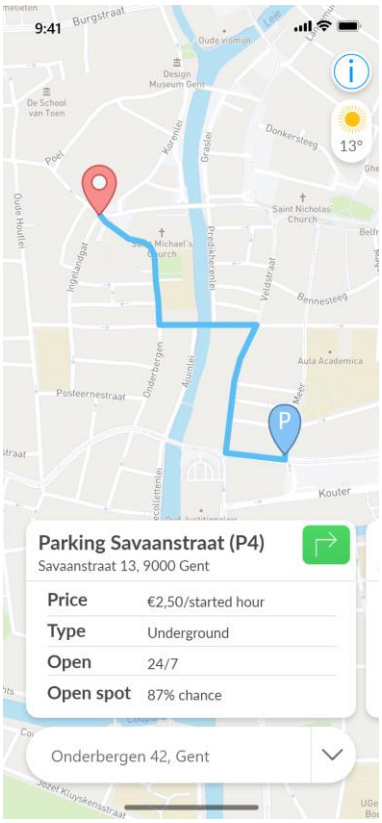
React Native App

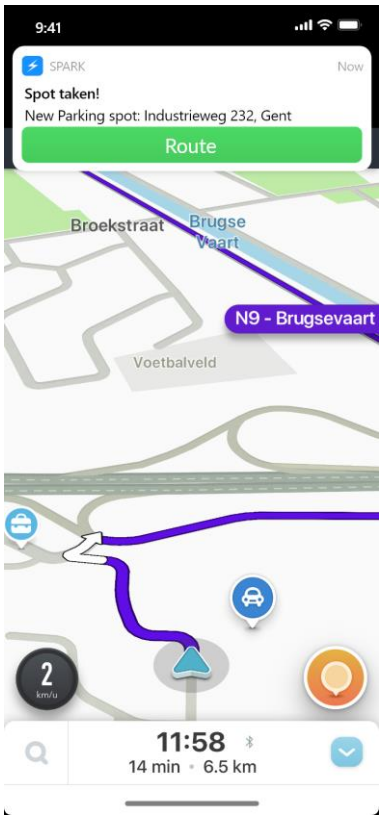



9:41
Wondelgem
MUIDE
BLOEMEKENSWIJK
RABOT
Sint-
Amandsberg
PATERSHOL
Destination...
Favorites
Onderbergen 42, Gent
Onderbergen 42, Gent
Onderbergen 42, Gent
Onderbergen 42, Gent

