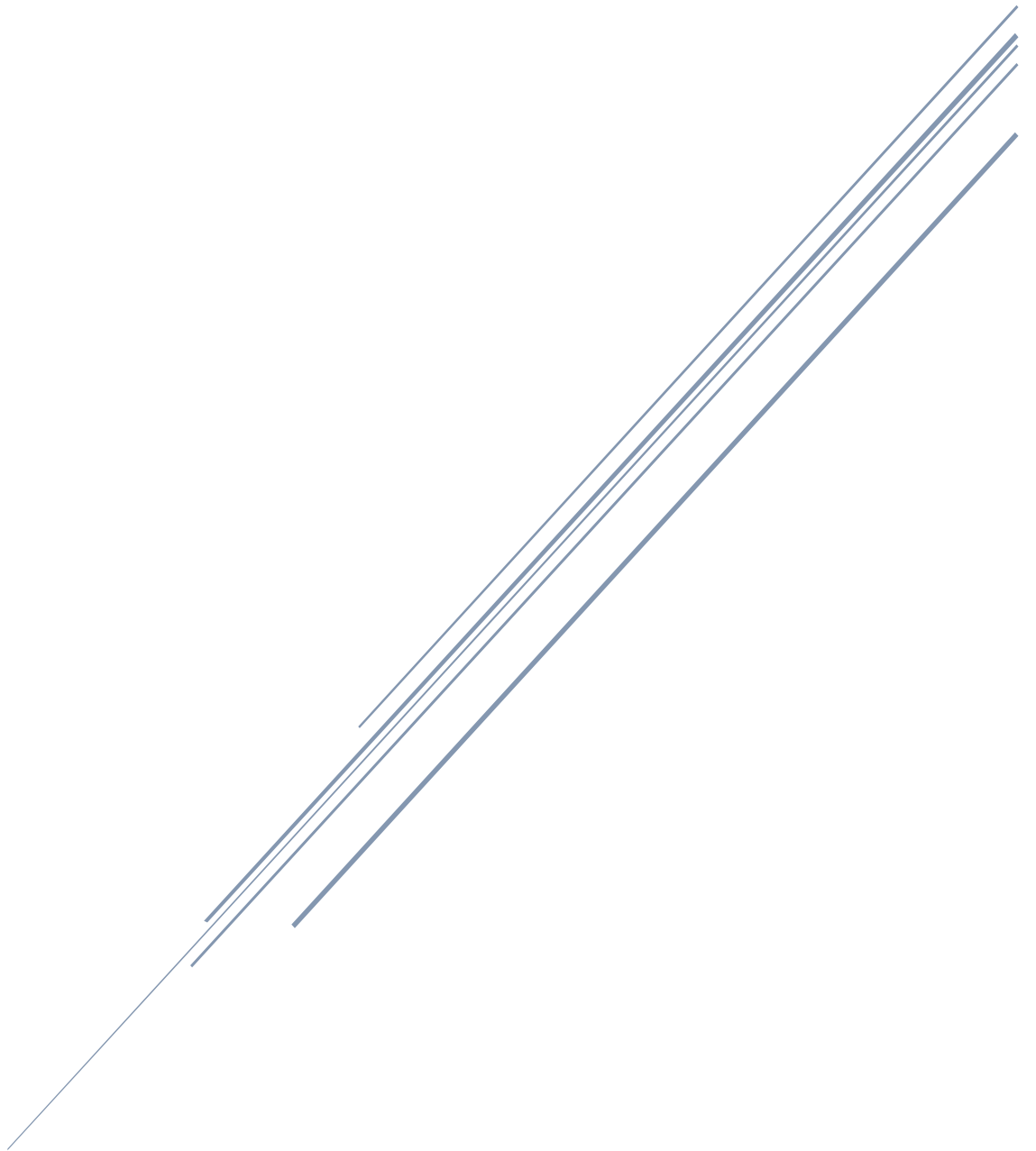


DRIVE



Mémoire réalisé par Renaud COLOMAR.

DEV 34-21-01

Table des matières

1. Remerciement	p. 2
2. Abstract	p. 3
3. Introduction	p. 4
4. Compétences du Titre couvertes par le Projet	p. 5
5. Analyse du besoin	p. 6
a. Présentation du projet	p. 6
b. Contexte – Besoin	p. 7
c. Contraintes techniques	p. 8
6. Spécifications fonctionnelles	p. 10
a. Cas d'utilisation (Use Case)	p. 10
b. Diagramme d'activité	p. 11
i. Diagramme de création d'un compte Utilisateur	p. 11
ii. Diagramme de connexion	p. 13
c. Diagramme de séquence	p. 14
i. Diagramme de création d'un compte Utilisateur	p. 14
ii. Diagramme de connexion	p. 16
d. Maquettage	p. 17
7. Conception	p. 21
a. Modèle Conceptuel des Données (MCD)	p. 21
b. Modèle Logique des Données (MLD)	p. 24
c. Création de la base de données	p. 25
8. Arborescence	p. 29
9. Outils techniques utilisés	p. 30
10. Fonctionnalités	p. 31
a. Architecture	p. 31
b. Connexion d'un Utilisateur	p. 34
c. Inscription d'un Utilisateur	p. 36
d. Consultation d'un compte Utilisateur	p. 40
11. Conclusion	p. 43

1) Remerciements

Je tiens à remercier toute l'équipe de formateurs de l'ADRAR pour les connaissances qu'ils m'ont transmis et leur accompagnement.

Je remercie également Titouan FOURREAU, mon maître de stage, qui m'a confié et fait confiance sur le développement de cette application.

Je remercie également les autres membres de la session DEV-DEVOPS 34-21-01 pour l'aide qu'ils ont pu m'apporter et pour la bonne humeur qu'il y a eu au sein du groupe tout au long de la formation.

2) Abstract

Manager in a chain of garden centers for 15 years, at 49 years old and after 22 years of passion in the animal trade, I decided to start a reconversion, driven by a strong need for renewal. 2 years ago, I carried out a skills assessment that allowed me to take stock of my life, my expectations and my future prospects. It is my interest in new technologies from a very young age, computer science and professional opportunities that motivated my approach to undertake a web developer training at ADRAR.

During this course, I did a 2-month internship at the company Instadrone which offers technical services by drones such as the capture of aerial and underwater topographic data, radio studies for telecommunications, Photogrammetric processing and 3D modeling. The project entrusted to me is a mobile application allowing the monitoring of the company's car during the travel of its technicians. Several constraints were imposed on me: develop in React Native, the graphic model and the database on Google Cloud server. I worked on this application in total autonomy being the only developer within the company. It is this project that I will present to you in this brief.

3) Introduction

Manager dans une chaîne de jardineries pendant 15 ans, à 49 ans et après 22 ans de passion dans le commerce animalier, j'ai décidé de me lancer dans une reconversion, poussé par un fort besoin de renouveau. Il y a 2 ans j'ai effectué un bilan de compétences qui m'a permis de faire le point sur ma vie, mes attentes et mes perspectives d'avenir. C'est mon intérêt pour les nouvelles technologies depuis mon plus jeune âge, l'informatique et les opportunités professionnelles qui ont motivés ma démarche d'entreprendre une formation de développeur web à l'ADRAR.

Au cours de ce cursus, j'ai effectué un stage de 2 mois au sein de l'entreprise Instadrone qui offre des prestations techniques par drones telles que la captation de données topographiques aériennes et sous-marines, les études radio pour les télécommunications, le traitement Photogrammétrique et la modélisation 3D. Le projet qui m'a été confié est le développement d'une application mobile permettant le suivi de la flotte automobile de l'entreprise lors du déplacement de ses techniciens. Plusieurs contraintes m'ont été imposées : développer en React Native, la maquette graphique et la base de données sur serveur Google Cloud. J'ai travaillé sur cette application en totale autonomie étant le seul développeur au sein de l'entreprise. C'est ce projet que je vais vous présenter dans ce mémoire.

4) Compétences du Titre couvertes par le Projet

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles	Mémoire	Dossier Pro(DP)
1	Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	1	Maquetter une application	✓	
		2	Réaliser une interface utilisateur web statique et adaptable	✓	
		3	Développer une interface utilisateur web dynamique	✓	
		4	Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce		✓
2	Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	5	Créer une base de données	✓	
		6	Développer les composants d'accès aux données	✓	
		7	Développer la partie back-end d'une application web ou web mobile	✓	
		8	Elaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce		✓

5) Analyse du besoin

a) Présentation du projet

Instadrone est une entreprise créée en 2014 qui offre des prestations techniques par drones telles que la captation de données topographiques aériennes et sous-marines, les études radio pour les télécommunications, le traitement Photogrammétrique et la modélisation 3D.

Le projet du PDG de l'entreprise, M. Cédric BOTELLA, est de développer une application simple d'utilisation pour la gestion de sa flotte de véhicules. Mon stage était consacré à la réalisation de ce projet appelé provisoirement « DRIVE ».

L'application devra également être adaptable et permettre d'étendre son usage à d'autres entreprises ayant potentiellement les mêmes besoins (Entreprise de transport, entreprise de livraison, etc...) ou de ses équipements (drones, véhicules de travaux, etc...)

Elle permettra à un utilisateur de s'identifier, de choisir un véhicule dans une liste ou le scan d'un QR code parmi le parc automobile de l'entreprise, de prendre et enregistrer des photos du véhicule, modifier le kilométrage à l'emprunt et à la restitution du véhicule, d'enregistrer les frais de repas et de carburant lors du déplacement du technicien ainsi qu'un suivi par géolocalisation du trajet.

b) Contexte – Besoin

i) Contexte

Suite à divers incidents sur les véhicules et accessoire de l'entreprise, M. BOTTELA à penser à une application mobile qui permettrait d'avoir un suivi et un historique de l'état de chaque véhicule et de son utilisateur. Dans un premier temps, l'idée est de répondre à un besoin interne de l'entreprise. Mais l'objectif est également de pouvoir étendre son application à d'autres secteurs d'activité afin de proposer ce service à certains de ses clients.

Mon maître de stage M. FOURREAU, n'ayant ni le temps ni les compétences pour mener à bien ce projet, m'a donc confié le développement de l'application lors de mon stage. Lors de mon premier jour de stage, Il m'a été expliqué les besoins et contraintes décrites ci-dessous.

ii) Besoins

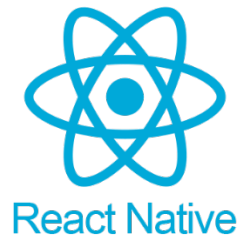
Afin de répondre aux demandes de M. BOTTELA, nous avons établi que l'application devra comprendre les composants suivants :

- L'identification de l'utilisateur
- Le choix d'un véhicule parmi le parc de l'entreprise
- Enregistrer les photos de l'état du véhicule
- Modifier le kilométrage du véhicule
- L'enregistrement des frais de repas et de carburant
- Un suivi par géolocalisation du déplacement

c) Contraintes techniques

Afin de développer une application répondant aux exigences de l'entreprise, il m'a été imposé les contraintes suivantes.

- **Le Framework :**



Il m'a été demandé de développer en React Native qui est un Framework d'applications mobiles open source créé par Facebook. Il est utilisé pour développer des applications pour Android, iOS en permettant aux développeurs d'utiliser React avec les fonctionnalités natives de ces plateformes.

- **Le langage :**



JSX est une extension React de la syntaxe du langage JavaScript qui permet de structurer le rendu des composants à l'aide d'une syntaxe familière à de nombreux développeurs. Il est similaire en apparence au HTML.

