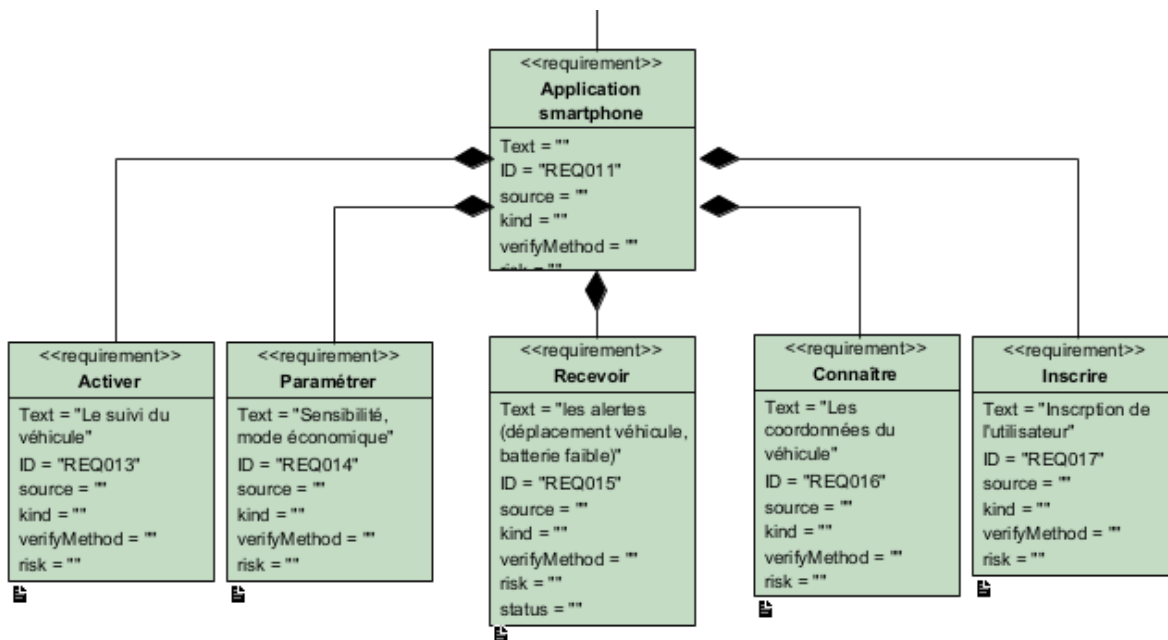


# **Création de l'application du smartphone de l'utilisateur**

*Sylvain BRUNET*

Mon objectif pour ce projet a été de réaliser une application capable de consulter la position et l'historique de déplacement de son véhicule. Elle devait aussi permettre le paramétrage du traqueur ainsi que d'avertir l'utilisateur lors d'un déplacement de celui-ci. Pour effectuer ces actions, l'application devait donner la possibilité à l'utilisateur de s'inscrire, de se connecter et d'ajouter un traqueur à son compte.



## 1 – Outils de développement

### 1.1 – React native :



Pour développer l'application, j'ai choisi d'utiliser le React Native, un framework d'applications mobiles open source créé par Meta pour développer des applications IOS et Android avec un seul et même code source. Je n'avais jamais utilisé ce langage et est donc dû passer par un phase d'apprentissage avant de me lancer.

```
import React from 'react';
import { AppRegistry, Text } from 'react-native';

const HelloWorldApp = () => {
  return (
    <Text>Hello world!</Text>
  );
}
export default HelloWorldApp;
```

La syntaxe est très proche de la librairie «React» de JavaScript notamment avec le système de balise pour l’affichage et la base JavaScript.

## 1.2 – Expo :



Expo est une plateforme open source permettant de créer des applications natives universelles pour Android, iOS et Web. Il permet tout d’abord de coder sur leur éditeur en ligne «Expo Snack» qui permet de regrouper et compiler le code dynamiquement et de l’exécuter dans un lecteur web ou un émulateur mobile. Il est aussi possible de tester et d’utiliser l’application sur Android et IOS très rapidement à travers l’application «ExpoGO».

## 2 – Faisabilités

### 2.1 – Navigation entre les pages :

Pour pouvoir naviguer entre les différentes pages de l’application, j’ai utilisé la librairie open-source «React navigation» développée par Expo. Pour l’utiliser, il suffit de mettre dans sa fonction principale les balises «NavigationContainer» et «Stack.Navigator» avec la page initiale, puis d’ajouter à l’intérieur une balise «Stack.Screen» par page en lui spécifiant un nom et la fonction/classe qui l’affichera.

```
function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Ensuite, pour naviguer entre les pages, il faudra la fonction «navigate» de l'objet «navigation» en spécifiant le nom de la page où on veut aller.

```
function HomeScreen({ navigation }) {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
}
```

## 2.2 – Notifications :

Pour permettre la réception des alertes, j'ai encore une fois utilisé une librairie open-source d'Expo qui nomme «Expo Notification». La fonction suivante permet de demander à l'utilisateur les permissions pour qu'il reçoive des notifications. Si l'utilisateur accepte, la fonction récupère son token qui sera envoyé puis utilisé par l'API pour envoyer des notifications. Cette fonction est asynchrone pour pouvoir faire des affichages en même temps.