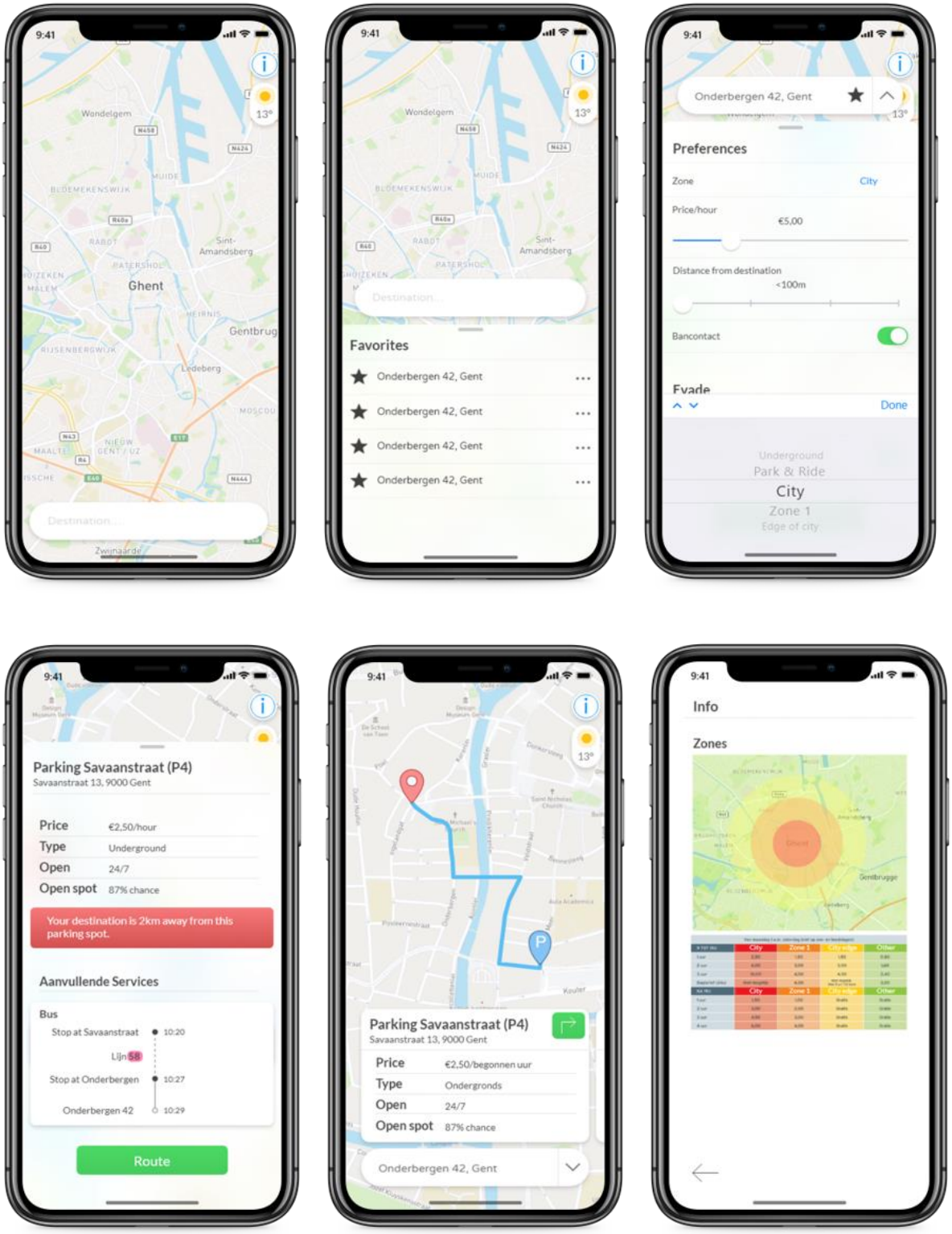

9:41
Parking Savaanstraat (P4)
Savaanstraat 13, 9000 Gent
Price €2,50/hour
Type Underground
Open 24/7
Open spot 87% chance
Your destination is 2km away from this parking spot.
Aanvullende Services
Bus
Stop at Savaanstraat 10:20
Lijn 58
Stop at Onderbergen 10:27
Onderbergen 42 10:29
Route


9:41
Onderbergen 42, Gent
Preferences
Zone City
Price/hour
€5,00
Distance from destination
<100m
Bancontact
Evade
Done
Underground
Park & Ride
City
Zone 1
Edge of city


9:41
Parking Savaanstraat (P4)
Savaanstraat 13, 9000 Gent
Price €2,50/started hour
Type Underground
Open 24/7
Open spot 87% chance
Onderbergen 42, Gent


9:41
SPARK Now
Spot taken!
New Parking spot: Industrieweg 232, Gent
Route
Broekstraat
Brugse Vaart
N9 - Brugsevaart
Voetbalveld
11:58
14 min · 6.5 km

Mock-Ups


9:41
Wondelgem
Ghent
Sint-Amandsberg
Gentbrugge
Ledeberg
Destination...
Favorites
Onderbergen 42, Gent
Onderbergen 42, Gent
Onderbergen 42, Gent
Onderbergen 42, Gent
Preferences
Zone
City
Price/hour
€5,00
Distance from destination
<100m
Bancontact
Done
Underground
Park & Ride
City
Zone 1
Edge of city
Parking Savaanstraat (P4)
Savaanstraat 13, 9000 Gent
Price
€2,50/hour
Type
Underground
Open
24/7
Open spot
87% chance
Your destination is 2km away from this parking spot.
Aanvullende Services
Bus
Stop at Savaanstraat
10:20
Lijn 58
Stop at Onderbergen
10:27
Onderbergen 42
10:29
Route
Parking Savaanstraat (P4)
Savaanstraat 13, 9000 Gent
Price
€2,50/begonnen uur
Type
Ondergronds
Open
24/7
Open spot
87% chance
Onderbergen 42, Gent
Info
Zones