- **La maquette :**



La maquette avait été préparée mon maitre de stage sur Adobe XD, qui est un outil de conception d'expérience utilisateur pour les applications web et les applications mobiles, développé et publié par Adobe Inc.



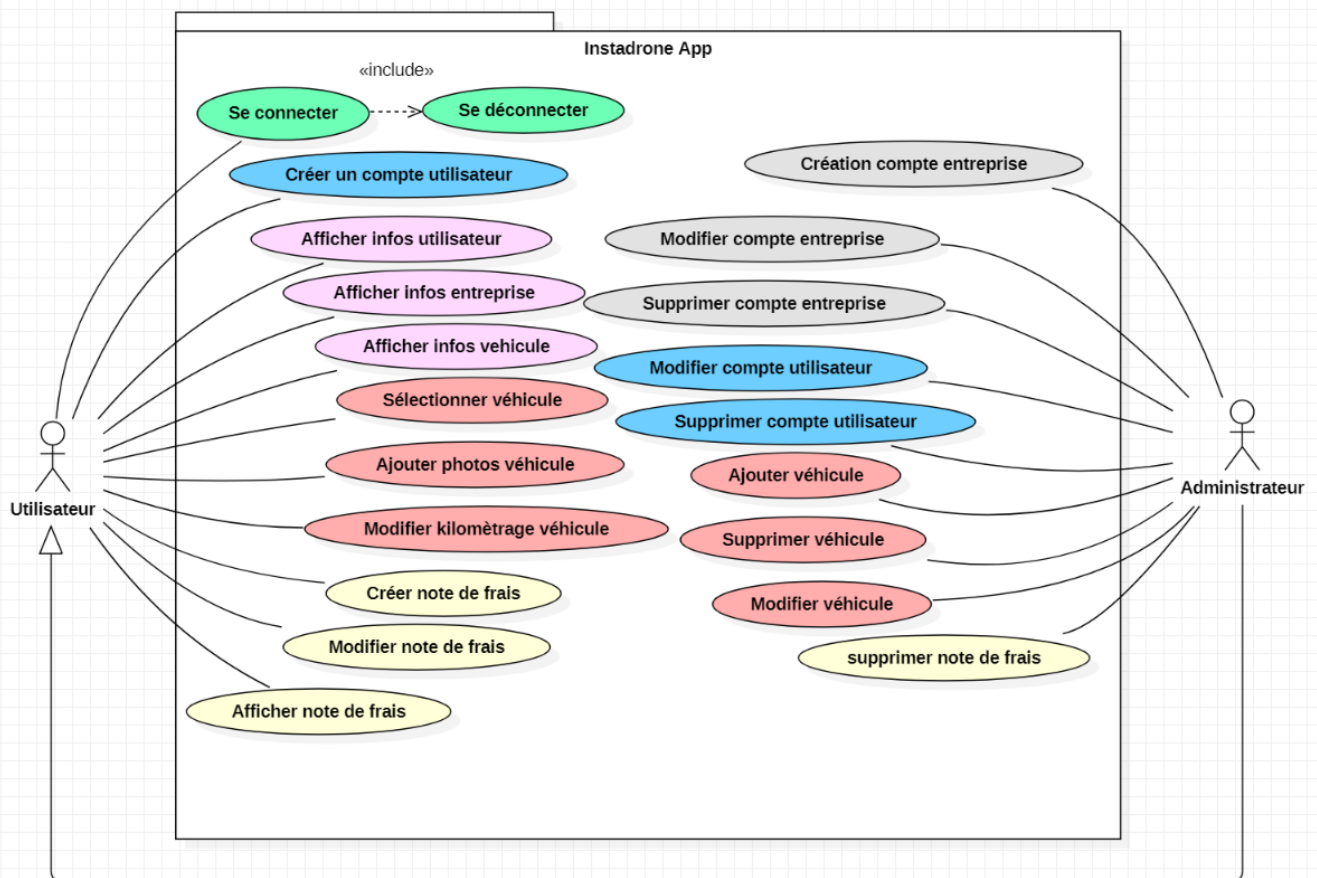
6) Spécifications fonctionnelles

a) Cas d'utilisation (Use case)

Le diagramme de cas d'utilisation permet de décrire, dans des documents lisibles par tous, la finalité des interactions du système et des différents acteurs.

Dans mon projet, celui-ci comporte deux acteurs :

- L'utilisateur devra se connecter pour accéder à l'application. S'il n'a pas de compte utilisateur, il pourra s'inscrire. Une fois connecté, il pourra afficher ses propres informations ainsi que celles de son entreprise. Il aura également la possibilité de modifier ses informations personnelles. Une fois connecté, l'utilisateur pourra sélectionner un véhicule, ajouter des photos du véhicule, modifier son kilométrage. Il aura également la possibilité de créer des notes de frais, d'ajouter une photo de la note de frais, de modifier la note et d'afficher ses détails. Et enfin, l'utilisateur pourra se déconnecter.
- L'administrateur héritera des fonctionnalités de l'utilisateur. Mais il aura également la possibilité de créer un compte d'entreprise, de le modifier, l'afficher et le supprimer. Il pourra également supprimer un compte utilisateur. Il ajoutera des véhicules, les supprimera ou modifiera leurs informations. Enfin, il aura la possibilité de supprimer les notes de frais.

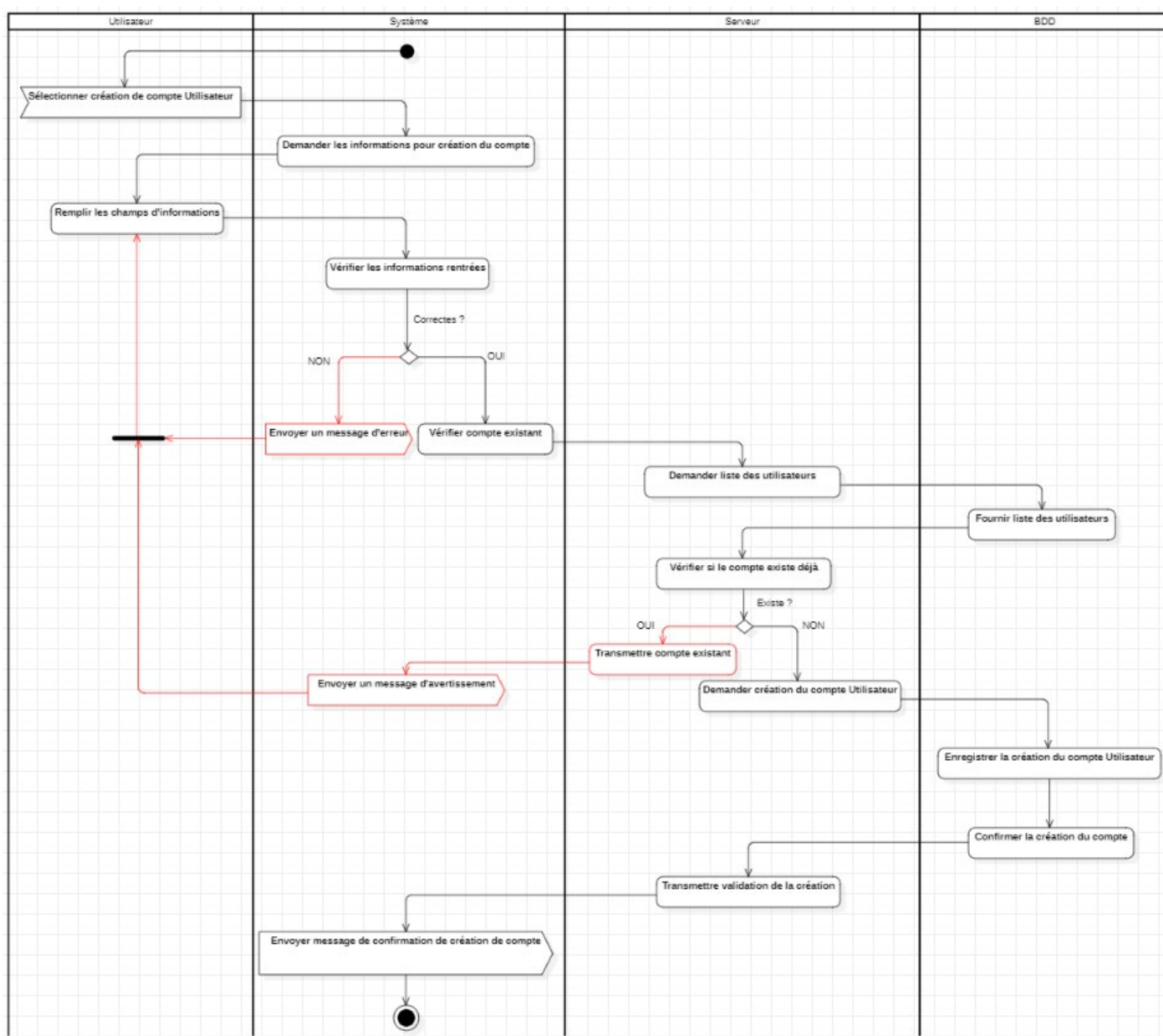


b) Diagramme d'activité

Le diagramme d'activité fournit une vue du comportement d'un système en décrivant la séquence d'action d'un processus. Les diagrammes sont modélisés avec StarUML.



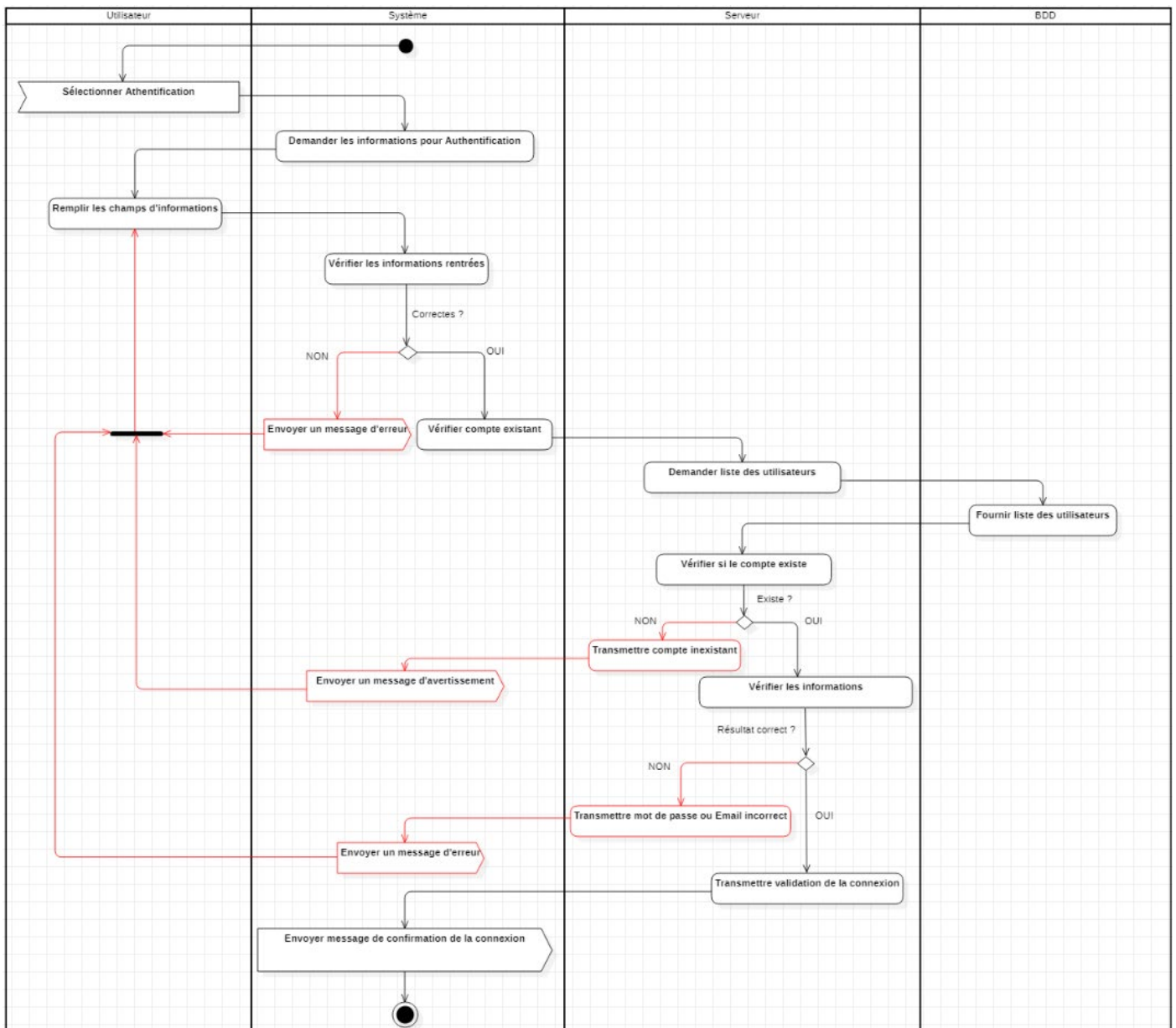
i) Diagramme de la création d'un compte utilisateur