```
/* Demande à l'utilisateur les permissions pour recevoir des notifications et récupère son token */
async function registerForPushNotificationsAsync() {
  let token;

  /* Si l'utilisateur est sur un vrai téléphone */
  if (Device.isDevice) {
    /* Demande les permissions */
    const { status: existingStatus } =
      await Notifications.getPermissionsAsync();
    let finalStatus = existingStatus;
    /* Si les permissions ne sont pas déjà accordées, les demandes */
    if (existingStatus !== 'granted') {
      const { status } = await Notifications.requestPermissionsAsync();
      finalStatus = status;
    }
    /* Si les permissions n'ont pas été accordées, affiche une erreur */
    if (finalStatus !== 'granted') {
      console.log('Permissions for notifications have been denied');
      return;
    }
    /* Si les permissions ont été acceptées, récupère le token de l'utilisateur. */
    else {
      token = (await Notifications.getExpoPushTokenAsync()).data;
      console.log(token);
    }
    /* Si l'utilisateur n'est pas sur un vrai téléphone (web, émulateur...), affiche une erreur */
  } else {
    alert('Must use physical device for Push Notifications');
  }

  return token;
}
```

## 3 – Réalisation et gestion des utilisateurs

### 3.1 – Inscription :

Après avoir démarré l'application, l'utilisateur peut se rendre dans la section «Register» pour créer un compte.

08:47 95%

← LoginNavigator

Email

Password

Register Login

08:47 95%

← RegisterNavigator

Email

Test@gmail.com

Password

...

Confirm Password

...

Register

L'application commence tout d'abord par stocker les informations saisies par l'utilisateur.

```
{/* Créé un champ qui met dans la variable 'email' ce que saisi l'utilisateur */}  
<Text style={Styles.paragraph}>E-mail</Text>  
<TextInput  
  style={Styles.input}  
  onChangeText={(text) => setEmail(text)}  
  value={email}  
</>
```

Ensuite, lorsque l'utilisateur a appuyé sur le bouton «Register», l'application vérifie si l'e-mail et le mot de passe sont corrects (bonne longueur, pas d'espace...).

```
/* Si l'utilisateur n'a rien saisi dans un des champs, affiche une erreur */
if (email == '' || password == '' || confirmPassword == '') {
  alert('Please enter information in the fields');
  /* Si l'utilisateur n'a pas saisi le même mot de passe */
} else if (password !== confirmPassword) {
  alert("Passwords fields haven't the same information");
  /* Si l'utilisateur ne rentre pas un email */
} else if (/@/g.test(email) == 0 || /g.test(email) == 1 || email.length > 255) {
  alert('Please enter correct email in the field');
  /* Si l'utilisateur met un espace dans le mot de passe */
} else if (/ /g.test(password) == 1) {
  alert("Please don't put a space in the password field");
  /* Si l'utilisateur saisi un mot de passe trop long ou trop court */
} else if (password.length < 8 || password.length > 24) {
  alert('Please enter a password between 8 and 24 characters');
```

Si les informations sont correctes, une requête «POST» à l'API en spécifiant l'e-mail et le mot de passe du nouvel utilisateur, ainsi que son token de notification qui permettra à l'API de lui envoyer des alertes.

POST	/user/{mail}/{password}/{notif}
Used to	Add a user to the database with the specified mail and password.
Json Output	Topics: {"topicTX": string, "topicRX": string}, TrackerId: int, "error": {"Code": int, "Message": string}}

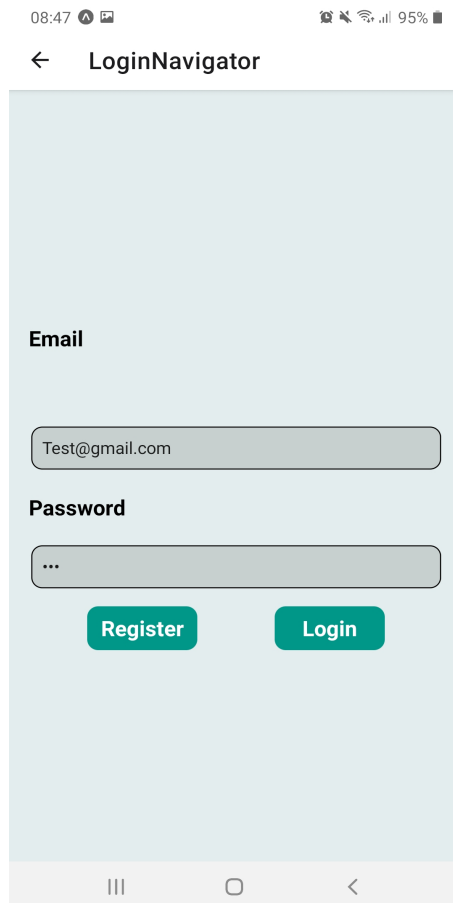
```
/* Sinon, fait une requête 'POST' à l'API en envoyant l'email, le mot de passe et le token expo de l'utilisateur */
alert(expoPushToken);
fetch('https://api.ovl.tech-user.fr/user/', {
  method: 'POST',
  headers: {
    Accept: 'application/json',
    'Content-Type': 'application/json',
  },
  /* Données envoyées à l'API */
  body: JSON.stringify({
    mail: email,
    password: password,
    notif: expoPushToken,
  }),
})
```

L'API répond ensuite (si il n'y a pas eu d'erreur) avec le topicRX et le topicTX pour le traqueur ainsi que l'ID de celui-ci qui sont affichés à l'écran. L'utilisateur est aussi renvoyé à la page de connexion.