Develop

Content

Logo

Functionaliteit

Code Snippets

React Native App

AsyncStorage

```
/**
 * @function _postDataToAsyncStorage
 * @param {String} key
 * @param {String} value
 * Posts a key with a certain value to the AsyuncStorage
 */
async _postDataToAsyncStorage (key, value) {
    try {
        await AsyncStorage.setItem(key, value);
    } catch (error) {
        console.error(error)
    }
}

/**
 * @function _retrieveDataFromAsyncStorage
 * @param {String} key
 * Will check if there is a key in the AsyncStorage
 */
async _retrieveDataFromAsyncStorage(key) {
    try {
      const value = await AsyncStorage.getItem(key);
      if (value !== null) {
        // We have data!!
        console.log(value);
      } else {
          console.log('no value')
      }
    } catch (error) {
      // Error retrieving data
      console.error(error)
    }
  }
```

Fetch request from self-made API

```
/**
 * @function postUserToAuthentication
 * Asynchronous function to post the user to the authentication in the database
 * If the post request was succesfull store the value in the AsyncStorage with the response token from the API
 */
async postUserToAuthentication(){
    const url = "http://192.168.1.5:8080/api/v1/login/local"
    var data = {
        email: this.state.email,
        password: this.state.password
    }

    await fetch(url, {
      method: 'POST', // or 'PUT'
      body: JSON.stringify(data), // data can be `string` or {object}!
      headers:{
        'Content-Type': 'application/json'
      }
    })
    .then(res => res.json())
    .then(response => {
        console.log('Success:', JSON.stringify(response))
        console.log('Token:', response.token)
        this._postDataToAsyncStorage('userToken', response.token)
    })
    .catch(error => {
        console.log('Error:', error)
        this.setState({errorPassVisible: true})

    })
}
```

Routing

```
/**
 * @function render
 * @returns The routing of the app used in the root component       You, 42 minutes ago • adde
 * If a user is not logged in, show different routing.
 */
render() {
    //Check if a user is logged in or not
    if (this.state.loggedIn){
        return(
            <Router>
                <Scene key="root" hideNavBar>
                    <Scene key="home" component={HomeScreen} title="Home" initial={true} />
                </Scene>
            </Router>
            )
    } else {
        return(
            <Router>
                <Scene key="root" hideNavBar>
                    <Scene key="login" component={LogInScreen} title="Login" initial={true}/>
                    <Scene key="register" component={RegisterScreen} title="Register"/>
                    <Scene key="home" component={HomeScreen} title="Home" />
                </Scene>
            </Router>
        )
    }
}
```

Validation

```
/**
 * @function validate
 * @param {String} input
 * @param {String} type
 * Validates the input by a certain type
 * @returns {boolean}      You, an hour ago • added comments
 */
export default function validate(input, type) {
    switch(type) {
        case "email":
            const regexEmail = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".
            return regexEmail.test(input)
        case "password":
            const regexPass = new RegExp("^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.{8,})");
            return regexPass.test(input)
    }
}
```

API call to get free parkingspots

```
    getResultsFromAPI = async(zone, price, distance, bancontact, lez, underground) => {
        const url = "http://192.168.5.136:8080/api/v1/searchparkingspots"

        var data = {
            "destinationGeo": {
                "long": this.state.destination.lng,
                "lat": this.state.destination.lat
                },
            "userId": this.state.userId,
            "settings": {
                "zonename": zone,
                "price_per_hour" : price,
                "distance_from_destination" : distance,
                "bankcontact" : bancontact,
                "low_emission_zone" : lez,
                "underground" : underground
                }
        }

        await fetch(url, {
          method: 'POST', // or 'PUT'
          body: JSON.stringify(data), // data can be `string` or {object}!
          headers:{
            'Content-Type': 'application/json'
          }
        })
        .then(res => res.json())
        .then(async(response) => await this.setState({
            parkings: response
        }))
        .catch(error => console.error('Error:', error))
    }
```

```
render() {
    return (
        <View style={styles.container}>
            <LogOutButton />
            <Map  ref={this.mapElement}/>
            <WeatherIcon/>
            <KeyboardAvoidingView style={styles.cardContainer} behavior="padding" enabled keyboardVerticalOffset={70}>
                <ScrollView ref={(scrollView) => { this.scrollView = scrollView; }} horizontal={true} decelerationRate={0} snapToInterval={width - 20}
                contentInset={{
                    top: 0,
                    left: 10,
                    bottom: 0,
                    right: 10,
                }} style={{display: this.state.cardDisplay}}>

                    {Array.from(this.state.parkings).map(parking => (
                        <Card
                        key={parking.address}
                        onClick={this.showMarkers}
                        destination={this.state.destinationAddress}
                        parkingName={parking.name}
                        address={`${parking.address.match(/(^.*?[^;][0-9])/g)},${(parking.address.split(",")[1])}`}
                        price={parking.price.day}
                        type={parking.type}
                        openWhen={parking.open}
                        chance={parking.chance}
                        />
                    ))
```