Le **nœud initial** indique le début de l'activité, il se situe dans le système. Le **flot de contrôle** se dirige vers l'utilisateur qui reçoit un **signal d'acceptation** :

- L'utilisateur clique sur création de compte utilisateur, le flot transit vers le système qui exécute une action de demande d'informations.
- L'utilisateur saisie les informations demandées, le flot transite vers le système qui reçoit les informations.
- Le système vérifie ces informations. Un **nœud de décision** effectue un choix :
 - Si les informations sont correctes, le flot transit vers le serveur.
 - Si les informations sont incorrectes, une action « **send signal** » en notifie l'utilisateur. On reste dans le système qui demande à nouveau les informations pour la création du compte.
- Le serveur demande la liste des comptes utilisateurs à la base de données.
- La base données fournis la liste des Utilisateurs au serveur.
- Le Serveur reçoit ces informations et les vérifie. Un **nœud de décision** effectue un choix :
 - Si le compte existe déjà dans la liste des utilisateurs, le serveur transmet l'information au Système, une action « **send signal** » en notifie l'utilisateur. On reste dans le Système qui demande à nouveau les informations pour la création du compte.
 - Sinon, le flot transite vers la Base de données.
- La Base de données exécute la requête de création de compte Utilisateur. Elle confirme au Serveur que le compte est créé.
- Le serveur transmet la confirmation de la création du compte au Système.
- Le système signal à l'Utilisateur par un message que le compte est créé. Le flot d'exécution atteint un **nœud de fin d'activité**, la fonctionnalité se termine.

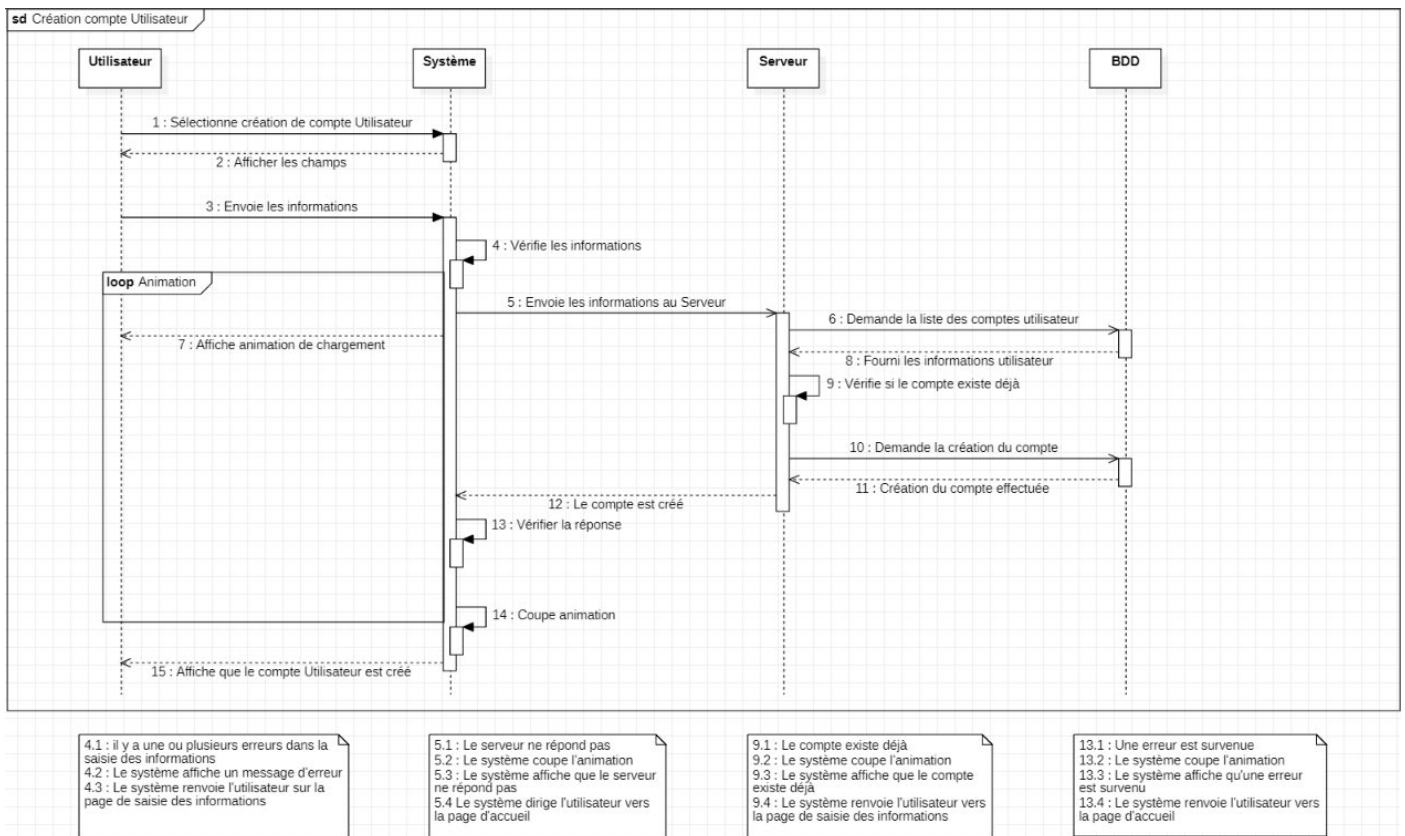
ii) Diagramme de connexion



c) Diagrammes de séquence

Le diagramme de séquence permet de représenter graphiquement les interactions entre les acteurs et le système selon un ordre chronologique. Chaque interaction est numérotée pour être séquencée.

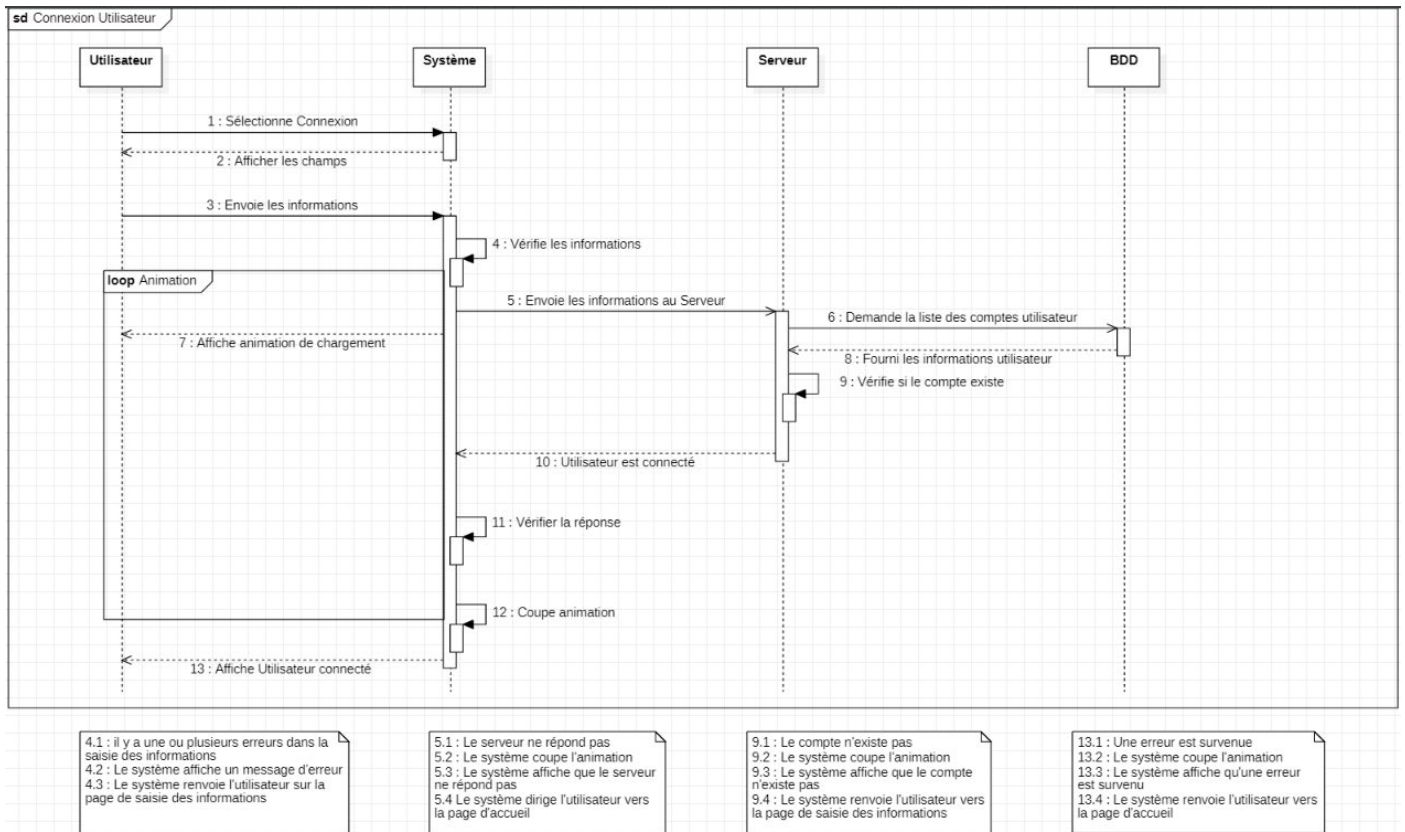
i) Diagramme de création d'un compte Utilisateur



Dans le diagramme de séquence ci-dessus, nous avons quatre acteurs : l'Utilisateur, le Système, le Serveur et la Base de données.