```
/* Récupère la réponse de l'API */
.then((response) => response.json())
.then((json) => {
  console.log(json);
  /* Si il n'y a pas eu d'erreur, envoie l'utilisateur à la page de login et affiche un succès d'inscription */
  if (json.error.Message == 'Nothing goes wrong.') {
    alert(
      'Registration has been successfully completed.\n' +
      'topicTX : ' +
      json.Topics.TX +
      '\n' +
      'topicRX : ' +
      json.Topics.RX +
      '\n' +
      'trackerID : ' +
      json.TrackerId
    );
    navigation.goBack();
  }
});
```

### 3.2 – Connexion :

Si l'utilisateur a déjà un compte, il peut alors se connecter via l'interface suivante en saisissant son mot de passe et son e-mail :



The screenshot shows a mobile application interface for a login screen. At the top, the status bar displays the time 08:47, signal strength, Wi-Fi, and 95% battery. Below the status bar is a navigation bar with a back arrow and the title 'LoginNavigator'. The main content area has a light blue background. It contains two labels: 'Email' and 'Password'. Under 'Email' is a text input field containing 'Test@gmail.com'. Under 'Password' is a text input field with a masked password '...'. At the bottom of the form are two green buttons: 'Register' and 'Login'. The bottom of the screen shows the standard Android navigation bar with three icons: a square, a circle, and a triangle.

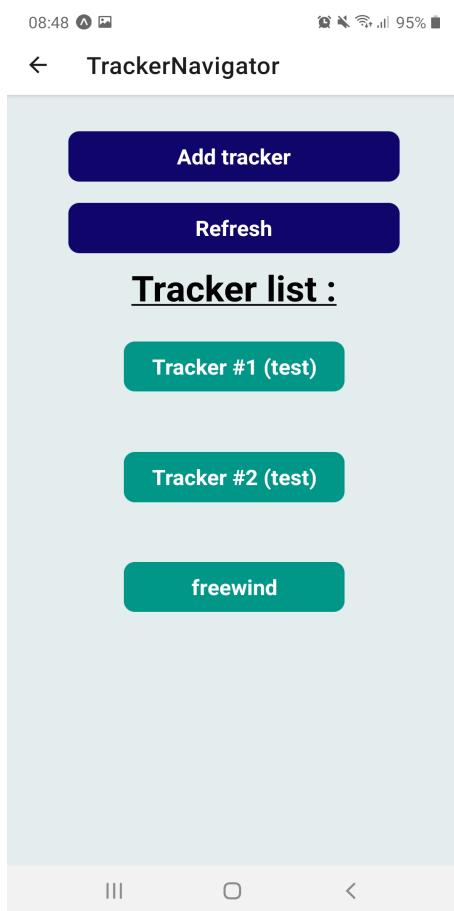
L'application stocke ces informations de la même manière que pour l'inscription. Lorsque le bouton «Login» est utilisé, une requête GET est envoyée à l'API en spécifiant l'e-mail et le mot de passe de l'utilisateur. Si l'utilisateur existe bien, l'API renvoie le token de l'utilisateur et l'application passe à la page de sélection du traqueur.

GET	/user/{mail}/{password}
Used to	Get an api token from a mail and password.
Json Output	<pre>{"token": string, "error": {"Code": int, "Message": string}}</pre>
<pre>/* Fonction qui va faire une requête à l'API en envoyant l'email et Le mot de passe saisi par l'utilisateur afin de recevoir son token */ const getToken = async () =&gt; {   /* Si il y a au moins 1 champ vide, affiche une erreur */   if (email == ''    password == '') {     alert('Please enter information in the fields');     /* Si Le champ 'e-mail ne contient pas d'arobase, affiche une erreur */   } else if (/@/g.test(email) == 0) {     alert('Please enter correct email in the field');   } else {     try {       console.log('Email : ' + email);       console.log('Password : ' + password);       /* Fait une requête à l'API en lui donnant l'email et le mot de passe saisi par l'utilisateur */       const response = await fetch(         'https://api.ovl.tech-user.fr/user/' + email + '/' + password       );       /* Met La réponse se l'API dans la variable 'json' */       const json = await response.json();       console.log(json);       /* Si il n'y a pas d'erreur, change la fenêtre de navigation pour celle de selection du tracker       en envoyant le token et la variable 'navigation' */       if (json.error.Message == 'Nothing goes wrong.') {         console.log('LoginScreen token : ' + json.user);         navigation.navigate('TrackerNavigator', {           token: json.user,           navigation: navigation,         });         /* Sinon, affiche un message d'erreur indiquant que les identifiants sont incorrects. */       } else {         alert("Nom d'utilisateur ou mot de passe incorrect.");       }     } catch (error) {       /* Si l'API ne répond pas, affiche une erreur indiquant que l'API ne fonctionne pas */       console.log(error);       alert('API down');     }   } };</pre>	



### 3.3 – Sélection du traqueur :

Une fois l'utilisateur connecté, une page avec la liste des traqueurs comme ci-dessous de l'utilisateur s'affiche.



Au lancement de la classe qui affiche cette page, la fonction suivante est exécutée et permet grâce à une requête GET envoyée à l'API de recevoir les IDs et les noms de chaque traqueur de l'utilisateur en spécifiant son token.

GET	/iot_list/{token}
Used to	Get the tracker list for the user from his token.
Json Output	<code>{"trackers": [{"name": string, "id": int},...], "user": string, "error": {"Code": int, "Message": string}}</code>