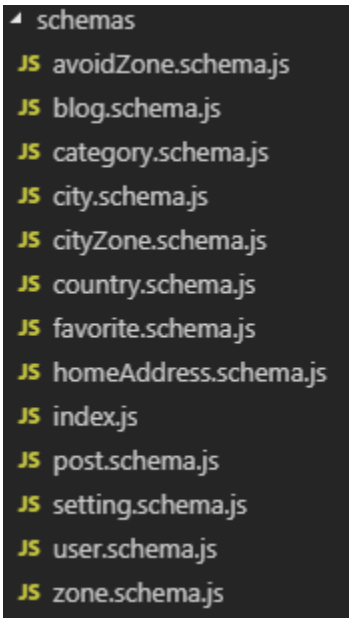
Back-office

Routing / RESTfull API

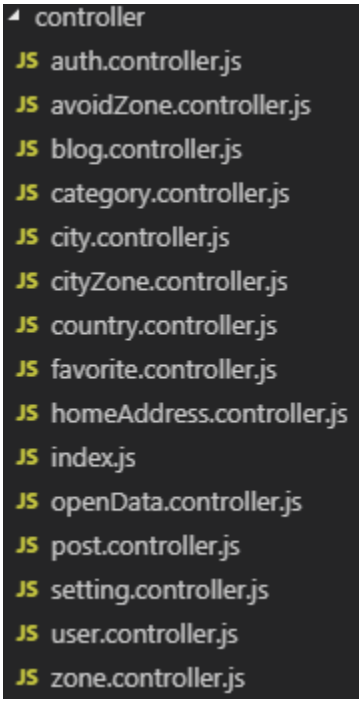
```
// Define and initiate an express router
const apiV1Router = express.Router();
authRouter(apiV1Router, authService);
blogRouter(apiV1Router, authService);
categoryRouter(apiV1Router, authService);
postRouter(apiV1Router, authService);
userRouter(apiV1Router, authService);
countryRouter(apiV1Router, authService);
cityRouter(apiV1Router, authService);
zoneRouter(apiV1Router, authService);
favoriteRouter(apiV1Router, authService);
homeAddressRouter(apiV1Router, authService);
avoidZoneRouter(apiV1Router, authService);
settingRouter(apiV1Router, authService);

openDataRouter(apiV1Router, authService);
cityZoneRouter(apiV1Router, authService);
export default apiV1Router;
```