- (1). Le visiteur click sur la création de compte, ce qui envoie un message au Système.
- (2). Le système réceptionne le message, il envoie un **message de réponse** contenant le formulaire de création de compte.
- (3). L'Utilisateur envoie les informations demandées.
- (4). Le Système **déclenche une opération** de vérification des informations (**scenario alternatif 4**).
- (5). Le Système envoie un **message asynchrone** au Serveur pour vérifier si le compte Utilisateur existe déjà (**scenario alternatif 5**).
- (6). Le Serveur envoie un message asynchrone à la base de données pour demander la liste des comptes Utilisateurs.
- (7). Le système affiche une animation de chargement dans l'attente de la réponse du serveur et de la base de données.
- (8). La Base de données envoie en réponse au Serveur la liste des comptes Utilisateurs
- (9). Le serveur vérifie si le compte existe dans la liste (**scenario alternatif 9**).
- (10). Le serveur envoie un message asynchrone à la Base de données de création de compte Utilisateur.
- (11). La Base de données déclenche une opération d'exécution de la création du compte.
- (12). Le Serveur envoie la réponse vers le Système.
- (13). Le Système déclenche une opération de vérification de la réponse (**scénario alternatif 13**).
- (14). Le Système coupe l'animation de chargement.
- (15). Le système envoie la réponse à l'Utilisateur lui confirmant la création de son compte.

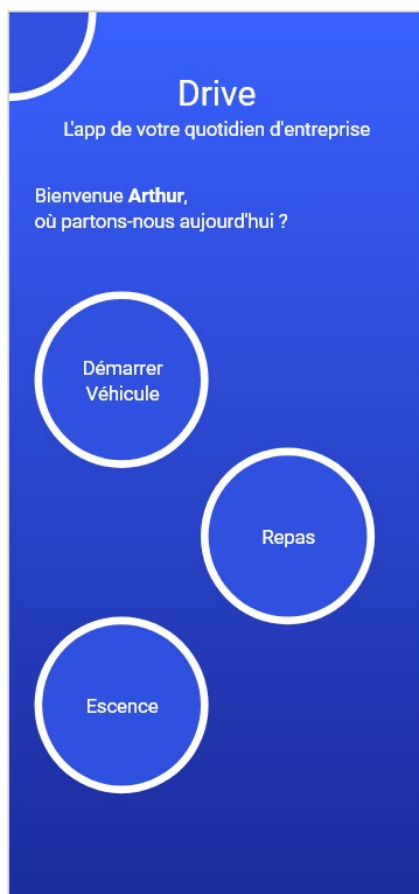
ii) Diagramme de connexion



d. Maquettage

La maquette de l'application m'a été fournie par l'entreprise.

Page d'accueil

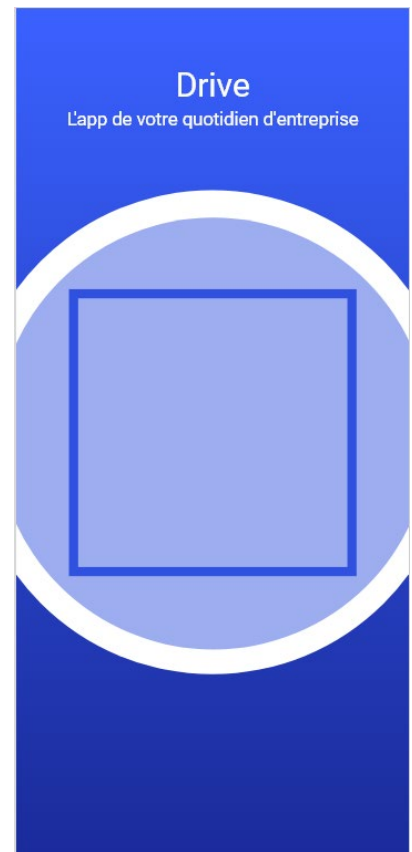
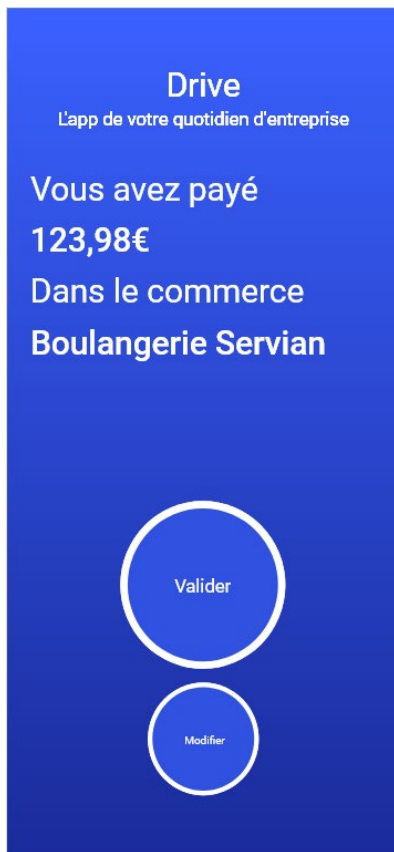


Page du véhicule

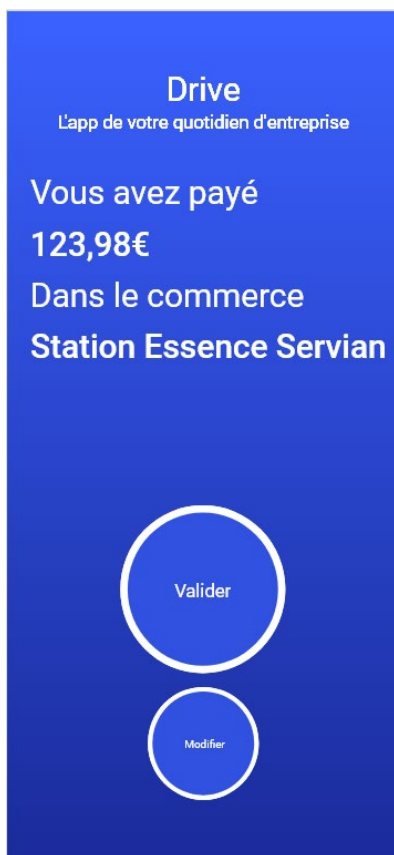


Page frais de repas

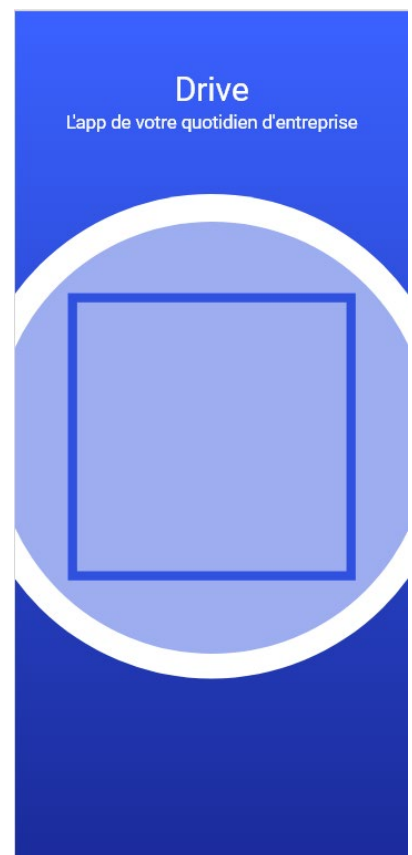
Page photo ticket repas



Page frais de carburant

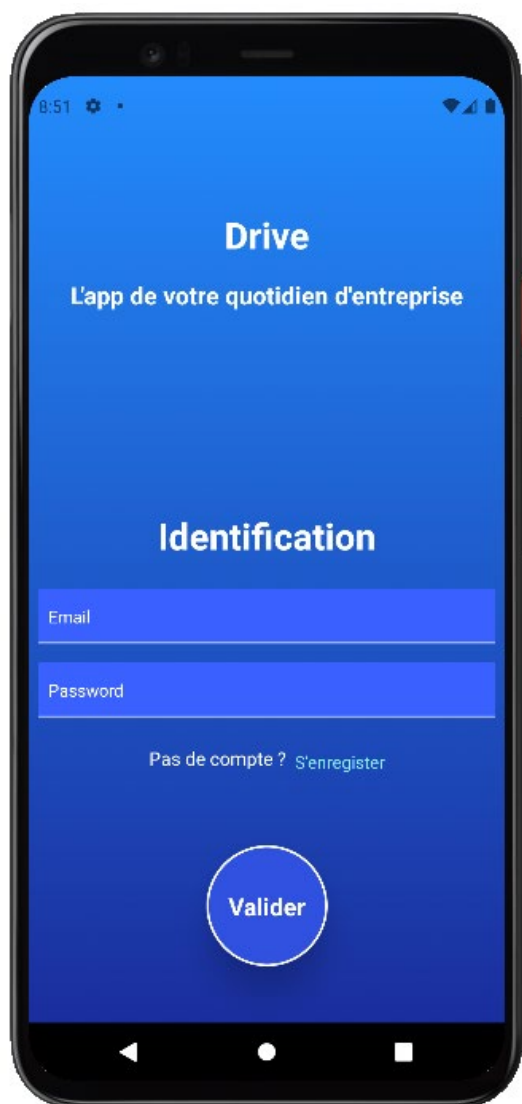


Page photo ticket carburant



La maquette n'étant pas complète, j'ai dû apporter quelques ajouts et quelques modifications pour améliorer l'expérience utilisateur (UI et UX) avec l'accord de l'employeur qui m'a laissé carte blanche. Nous pouvons voir sur les captures d'écran suivante certaines des pages manquantes ainsi que l'ajout d'une barre de navigation.

Page de connexion



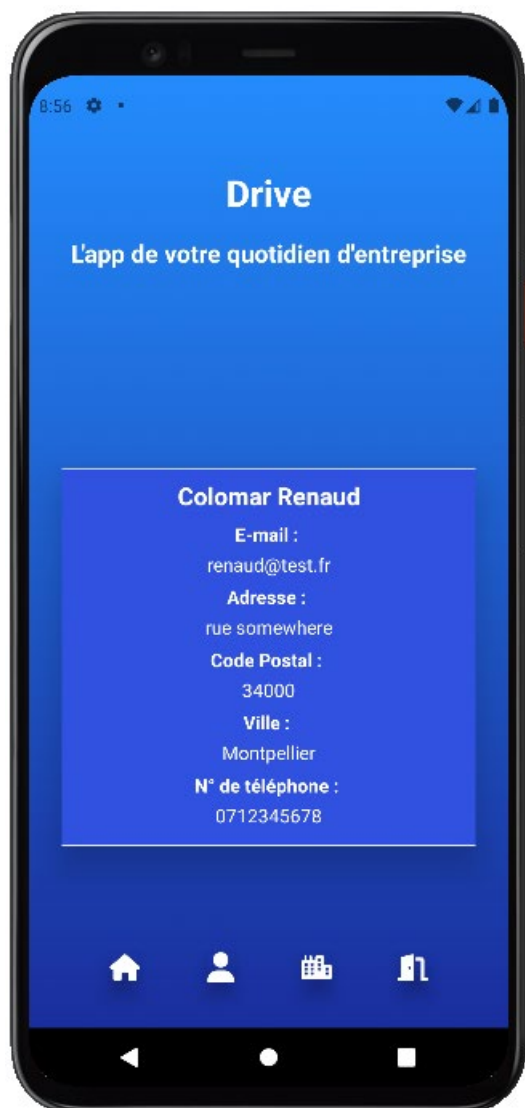
The login screen features a blue gradient background. At the top, the status bar shows the time 8:51, a gear icon, and signal/battery indicators. The app header includes the title 'Drive' and the subtitle 'L'app de votre quotidien d'entreprise'. The main heading is 'Identification'. Below it are two white input fields labeled 'Email' and 'Password'. A link 'Pas de compte ? S'enregister' is positioned below the password field. At the bottom is a large white circular button with the text 'Valider'. The Android navigation bar is visible at the very bottom.