```

/* Fonction qui va faire une requête à L'API en envoyant Le token de L'utilisateur
pour recevoir Le nom et L'ID de ses traqueurs */
async getTrackers() {
  try {
    /* Fait une requête à L'API en Lui donnant Le token de L'utilisateur */
    const response = await fetch(
      'https://api.ovl.tech-user.fr/iot_list/' + this.props.token
    );
    /* Met la réponse de L'API dans la variable 'json' */
    const json = await response.json();
    console.log(json);
    /* Si il n'y a pas d'erreur, met Le nom et L'ID des traqueurs dans Le tableau 'data' */
    if (json.error.Message == 'Nothing goes wrong.') {
      this.setState({ data: json.trackers.iotArray });
    } else {
      /* Sinon, affiche une erreur */
      alert('Unknown error');
    }
  } catch (error) {
    /* Si L'API ne répond pas, affiche une erreur indiquant que L'API ne fonctionne pas */
    console.log(error);
    alert('API down');
  }
}

```

Pour faire l'affichage des boutons correspondant aux traqueurs, j'appelle la fonction ci-dessous pour chacun d'eux en envoyant en paramètre leur ID et leur nom.

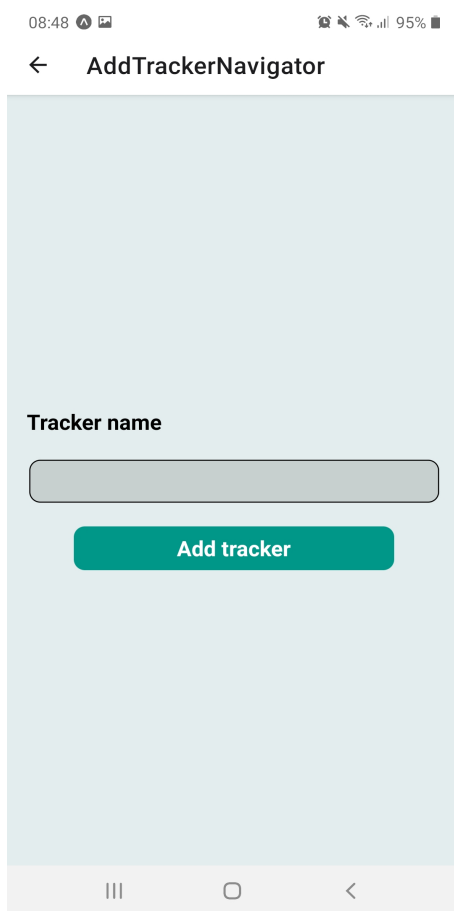
```

/* Affiche un boutons avec Le nom du traqueur reçu.
Reçoit en paramètre Le token de L'utilisateur, Le nom de son traqueur avec son ID et La variable 'navigation' */
const ButtonCreator = ({
  trackerName,
  trackerId,
  navigation,
  token,
}: ItemProps) => (
  <View style={Styles.item}>
    { /* Si Le bouton est cliqué, passe à La page qui affiche Les coordonnées du traqueur */ }
    <TouchableOpacity
      style={TrackerStyles.trackerButtonContainer}
      onPress={() =>
        navigation.navigate('CoordinatesNavigator', {
          trackerId: trackerId,
          navigation: navigation,
          token: token,
        })
      }
    >
      <Text style={TrackerStyles.appButtonText}>{trackerName}</Text>
    </TouchableOpacity>
  </View>
);

```

### 3.4 – Ajout d'un traqueur :

Si l'utilisateur veut ajouter un traqueur à son compte, il peut appuyer sur le bouton «AddTracker» sur la page de sélection du traqueur.



The screenshot shows a mobile application interface for adding a tracker. At the top, the status bar displays the time 08:48, signal strength, and battery level at 95%. Below the status bar, a navigation bar contains a back arrow and the title "AddTrackerNavigator". The main content area has a light blue background. It features a label "Tracker name" above a text input field. Below the input field is a green button labeled "Add tracker". At the bottom of the screen is an Android-style navigation bar with three icons: a square, a circle, and a triangle.

Une fois le nom du traqueur choisi, l'utilisateur peut appuyer sur le bouton «Add Tracker» pour exécuter la fonction suivante. Celle-ci envoie à l'API le token de l'utilisateur et le nom saisi par l'utilisateur et récupère l'ID du nouveau traqueur ainsi que le topicTX et topicRX à rentrer dans le traqueur. L'utilisateur est aussi renvoyé à la page précédente.

POST	/iot/{token}/{name}
Used to	Add and link a tracker to the account and get the parameters needed to configure the tracker
Json Output	<code>{"error": {"Code": int, "Message": string}}, "Topics": {"RX": string, "TX": string}, TrackerId: int</code>