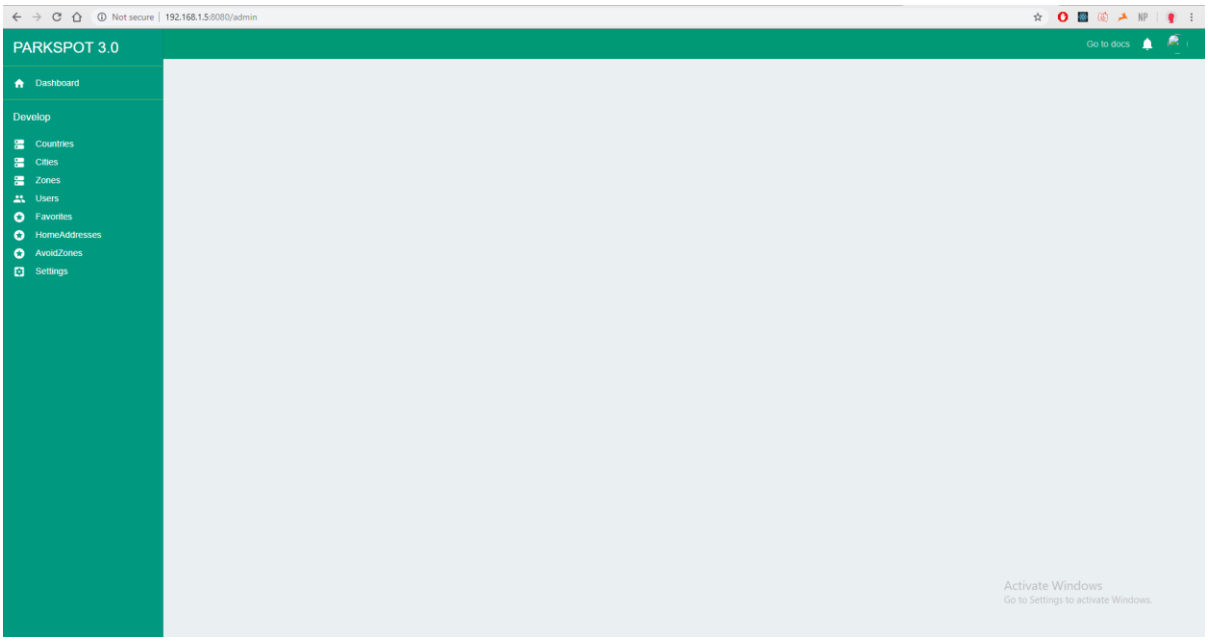
Models

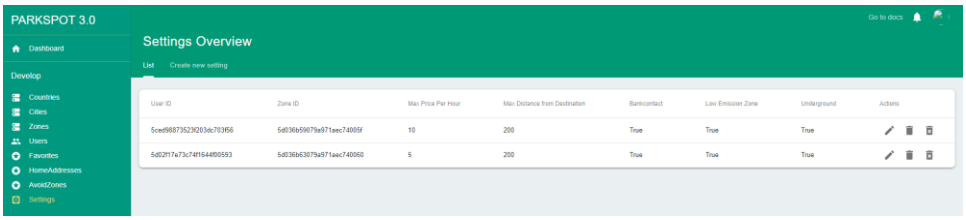


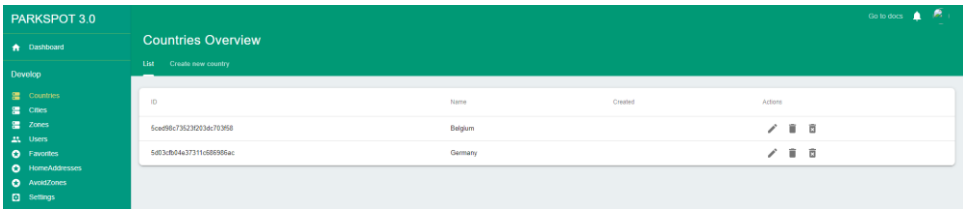
Controllers

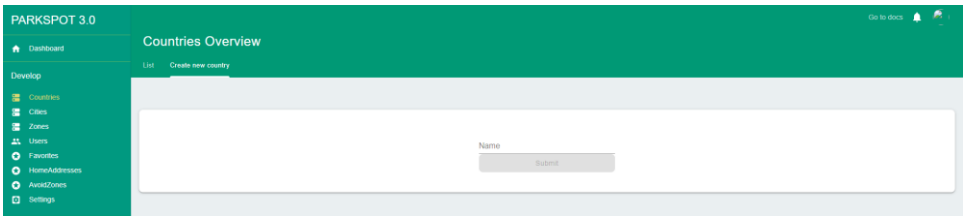


Back-office interface









Deliver

Handleiding

Logged in users

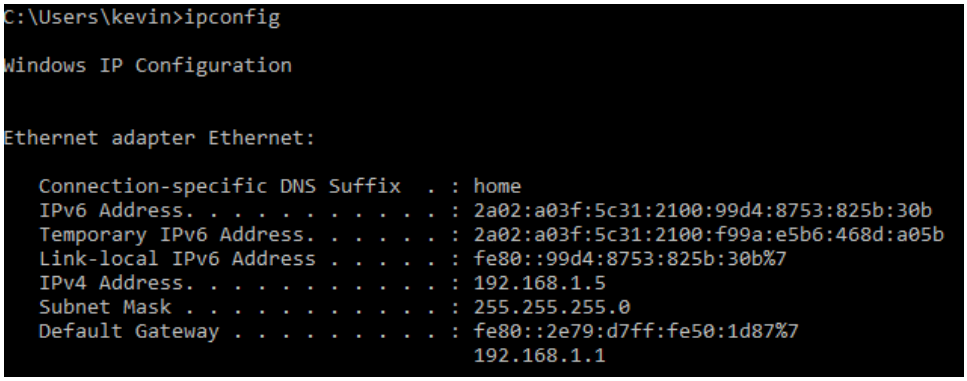
- Kan aanmelden, uitloggen, account verwijderen
- Heeft toegang tot de front-end van de react native app
- Kan favorieten toevoegen en verwijderen, updaten
- Kan voorkeuren meegeven bij zijn eindbestemming
- Kan kiezen tussen verschillende eindbestemmingen
- Kan zijn favoriete gebruikte navigatie-app gebruiken voor de navigatie

Admin

- Toegang tot alles wat een geregistreerde gebruiker kan
- Kan aanmelden in de back-office
- Kan alle data in de database aanpassen

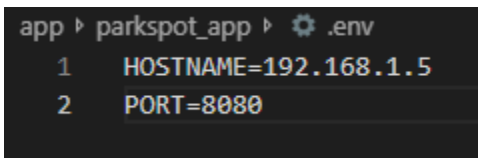
Deploy

1. Maak een folder op uw pc
2. Download Parkspot App https://github.com/parkspot/parkspot-3
3. Ga in de root van de app naar de folder app
4. Ga in de folder app naar de parkspot folder
5. Voer npm install uit in je terminal of command line
6. In deze folder moet je ook een .env file aanmaken met daarin de HOSTNAME en PORT van je ip:
    A. Windows:
        1. Open cmd uit start
        2. Voer hier ipconfig uit
        3. Het IPv4 address is je HOSTNAME
        4. PORT is standaard 8080



    B. MacOS:
        1. In Apple menu ga je naar Systeem voorkeuren