Page de création de compte

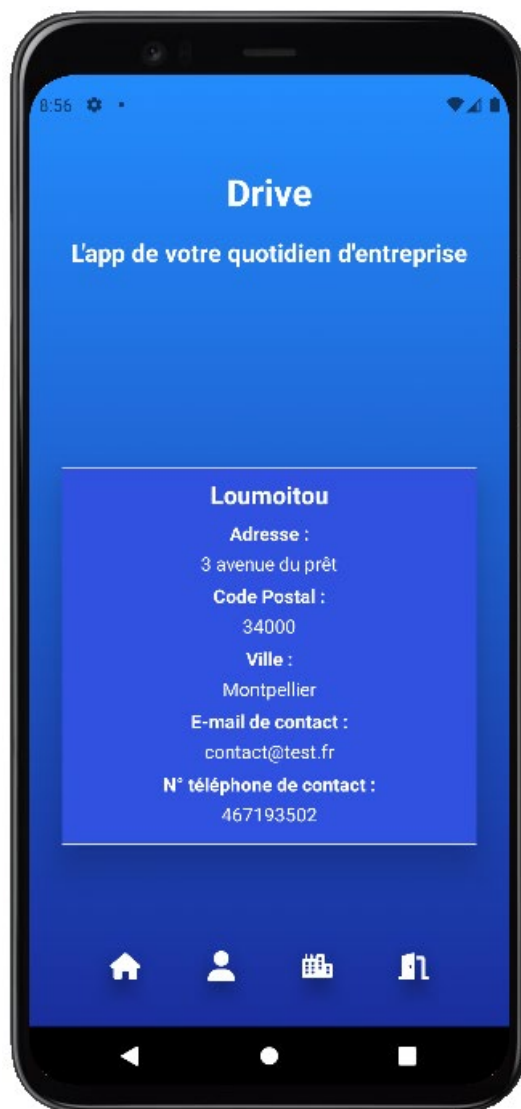


The registration screen has the same blue gradient background and header as the login screen. It contains a series of white input fields for 'Email', 'Password', 'Prénom', 'Nom', 'Adresse', 'Code Postal', 'Ville', and 'Téléphone'. Below these fields is a link 'Déjà un compte ? S'identifier'. At the bottom is a large white circular button with the text 'Valider'. The Android navigation bar is visible at the very bottom.

Ecran informations utilisateur



Ecran informations entreprise



7) Conception

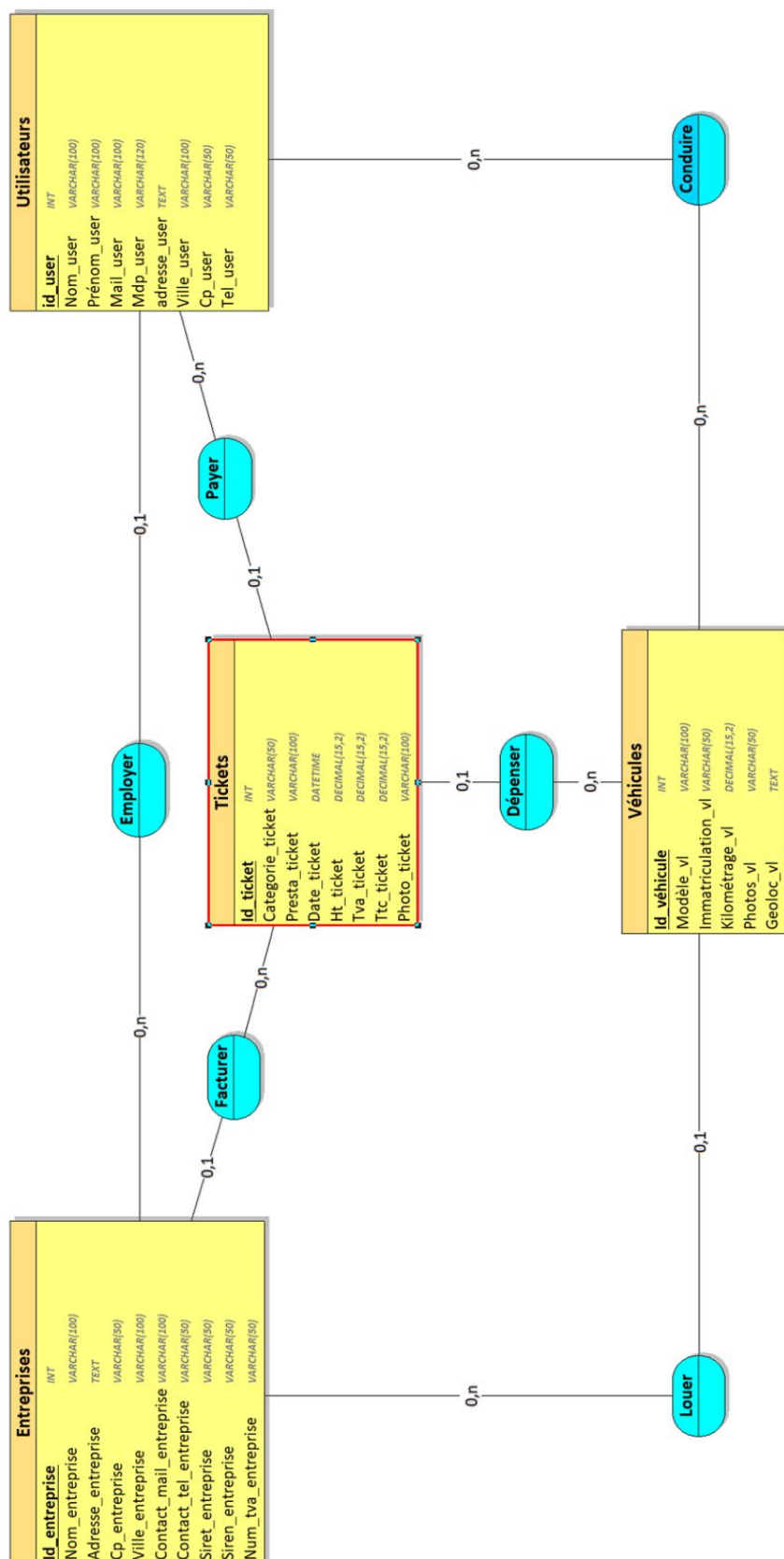
Le modèle conceptuel des données (MCD) est la représentation graphique et abstraite des données de manière structurée. Elle repose sur la notion d'**entité** et d'**association**.

Une **entité** est un élément unique composé d'**attributs**. L'un de ces **attributs** est l'identifiant (id) qui possède une valeur unique permettant d'identifier un enregistrement.

Une association est une interaction entre plusieurs entités. De part et d'autre d'une association il y a des cardinalités qui définissent le minimum et maximum de liens qui peuvent exister entre différentes entités.

J'ai réalisé mon projet en React Native et utilisé Firebase comme base de données. Firebase est un ensemble de services d'hébergement pour le développement d'applications. La plateforme est soutenue par Google, Google Cloud m'ayant été imposé par l'entreprise. Firebase est une base donnée NoSQL (non relationnelle), ce qui m'aurait permis de ne pas réaliser de MCD. Cependant, dans le cadre du titre, j'ai tenu à réaliser les différentes tables de l'application.

a) Modèle conceptuel des données (MCD)



Dans mon MCD j'ai conçu quatre tables et une table associative. Voici les relations entre elles :

○ Entreprises -> Employer -> Utilisateurs :

Une entreprise emploie zéro ou plusieurs utilisateurs (0,n),
chaque utilisateur n'est employé que par une seule entreprise (0,1).

○ Utilisateurs -> Payer -> Tickets :

Un utilisateur paie zéro ou plusieurs tickets (0,n),
un ticket ne peut être payé que par un utilisateur (0,1).

○ Tickets -> Facturer -> Entreprises :

Zéro ou plusieurs tickets sont facturés à une entreprise (0,n),
un ticket ne peut être facturé qu'à une seule entreprise (0,1).

○ Véhicule -> Dépenser -> Tickets :

Zéro ou plusieurs tickets peuvent être dépensés pour un véhicule (0,n),
un ticket ne peut être dépensé que par un véhicule (0,1).

○ Utilisateurs -> Conduire <- Véhicules :

Zéro ou plusieurs utilisateurs peuvent conduire un Véhicule (0,n),
zéro ou plusieurs Véhicules peuvent être conduits par un utilisateur (0,n).

Ici, Conduire devient une table associative dont la clé primaire est composée des clés étrangères référençant Utilisateurs et Véhicules.

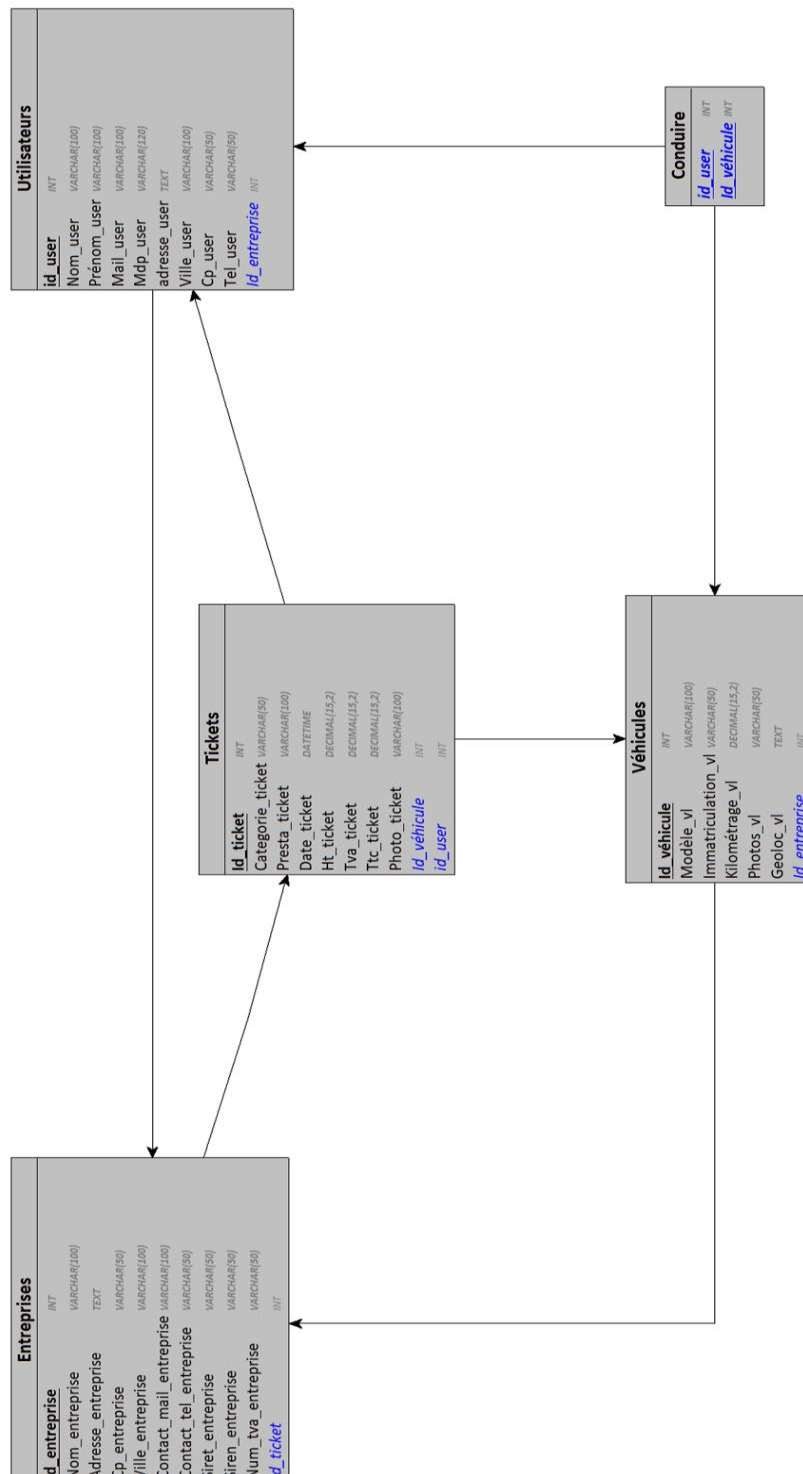
○ Entreprises -> Louer -> Véhicules :

Une Entreprise peut louer zéro ou plusieurs Véhicules (0,n),
Un véhicule ne peut être loué que par une entreprise (0,1).

b) Modèle Logique des Données (MLD)

Le MLD vient préciser le MCD et nous rapproche au plus près du modèle physique. Chaque table possède un identifiant unique que l'on appelle clé primaire (Primary Key) et les associations sont traduites par les clés étrangères (Foreign Keys).