```

/* Permet de faire une requête à L'API pour ajouter un traqueur
Reçoit en paramètre le token de L'utilisateur, le nom de son traqueur et la variable 'navigation' */
const AddTracker = (token, trackerName, navigation) => {
  /* Si L'utilisateur n'a pas saisi le nom de son nouveau traqueur, affiche une erreur */
  if (trackerName == '') {
    alert('Please enter the name of your tracker');
  } else {
    /* Sinon, fait une requête 'PUT' à L'API en envoyant le token de L'utilisateur et le nom du traqueur qui a été sélectionné */
    fetch('https://api.ovl.tech-user.fr/iot', {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json',
      },
      /* Données envoyées à L'API */
      body: JSON.stringify({
        token: token,
        name: trackerName,
      }),
    })
    /* Récupère la réponse de L'API */
    .then((response) => response.json())
    .then((json) => {
      console.log(json);
      /* Si il n'y a pas eu d'erreur, affiche le nom du traqueur, son ID et ses coordonnées et envoie l'utilisateur à la page de sélection des traqueurs en renvoyant le token */
      if (json.error.Message == 'Nothing goes wrong.') {
        alert(
          'Tracker name : ' + trackerName + '\n' +
          'Tracker ID : ' + json.TrackerId + '\n' +
          'TopicTX : ' + json.Topics.TX + '\n' +
          'TopicRX : ' + json.Topics.RX
        );
        navigation.navigate('TrackerNavigator', {
          token: token,
          navigation: navigation,
        });
        /* Sinon, affiche une erreur */
      } else {
        alert('There was an error when adding the tracker');
      }
    })
    /* Si L'API ne répond pas, affiche une erreur indiquant que L'API ne fonctionne pas */
    .catch((error) => {
      console.error(error);
      alert('API down');
    });
  }
};

```

### 3.5 – Affichage des coordonnées :

Lorsque l'utilisateur a choisi son traqueur sur la page de sélection, la page suivante s'affiche. Les dernières coordonnées du traqueur connues ainsi que la date correspondant à celles-ci sont affichées. Le bouton «Refresh» actualise la position, le bouton «Zoom» zoom (faisable aussi à la main) et le bouton «Extend» agrandi la carte.



Au lancement de la page, la fonction «getPos» se lance. Elle envoie une requête GET à l'API en spécifiant l'ID du traqueur sélectionné puis stocke la position et la date reçu.

GET	/position/now/{id}
Used to	Get the current position of the tracker based on his id.
Json Output	{position: {"lon": float, "lat": float}, "error": {"Code": int, "Message": string}}

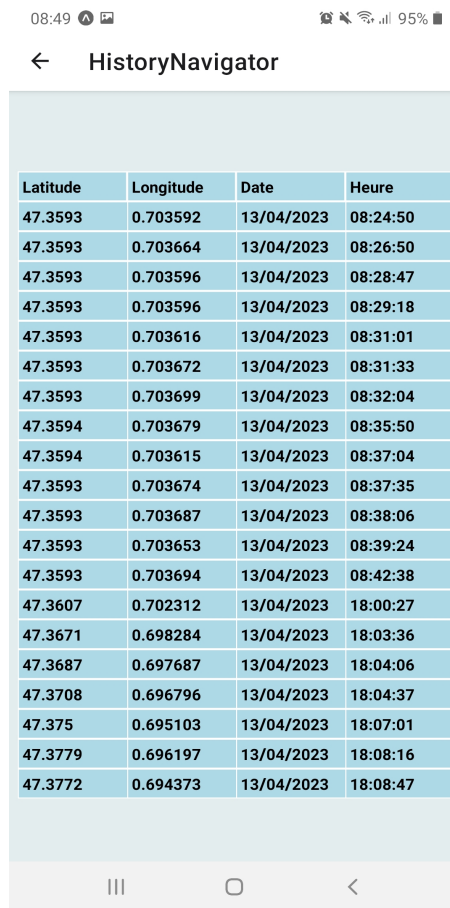
```
/* Fonction qui va faire une requête à l'API en envoyant l'ID d'un traqueur afin de récupérer la dernière position enregistré de ce traqueur */
async getPos() {
  try {
    /* Faire une requête à l'API en lui donnant l'ID d'un traqueur */
    const response = await fetch(
      'https://api.ovl.tech-user.fr/position/now/' + this.props.trackerId
    );
    /* Attend la réponse de l'API et la met en json dans la variable 'json' */
    const json = await response.json();
    console.log(json);
    /* Si il n'y a pas d'erreur, stock les données envoyées par l'API */
    if (json.error.Message == 'Nothing goes wrong.') {
      /* Met sous forme de date le timestamp reçu */
      var tempDate = new Date(Number(json.now.timestamp) * 1000);
      /* Formate la date pour la mettre sous la forme 'jj/mm/aaaa hh:mm:ss' */
      var formattedDate =
        ('0' + tempDate.getDate()).slice(-2) +
        '/' +
        ('0' + (tempDate.getMonth() + 1)).slice(-2) +
        '/' +
        tempDate.getFullYear() +
        ' ' +
        ('0' + tempDate.getHours()).slice(-2) +
        ':' +
        ('0' + tempDate.getMinutes()).slice(-2) +
        ':' +
        ('0' + tempDate.getSeconds()).slice(-2);
      console.log('Timestamp : ' + json.now.timestamp);
      console.log('Formatted date : ' + formattedDate);
      console.log('Latitude : ' + json.now.lat);
      console.log('Longitude : ' + json.now.lon);
      /* Stock la longitude, la latitude et la date de la dernière position enregistrée dans les attributs de la class*/
      this.setState({
        date: formattedDate,
        latitude: json.now.lat,
        longitude: json.now.lon,
      });
      /* Sinon, affiche une erreur */
    } else {
      alert('Erreur inconnu');
    }
  } catch (error) {
    /* Si l'API ne répond pas, affiche une erreur indiquant que l'API ne fonctionne pas */
    console.log(error);
    alert('API down');
  }
}
```

Pour l'affichage de la carte et du marqueur, il suffit de mettre une balise «MapView» et une «Marker» et de spécifier la longitude et latitude.