        2. Klik op Netwerk voorkeuren
        3. Uw IP address is zichtbaar aan de rechterzijde zoals op de screenshot zichtbaar is

7. Je .env file ziet er zo uit:



8. Daarna voer je npm start uit in de terminal of command line
9. Maak een nieuwe folder aan die losstaat van de parkspot app
10. Download de back-end voor het project op https://github.com/nawatend/parkspot-backend
11. Ga in de root van de repository naar de parkspot_backoffice folder
12. Hier voeg je in de root van deze folder een .env file toe en die ziet er als volgt uit:

parkspot_backoffice ▸ .env

```
NMD_BASELINE='Like Graphics Love Code'
NODE_ENV= Development
NODE_SERVER_HOSTNAME=192.168.1.5
NODE_SERVER_PORT=8080
MONGODB_CONNECTION="mongodb+srv://nawang:0Happyman@parkspot-flhvd.gcp.mongodb.net/parkspot?retryWrites=true"

SKIP_PREFLIGHT_CHECK=true
AUTH_BCRYPT_SALT=10
AUTH_JWT_SECRET='parkspot'
AUTH_JWT_SESSION=true
AUTH_FACEBOOK_CLIENT_ID={your Facebook Client id}
AUTH_FACEBOOK_CLIENT_SECRET={your Facebook Client secret}


AUTH_GOOGLE_CLIENT_ID="349015460192-0qf7qh04rv0pfvb5v0mhf6f7iblivnui.apps.googleusercontent.com"
AUTH_GOOGLE_CLIENT_SECRET="IB7MZHvnz7tHRlMDowjxDWS0"
```

13. Je past NODE_SERVER_HOSTNAME aan, aan de hostname die je gebruikt hebt in de react native app
14. Voer nu yarn install uit in de terminal
15. Daarna start je de server op door server:start in de terminal in te voeren