Dans table associative Conduire la clé primaire est composée des clés étrangères référençant Utilisateurs et Véhicules.

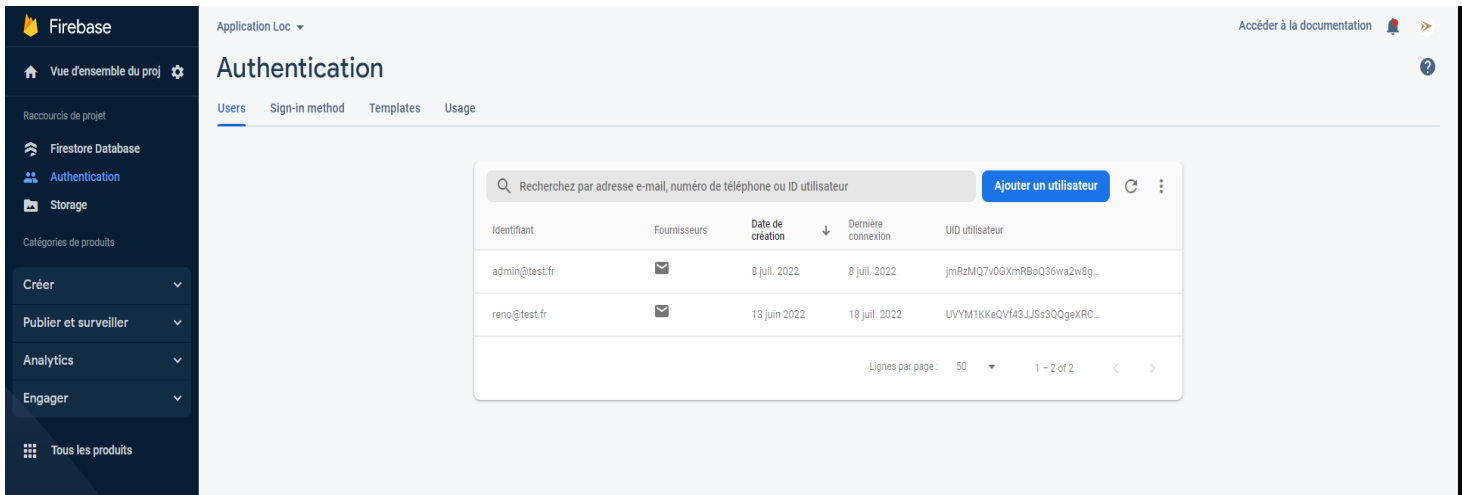


c) Création de la base de données

Firebase est composé de différents services :

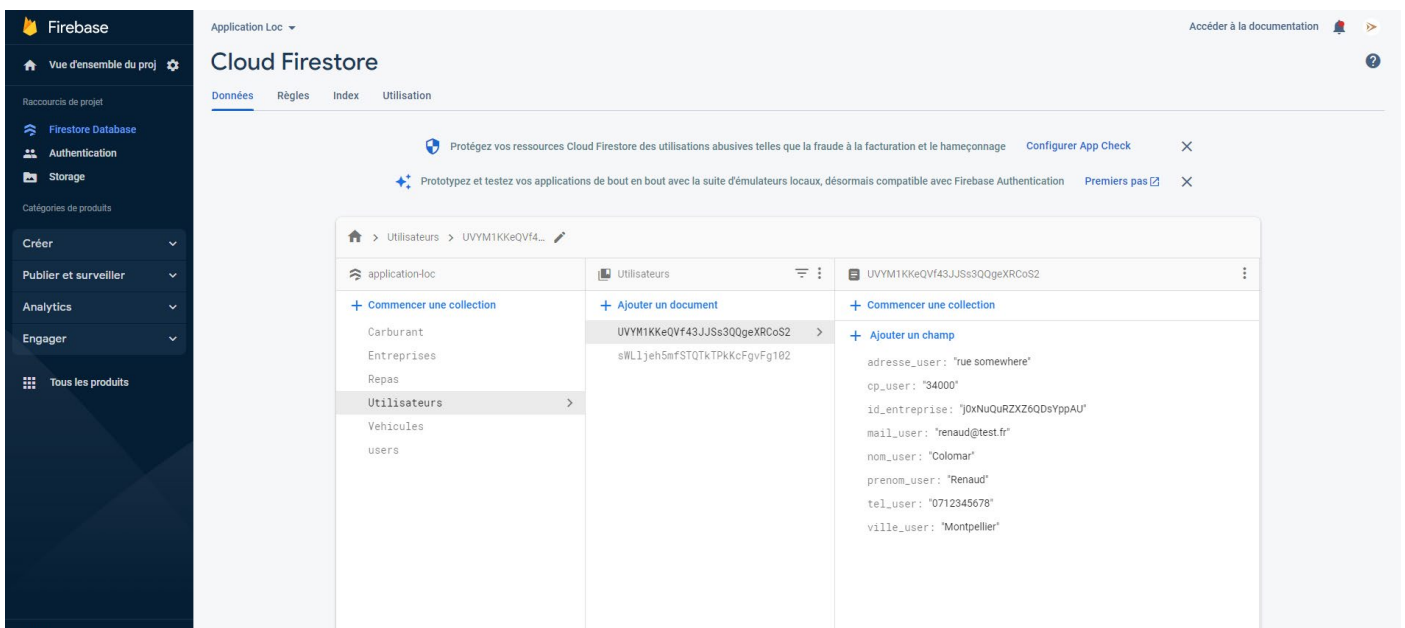
- Authentication :

Firebase Authentication vise à faciliter la création de systèmes d'authentification sécurisés. Nous l'utiliseront pour protéger l'accès à notre application.



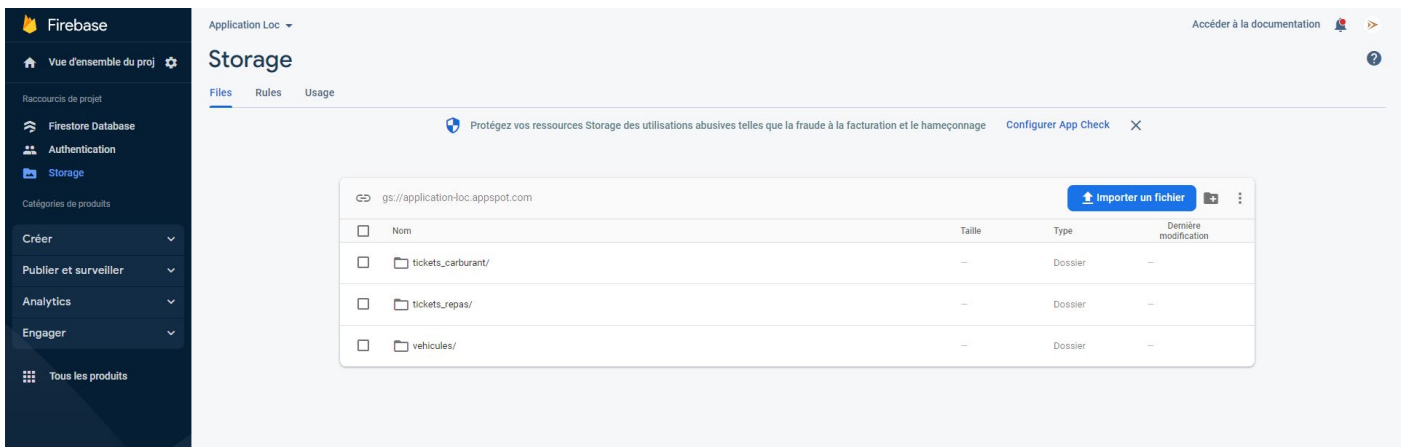
- Firestore Database :

Cloud Firestore est une base de données de documents NoSQL qui vous permet de stocker, de synchroniser et d'interroger facilement les données de vos applications mobiles et Web.



- Cloud Storage :

Cloud Storage pour Firebase est conçu pour les développeurs d'applications qui ont besoin de stocker et de diffuser du contenu généré par les utilisateurs, comme des photos ou des vidéos.



Mais si j'avais dû faire mon code SQL pour créer les tables de ma base de données, j'aurais fait :

- La création des tables en premier lieu

CREATE TABLE entreprises(

```
id_entreprise int PRIMARY KEY AUTO_INCREMENT NOT null,  
nom_entreprise varchar(100) NOT null,  
adresse_entreprise text NOT null,  
cp_entreprise varchar(50) NOT null,  
ville_entreprise varchar(100) NOT null,  
contact_mail_entreprise varchar(100) NOT null,  
contact_tel_entreprise varchar(100) NOT null,  
siret_entreprise varchar(50) NOT null,  
siren_entreprise varchar(50) NOT null,  
num_tva_entreprise varchar(50) NOT null,  
id_ticket int NOT null
```

);

CREATE TABLE utilisateurs(

```
id_user int PRIMARY KEY AUTO_INCREMENT NOT null,
```

```
nom_user varchar(100) NOT null,  
prenom_user varchar(100) NOT null,  
mail_user varchar(100) NOT null,  
mdp_user varchar(255) NOT null,  
adresse_user text NOT null,  
ville_user varchar(100) NOT null,  
cp_user varchar(50) NOT null,  
tel_user varchar(50) NOT null,  
id_entreprise int NOT null  
);
```

```
CREATE TABLE tickets(  
    id_ticket int PRIMARY KEY AUTO_INCREMENT NOT null,  
    categorie_ticket varchar(100) NOT null,  
    presta_ticket varchar(100) NOT null,  
    date_ticket datetime NOT null,  
    ht_ticket currency NOT null,  
    tva_ticket decimal(15,2) NOT null,  
    ttc_ticket currency NOT null,  
    photo_ticket varchar(100) NOT null,  
    id_vehicule int NOT null,  
    id_user int NOT null  
);
```

```
CREATE TABLE vehicules(  
    id_vehicule int PRIMARY KEY AUTO_INCREMENT NOT null,  
    modele_vl varchar(100) NOT null,  
    immatriculation_vl varchar(50) NOT null,  
    kilometrage_vl decimal(15,2) NOT null,  
    photo_vl varchar(50) NOT null,  
    geoloc_vl text NOT null,  
    id_entreprise int NOT null  
);
```

```
CREATE TABLE conduire(  
    id_user int NOT null,  
    id_vehicule int NOT null,  
    CONSTRAINT id_conduire PRIMARY KEY (id_user, id_vehicule),
```