```
<MapView
  region={{
    latitude: Number(this.state.latitude),
    longitude: Number(this.state.longitude),
    latitudeDelta: this.state.latDelta,
    longitudeDelta: this.state.lonDelta,
  }}
  style={{
    width: this.state.width,
    height: this.state.height,
    alignSelf: 'center',
    margin: 10,
  }}>
  <Marker
    coordinate={{
      latitude: Number(this.state.latitude),
      longitude: Number(this.state.longitude),
      latitudeDelta: 0.1,
      longitudeDelta: 0.1,
    }}
  />
</MapView>
```

### 3.6 – Historique :

Sur la page affichant les coordonnées, l'utilisateur peut appuyer sur le bouton «History» pour afficher cette page. L'utilisateur peut y consulter les coordonnées des dernières 48 heures.



Latitude	Longitude	Date	Heure
47.3593	0.703592	13/04/2023	08:24:50
47.3593	0.703664	13/04/2023	08:26:50
47.3593	0.703596	13/04/2023	08:28:47
47.3593	0.703596	13/04/2023	08:29:18
47.3593	0.703616	13/04/2023	08:31:01
47.3593	0.703672	13/04/2023	08:31:33
47.3593	0.703699	13/04/2023	08:32:04
47.3594	0.703679	13/04/2023	08:35:50
47.3594	0.703615	13/04/2023	08:37:04
47.3593	0.703674	13/04/2023	08:37:35
47.3593	0.703687	13/04/2023	08:38:06
47.3593	0.703653	13/04/2023	08:39:24
47.3593	0.703694	13/04/2023	08:42:38
47.3607	0.702312	13/04/2023	18:00:27
47.3671	0.698284	13/04/2023	18:03:36
47.3687	0.697687	13/04/2023	18:04:06
47.3708	0.696796	13/04/2023	18:04:37
47.375	0.695103	13/04/2023	18:07:01
47.3779	0.696197	13/04/2023	18:08:16
47.3772	0.694373	13/04/2023	18:08:47



Comme pour l'affichage des coordonnées, la fonction est exécutée au lancement de la page et fait une requête à l'API pour recevoir l'historique de position. Les latitudes, longitudes et dates sont ensuite stockées dans des tableaux.

GET	/position/history/{id}
Used to	Obtain all position history of a tracker based on his id.
Json Output	<code>{"history": [{"lat": float, "lon": float, "timestamp": int},...], "error": {"Code": int, "Message": string}}</code>

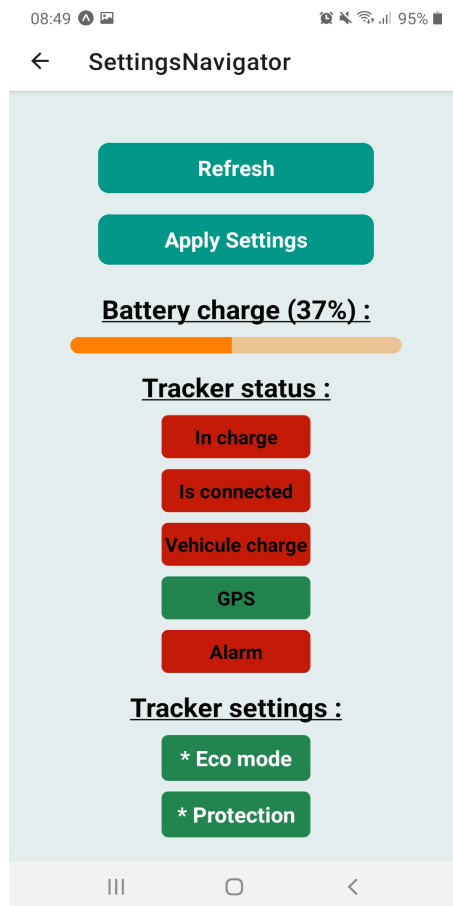
```

/* Fonction qui va faire une requête à L'API en lui donnant L'ID d'un traqueur afin de
récupérer toutes les positions du traqueur enregistrées les dernières 48 heures */
async getHistory() {
  try {
    /* Faire une requête à L'API en lui donnant le token L'utilisateur */
    const response = await fetch(
      'https://api.ovl.tech-user.fr/position/history/' + this.props.trackerId
    );
    /* Attend la réponse de L'API et la met en json dans la variable 'json' */
    const json = await response.json();
    console.log(json);
    /* Si il n'y a pas d'erreur, stock les données envoyées par L'API */
    if (json.error.Message == 'Nothing goes wrong.') {
      console.log(
        "Nombre de position dans l'historique : " + json.history.length
      );
      var hour = [];
      var date = [];
      /* Boucle le nombre de fois qu'il y a de position dans L'historique */
      for (var i = 0; i < json.history.length; i++) {
        /* Met sous forme de date le timestamp reçu */
        var tempDate = new Date(Number(json.history[i].timestamp) * 1000);
        /* Formate la date pour la mettre sous la forme 'jj/mm/aaaa' */
        date[i] =
          ('0' + tempDate.getDate()).slice(-2) +
          '/' +
          ('0' + (tempDate.getMonth() + 1)).slice(-2) +
          '/' +
          tempDate.getFullYear();
        /* Formate l'heure pour la mettre sous la forme 'hh:mm:ss' */
        hour[i] =
          ('0' + tempDate.getHours()).slice(-2) +
          ':' +
          ('0' + tempDate.getMinutes()).slice(-2) +
          ':' +
          ('0' + tempDate.getSeconds()).slice(-2);
      }
      /* Stock dans les attributs de la class la date, l'heure et l'historique de position */
      this.setState({
        data: json.history,
        date: date,
        hour: hour,
      });
      /* Sinon affiche une erreur */
    } else {
      alert('Erreur inconnu');
    }
  } catch (error) {
    /* Si L'API ne répond pas, affiche une erreur indiquant que L'API ne fonctionne pas */
    console.log(error);
    alert('API down');
  }
}

```

### 3.7 – Paramètres :

Comme pour voir l'historique, l'utilisateur peut appuyer sur le bouton «Settings» pour afficher cette page. L'utilisateur peut y consulter les statuts de son traqueur et changer ses paramètres.



La fonction suivante est exécutée au lancement de la page et fait une requête GET à l'API pour recevoir les statuts du traqueur en spécifiant le token de l'utilisateur.

GET	/status/{id_iot}
Used to	Get the status of the requested tracker from his id.
Json Output	<pre>status: {"bat": int, "charge": bool, "vehiculechg": bool, "protection": bool, "ecomode": bool, "alarm": bool, "protection": bool, "gps": bool, "error": {"Code": int, "Message": string}}</pre>