);

- Puis l'ajout des clés étrangères dans chaque table

```
ALTER TABLE entreprises ADD CONSTRAINT fk_tickets  
    FOREIGN KEY (id_ticket)  
    REFERENCES tickets(id_ticket);
```

```
ALTER TABLE utilisateurs ADD CONSTRAINT fk_entreprises  
    FOREIGN KEY (id_entreprise)  
    REFERENCES entreprises(id_entreprise);
```

```
ALTER TABLE tickets ADD CONSTRAINT fk_vehicules  
    FOREIGN KEY (id_vehicule)  
    REFERENCES vehicules(id_vehicule);
```

```
ALTER TABLE tickets ADD CONSTRAINT fk_utilisateurs  
    FOREIGN KEY (id_user)  
    REFERENCES utilisateurs(id_user);
```

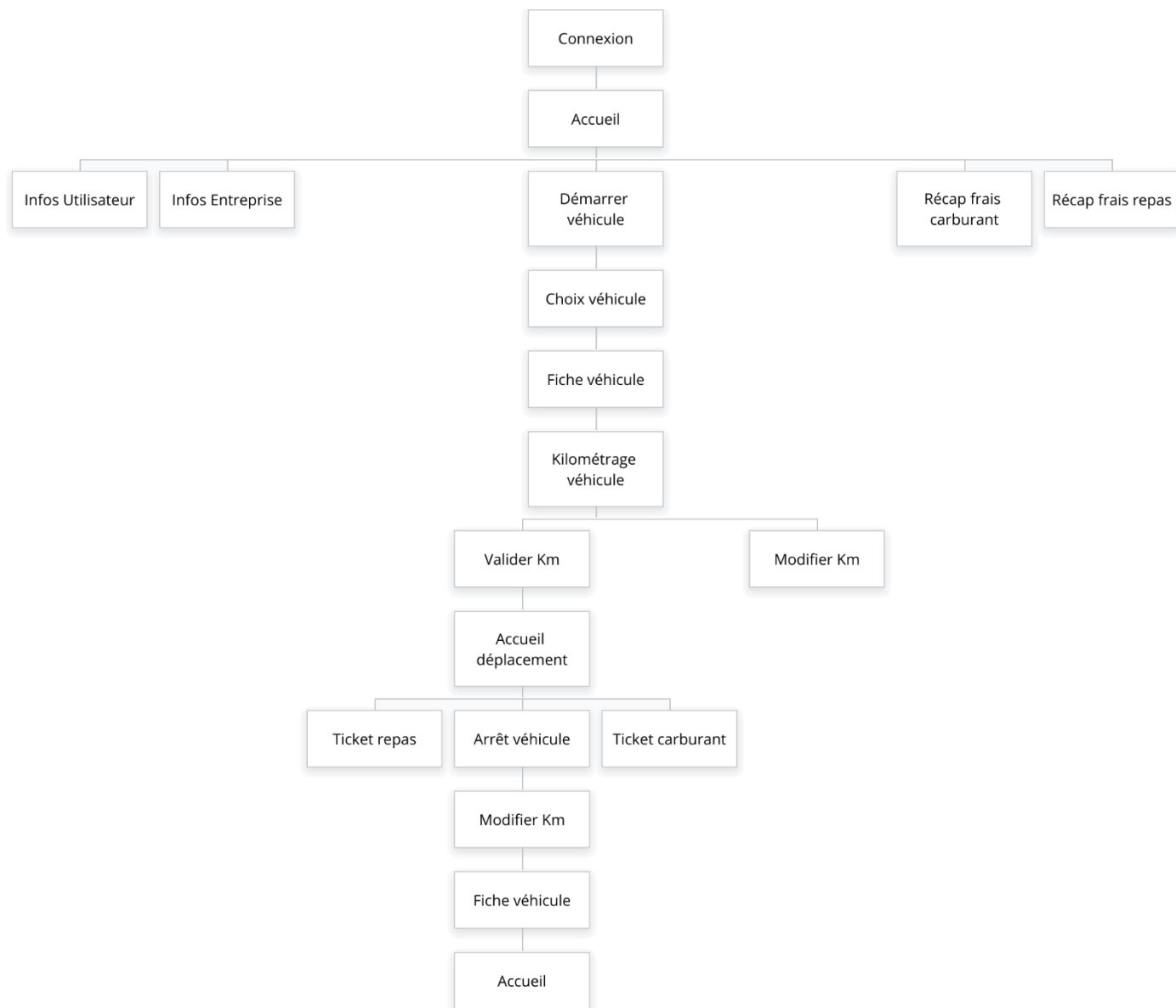
```
ALTER TABLE vehicules ADD CONSTRAINT fk_entreprises  
    FOREIGN KEY (id_entreprise)  
    REFERENCES entreprises(id_entreprise);
```

```
ALTER TABLE conduire ADD CONSTRAINT fk_utilisateurs  
    FOREIGN KEY (id_user)  
    REFERENCES utilisateurs(id_user);
```

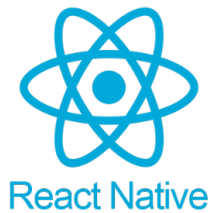
```
ALTER TABLE conduire ADD CONSTRAINT fk_vehicules  
    FOREIGN KEY (id_vehicule)  
    REFERENCES vehicules(id_vehicule);
```

8) Arborescence

Le projet étant toujours en cours de développement, certaines pages figurant dans cette arborescence ne font pas partie de la version présentée.



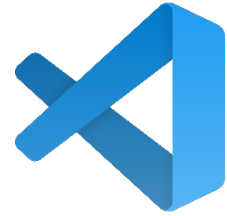
9) Outils techniques utilisés



React Native est un framework d'applications mobiles open source créé par Facebook. Il est utilisé pour développer des applications pour Android, iOS et UWP (Universal Windows Platform) en permettant aux développeurs d'utiliser React avec les fonctionnalités natives de ces plateformes.



JSX (JavaScript Syntax Extension et parfois appelé JavaScript XML) est une extension React de la syntaxe du langage JavaScript qui permet de structurer le rendu des composants à l'aide d'une syntaxe similaire en apparence au HTML.



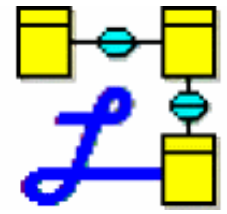
Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et MacOS.



Adobe XD est un outil de conception d'expérience utilisateur pour les applications web et les applications mobiles, développé et publié par Adobe Inc.



Android Studio est un environnement de développement pour développer des applications Android développé par Google. Il intègre un émulateur permettant de faire tourner un système Android virtuel sur ordinateur.



Looping est un logiciel de modélisation conceptuelle de données qui vous permet d'organiser et de structurer vos données en diagrammes et représentations graphiques ordonnées.



StarUML est un logiciel de modélisation UML.



Firebase est un ensemble de services d'hébergement pour n'importe quel type d'application. Il propose d'héberger en NoSQL et en temps réel.

10) Fonctionnalités

Je vais vous présenter trois fonctionnalités de l'application :

- La connexion d'un Utilisateur
- L'inscription d'un Utilisateur
- La consultation d'un compte Utilisateur

a) Architecture

- L'application

	<ul style="list-style-type: none"> - <code>.expo</code> : librairies Expo - <code>.expo-shared</code> : setting Expo - <code>.vscode</code> : setting vscode - <code>Assets</code> <ul style="list-style-type: none"> ▪ <code>images</code> : images de l'application - <code>components</code> : composants intégrés dans les pages <ul style="list-style-type: none"> ▪ <code>Connect</code> : composants fonctionnels définissant l'accès des pages de l'application - <code>node_modules</code> : modules node - <code>screens</code> : screens représentant le template des pages et les données à afficher - <code>styles</code> : styles en JavaScript utilisés dans l'application - <code>utils</code> : configuration de la connexion à la base de données - <code>.env.local</code> - <code>.gitignore</code> - <code>App.js</code> : script principal d'importation des composants et dépendances - <code>App.json</code> - <code>babel.config.js</code> - <code>package-lock.json</code> : dépendance npm - <code>package.json</code>
---	---

- Les pages (screens)

L'architecture du code d'une page en React Native est spécifique à ce framework :


```

1  import React, {useState} from 'react';
2  import { View, Text, ImageBackground, TouchableOpacity } from 'react-native';
3  import { onAuthStateChanged } from 'firebase/auth';
4  import { auth } from '../utils/firebase-config';
5
6  import AppStyles from '../styles/AppStyles';
7  import InlineTextButton from '../components/InlineTextButton';
8
9  import Header from '../components/Header';
10 import NavBar from '../components/NavBar';
11
12 const Architecture = (props) => {
13   const background = require('../assets/images/gradient.png');
14
15   // Vérifier si l'utilisateur est connecté
16   const [user, setUser] = useState(null);
17
18   onAuthStateChanged(auth, (currentUser) => {
19     setUser(currentUser);
20   });
21   return (
22     <ImageBackground source={background} style={AppStyles.container}>
23       <Header />
24       <View style={AppStyles.content}>
25         <View>
26           <Text style={AppStyles.titleH3}>Bienvenue {user}</Text>
27         </View>
28         <View style={styles.screen}>
29           <TouchableOpacity style={styles.roundButtonL} onPress={() => navigation.navigate("VehiculeScreen")}>
30             <Text style={styles.textButton}>Démarrer</Text>
31           </TouchableOpacity>
32         </View>
33       </View>
34       <NavBar />
35     </ImageBackground>
36   );
37 };
38
39 export default Architecture;
40

```

Je commence par importer les différentes bibliothèques ou modules nécessaires à mon code (**lignes 1 à 4**).

Puis j'importe les différents **components** que je vais intégrer à ma page (**lignes 6 à 10**).

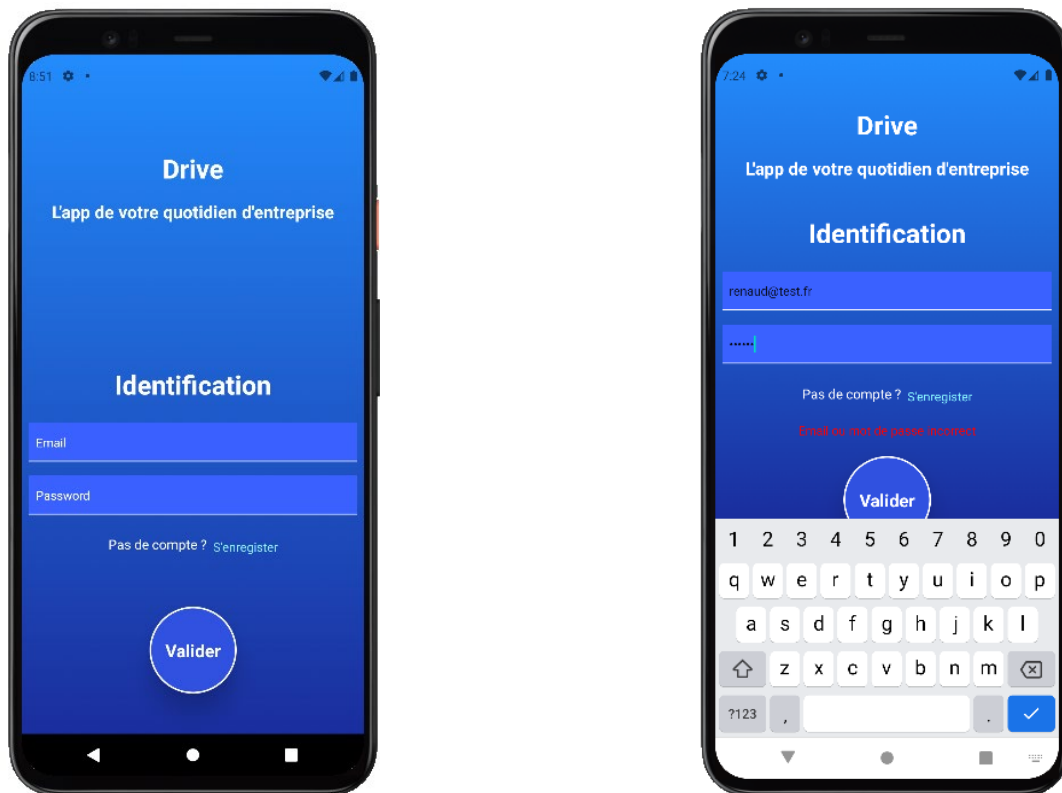
Viens ensuite le « corps » de notre page, ici la constante **Architecture**, qui commencera **ligne 12** et se terminera **ligne 37**.