```
/* Fait le requête à l'API en envoyant le token de l'utilisateur */
const response = await fetch(
  'https://api.ovl.tech-user.fr/status_list/' + this.props.token
);
/* Récupère la réponse dans la variable 'json' */
const json = await response.json();
/* Met dans 'data' le json */
this.setState({
  data: json.status_list[this.props.trackerId - 1],
});
console.log(json.status_list[this.props.trackerId - 1]);
```

Une fois que l'utilisateur a changé les paramètres souhaité et qu'il a appuyé sur «ApplySettings», l'application enverra les nouveaux statuts à l'API avec une requête PUT en spécifiant l'ID du traqueur.

PUT	/set/status/{id_iot}/{status_alarm}/{status_ecomode}/{status_protection}/{status_vh_charge}
Used to	Set new iot_device settings.
Json Output	"error": {"Code": int, "Message": string}}

```
/* Applique Les settings changés
Reçoit en paramètre L'ID d'un traqueur ainsi que le statut de l'alarm, du mode économique, de la protection et de la charge du véhicule */
putSettings = (trackerId, alarm, ecoMode, protection, vhCharge) => {
  /* Réinitialise Les statuts de changement */
  this.setState({
    ecoChanged: '',
    protectionChanged: '',
  });
  /* Fait une requête à L'API pour changer Les paramètres du traqueur */
  fetch('https://api.ovl.tech-user.fr/set/status/', {
    method: 'PUT',
    headers: {
      Accept: 'application/json',
      'Content-Type': 'application/json',
    },
  },
  /* Envoie L'ID du traqueur et Le statut de chaque paramètre */
  body: JSON.stringify({
    id_iot: trackerId,
    status_alarm: alarm,
    status_ecomode: ecoMode,
    status_protection: protection,
    status_vh_charge: vhCharge,
  })),
  )
  /* Récupère La réponse de L'API */
  .then((response) => response.json())
  .then((json) => {
    console.log(json);
    /* Si Le changement s'est bien effectué, affiche que ça s'est bien effectué */
    if (json.error.Message == 'Nothing goes wrong.') {
      alert('Changement appliqués !');
    } else {
      alert("Erreur lors de l'application des changements.");
    }
  })
  .catch((error) => {
    /* Si L'API ne répond pas, affiche une erreur indiquant que L'API ne fonctionne pas */
    console.log(error);
    alert('API down');
  });
};
```

## 4 – Problèmes rencontrés

### 4.1 – Problèmes rencontrés :

La partie la plus compliqué du projet à été pour moi d'apprendre le langage React Native puisque je n'avais jamais coder dans ce langage. De plus, mettre en place les notifications a été difficile et je n'ai pas réussi à les faire fonctionner sur Android.

## 5 – Tests

<input checked="" type="checkbox"/> <b>Test unitaire</b>				
<b>Objectif :</b> Open Vehicule Locator – Créer un compte				
<b>Éléments à tester :</b> Méthode « RegisterScreen »				
<b>Conditions de réalisation :</b>  Besoins matériels : - Smartphone IOS ou Android ou PC connecté à internet  Besoins logiciels (Versions) : - Application Expo Go sur mobile ou snack.expo.dev sur le web  Nom(s) du (des) fichier(s) de test : - test_register				
<b>Initialisation :</b>				
<b>Scénario</b>				
ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status : OK / NOK
1	Démarrer l'application		Affiche la page de connexion	OK
2	Appuyer sur le bouton «Register»		Affiche la page d'inscription	OK
3	Rentrer un e-mail et deux fois un mot de passe valide dans les champs spécifiés	E-mail: test2@gmail.com Mot de passe: mdp Confirmation mot de passe: mdp	Affiche l'ID du traqueur, le topicTX et le topicRX Reviens à la page de connexion	OK

Rapport de test	Testé par : Etudiant2	Le : 03/05/2023
<b>Commentaire : Le test fonctionne parfaitement</b>		
	<b>Approbation : Oui</b>	<b>Signature:</b>

## ☒ Test unitaire

**Objectif :** Open Vehicule Locator – Se connecter à son compte

**Éléments à tester :** Méthode « LoginScreen »

**Conditions de réalisation :**

Besoins matériels :

- Smartphone IOS ou Android ou PC connecté à internet

Besoins logiciels (Versions) :

- Application Expo Go sur mobile ou snack.expo.dev sur le web

Nom(s) du (des) fichier(s) de test :

- test\_login

**Initialisation :**

### Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status : OK / NOK
1	Démarrer l'application		Affiche la page de connexion	OK
2	Rentrer un e-mail et un mot de passe valide dans les champs spécifiés	E-mail: test@gmail.com Mot de passe: mdp	Affiche la page de sélection du traqueur	OK

Rapport de test

Testé par : Etudiant2

Le : 07/03/2023

**Commentaire : Le test fonctionne parfaitement**

**Approbation : Oui**

**Signature:**

## ☒ Test unitaire

**Objectif :** Open Vehicule Locator – Ajouter un traqueur à son compte

**Éléments à tester :** Méthodes « addTrackerScreen » & « trackerScreen »

### Conditions de réalisation :

Besoins matériels :

- Smartphone IOS ou Android ou PC connecté à internet

Besoins logiciels (Versions) :

- Application Expo Go sur mobile ou snack.expo.dev sur le web

Nom(s) du (des) fichier(s) de test :

- test\_addTracker

### Initialisation :

#### Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status : OK / NOK
1	Démarrer l'application	L'utilisateur est déjà connecté	Affiche la page de sélection du traqueur	OK
2	Appuyer sur le bouton « Add tracker »		Affiche la page pour ajouter un traqueur	OK
3	Rentrer le nom du traqueur	Nom du traqueur: tracker2	Affiche l'ID du traqueur, le topicTX et le topicRX Reviens à la page de connexion	OK

Rapport de test

Testé par : Etudiant2

Le : 07/03/2023

**Commentaire : Le test fonctionne parfaitement**

**Approbation : Oui**

**Signature:**

## ☒ Test unitaire

**Objectif :** Open Vehicule Locator – Sélectionner son traqueur et afficher ses coordonnées GPS

**Éléments à tester :** Méthodes « coordinatesScreen » & « trackerScreen »

### Conditions de réalisation :

Besoins matériels :

- Smartphone IOS ou Android ou PC connecté à internet

Besoins logiciels (Versions) :

- Application Expo Go sur mobile ou snack.expo.dev sur le web

Nom(s) du (des) fichier(s) de test :

- test\_coordinates

### Initialisation :

#### Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status : OK / NOK
1	Démarrer l'application		Affiche la page de connexion	OK
2	Rentrer un e-mail et un mot de passe valide dans les champs spécifiés	E-mail: test@gmail.com Mot de passe: mdp	Affiche la page de sélection du traqueur	OK
3	Sélectionner un traqueur en appuyant sur le bouton à son nom		Affiche la page contenant les coordonnées GPS du traqueur avec sa position sur une carte ainsi que la date de dernière mise à jour	OK

Rapport de test

Testé par : Etudiant2

Le : 07/03/2023

**Commentaire : Le test fonctionne parfaitement**

Approbation : Oui

Signature:

☒ **Test unitaire**

**Objectif** : Open Vehicule Locator – Afficher l'historique de position du traqueur

**Éléments à tester** : Méthodes « historyScreen »

**Conditions de réalisation** :

Besoins matériels :

- Smartphone IOS ou Android ou PC connecté à internet

Besoins logiciels (Versions) :

- Application Expo Go sur mobile ou snack.expo.dev sur le web

Nom(s) du (des) fichier(s) de test :

- test\_history

**Initialisation** :

**Scénario**

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status : OK / NOK
1	Appuyer sur le bouton «History»	L'utilisateur est déjà connecté et à déjà sélectionné son traqueur	Affiche l'historique des coordonnées du traqueur (date+heure+position)	OK

Rapport de test

Testé par : Etudiant2

Le : 07/03/2023

**Commentaire** : Le test fonctionne parfaitement

**Approbation** : Oui

**Signature:**



## ☒ Test unitaire

**Objectif :** Open Vehicule Locator – Afficher les statuts du traqueur

**Éléments à tester :** Méthodes « settingsScreen »

### Conditions de réalisation :

Besoins matériels :

- Smartphone IOS ou Android ou PC connecté à internet

Besoins logiciels (Versions) :

- Application Expo Go sur mobile ou snack.expo.dev sur le web

Nom(s) du (des) fichier(s) de test :

- test\_settings

### Initialisation :

#### Scénario

ID	Démarche / Opération	Données en entrée / Condition initiale	Comportement / Résultat attendu	Status : OK / NOK
1	Démarrer l'application	L'utilisateur est déjà connecté et à déjà sélectionné son traqueur	Affiche la page contenant les coordonnées GPS du traqueur	OK
2	Appuyer sur le bouton «Settings»		Affiche les statuts des traqueurs et ses paramètres	OK
3	Appuyer sur le bouton d'un des paramètres	Appuyer sur «Eco mode»	Change l'apparence du bouton pour indiquer le changement	OK
4	Appuyer sur le bouton «Apply settings»	Statut des paramètres (ON/OFF)	Affiche le succès de la modification des paramètres	OK

Rapport de test

Testé par : Etudiant2

Le : 07/03/2023

**Commentaire : Le test fonctionne parfaitement**

**Approbation : Oui**

**Signature:**