Cette constante comportera deux sections :

- Entre **ligne 13** et **ligne 20**, je déclare la logique (les différentes variables et fonctions) de notre page.
- Entre **ligne 21** et **ligne 36**, le **return()** dans le quel nous retrouvons la partie **Jsx**, donc le rendu de notre page.

Ligne 39 nous **exportons** la constante **Architecture**, ce qui nous permettra, par exemple, de l'importer dans notre App.js et ainsi afficher la page.

b) Connexion d'un Utilisateur



Le formulaire de connexion est la première page qui s’affiche à l’ouverture de **Drive**. Tout Utilisateur doit avoir un compte afin de se connecter et avoir accès à l’application.

Une fois les informations renseignées, l’Utilisateur clique sur le bouton « Valider » et sera dirigé vers la page d’accueil.

S’il y a une ou plusieurs erreurs dans la saisie des informations, un message d’erreur s’affichera en rouge à l’écran. Ce message ne précisera pas quelle entrée est incorrecte (sécurité).

Si l’Utilisateur n’a pas de compte, il pourra s’inscrire en cliquant sur « s’enregistrer » et sera dirigé vers le formulaire de la création de compte.

Figure A

```

1 import React, {useState} from 'react';
2 import { View, Text, ImageBackground, TextInput, TouchableOpacity } from 'react-native';
3 import { signInWithEmailAndPassword } from 'firebase/auth';
4 import { auth } from '../utils/firebase-config';
5

```

Figure B

```

5
6 import AppStyles from '../styles/AppStyles';
7 import InlineTextButton from '../components/InlineTextButton';
8
9 import Header from '../components/Header';
10

```

Figure C

```

14 const background = require('../assets/images/gradient.png');
15
16 const [email, setEmail] = useState('');
17 const [password, setPassword] = useState('');
18 const [error, setError] = useState(false);
19
20 // Variable demandant la connexion de l'utilisateur à Firebase
21 const handleLogin = (e) => {
22   signInWithEmailAndPassword(
23     auth,
24     email,
25     password
26   ).then((e) => {
27     console.log('success');
28     const user = e.user;
29     console.log(user);
30   })
31   .catch (error => {
32     setError(true);
33     console.log(error.message);
34   });
35 }
36

```

Figure D

```

38
39 <ImageBackground source={background} style={AppStyles.container}>
40   <Header />
41   <View style={AppStyles.content}>
42     <Text style={AppStyles.titleH1}>Identification</Text>
43     <TextInput
44       style={AppStyles.textInput}
45       placeholder='Email'
46       placeholderTextColor='white'
47       onChangeText={({text}) => setEmail(text)} />
48     <TextInput
49       style={AppStyles.textInput}
50       placeholder='Password'
51       placeholderTextColor='white'
52       secureTextEntry={true}
53       onChangeText={({text}) => setPassword(text)} />
54     <View style={AppStyles.rowContainer}>
55       <Text style={AppStyles.text}>Pas de compte ? </Text>
56       <InlineTextButton text="S'enregister" onPress={() => navigation.navigate("SignUp")} />
57     </View>
58     <Text style={{color:'red'}}>{error && 'Email ou mot de passe incorrect'}</Text>
59
60     <TouchableOpacity style={AppStyles.validButton} onPress={handleLogin}>
61       <Text style={{color:'white', fontSize:20, fontWeight:'bold'}}>Valider</Text>
62     </TouchableOpacity>
63
64   </View>
65 </ImageBackground>
66
67 );
68 }
69

```

Dans la construction de notre page, nous commençons par intégrer le background dont l'image se trouve dans notre dossier 'images' (figure C, ligne 14).

Puis nous affichons le Header que nous importons des **components** (figure B, ligne 9).

Vient le formulaire avec le `<TextInput />` de l'Email et celui du mot de passe. **onChangeText** (figure D, lignes 47 et 53) va permettre à setEmail et setPassword de prendre les valeurs du texte renseigné par l'Utilisateur et de donner ces valeurs au State (figure C, lignes 16 et 17).

Figure D, ligne 52 **secureTextEntry** va cacher la saisie du mot de passe.

Figure D, ligne 56 j'affiche le bouton « S'enregistrer », component importé figure B, ligne 7. **onPress** dirige l'Utilisateur vers le page 'SignUp'.

Dans le cas d'une erreur dans la saisie des informations, un message d'erreur s'affichera sous le formulaire (figure D, ligne 59). Par défaut, son état est défini comme **false** (figure C, ligne 18).

Figure D, des lignes 61 à 63 `<TouchableOpacity />` va définir un bouton « Valider » pressable qui, au clique, appliquera la fonction **handleLogin** (figure D, ligne 61).

Cette fonction va demander la connexion de l'Utilisateur à la base de données par **Firebase Authentication** (figure C, ligne 21).

Pour ce faire, j'utilise la méthode **signInWithEmailAndPassword** (figure C, ligne 22) que j'importe de la bibliothèque **firebase/auth** (figure A, ligne 3). Dans cette méthode je renseigne **auth** qui est la constante que j'importe de **firebase-config** (figure A, ligne 4) et qui autorise l'accès à **Firebase Authentication**. Puis je renseigne **email** qui reprend la valeur du state définie par **setEmail** (figure C, ligne 16), et enfin **password** qui reprend la valeur du state définie par **setPassword** (figure C, ligne 17).

.then (ligne 26, figure C) : l'authentification est validée, je donne à **user** les données de l'utilisateur renseignées dans la base de données.

.catch (ligne 31, figure C) : il y a une erreur lors de l'authentification, **setError** donne la valeur **true** au state (figure C, ligne 18). Le message d'erreur (ligne 59 de la figure D) s'affichera.

Une fois connecté, l'Utilisateur est automatiquement dirigé sur la page d'accueil de l'application.

c) Inscription d'un Utilisateur

The image displays two smartphone screens side-by-side, both showing the 'Drive' app interface. The top of each screen has a blue header with the word 'Drive' and the subtitle 'L'app de votre quotidien d'entreprise'. Below this is a registration form with the following fields: Email, Password, Prénom, Nom, Adresse, Code Postal, Ville, and Téléphone. On the left screen, all fields are empty. On the right screen, the fields are filled with example data: Email is 'renaud.frn', Password is masked with '*****', and the other fields are empty. At the bottom of each screen is a circular button labeled 'Valider'. Above this button is a link that reads 'Déjà un compte ? S'identifier'. The background of the app is a solid blue color.

Comme je l'ai dit précédemment, tout Utilisateur doit avoir un compte afin de se connecter et avoir accès à l'application. Si tel n'est pas le cas, l'utilisateur accèdera au formulaire d'inscription en cliquant sur le bouton « S'enregistrer » de la page de connexion.

Une fois les informations renseignées, l'Utilisateur clique sur le bouton « Valider » et sera dirigé vers la page d'accueil.

S'il y a une ou plusieurs erreurs dans la saisie des informations, un message d'erreur s'affichera en rouge à l'écran.

Si l'Utilisateur a déjà un compte, il pourra se connecter en cliquant sur « S'identifier » et sera dirigé vers le formulaire de connexion.

Figure A

```

1  import React, {useRef, useState} from 'react';
2  import { View, Text, ImageBackground, TextInput, TouchableOpacity, StyleSheet } from 'react-native';
3  import { auth, db } from '../utils/firebase-config';
4  import { createUserWithEmailAndPassword } from 'firebase/auth';
5

```

Figure B

```

6  import AppStyles from '../styles/AppStyles';
7  import InlineTextButton from '../components/InlineTextButton';
8
9  import Header from '../components/Header';
10

```

Figure C

```

13  const background = require('../assets/images/gradient.png');
14
15  const [email, setEmail] = useState('');
16  const [password, setPassword] = useState('');
17  const [error, setError] = useState(false);
18
19  const [prenom, setPrenom] = useState('');
20  const [nom, setNom] = useState('');
21  const [adresse, setAdresse] = useState('');
22  const [codePostal, setCodePostal] = useState('');
23  const [ville, setVille] = useState('');
24  const [telephone, setTelephone] = useState('');
25
26  const handleRegister = async (e) => {
27    // Création de l'utilisateur dans Authentication de Firebase
28    createUserWithEmailAndPassword(
29      auth,
30      email,
31      password
32    )
33      .then((userCredential) => {
34        console.log('success');
35        const user = userCredential.user;
36        console.log(user);
37      })
38      .catch (error => {
39        setError(true);
40        console.log(error.message);
41      });
42    // Création de l'utilisateur dans la base de données Firestore
43    addDoc(collection(db, 'Utilisateurs'), {
44      prenom_user: prenom,
45      nom_user: nom,
46      mail_user: email,
47      adresse_user: adresse,
48      codePostal_user: codePostal,
49      ville_user: ville,
50      telephone_user: telephone})
51      .then(() => {
52        console.log('Document successfully written!');
53      })
54      .catch((error) => {
55        (() => navigation.navigate("Login"))
56        console.error('Error writing document: ', error);
57      });
58  }

```

Figure D

```

61 <ImageBackground source={background} style={AppStyles.container}>
62
63 <Header />
64 <View style={AppStyles.content}>
65 <TextInput
66   style={styles.input}
67   placeholder='Email'
68   placeholderTextColor='white'
69   onChangeText={({text}) => setEmail(text)} />
70 <TextInput
71   style={styles.input}
72   placeholder='Password'
73   placeholderTextColor='white'
74   secureTextEntry={true}
75   onChangeText={({text}) => setPassword(text)} />
76 <TextInput value={prenom} onChangeText={({prenom}) => setPrenom(prenom)} placeholder="Prénom" style={styles.input} placeholderTextColor='white' />
77 <TextInput value={nom} onChangeText={({nom}) => setNom(nom)} placeholder="Nom" style={styles.input} placeholderTextColor='white' />
78 <TextInput value={adresse} onChangeText={({adresse}) => setAdresse(adresse)} placeholder="Adresse" style={styles.input} placeholderTextColor='white' />
79 <TextInput value={codePostal} onChangeText={({codePostal}) => setCodePostal(codePostal)} placeholder="Code Postal" style={styles.input} placeholderTextColor='white' />
80 <TextInput value={ville} onChangeText={({ville}) => setVille(ville)} placeholder="Ville" style={styles.input} placeholderTextColor='white' />
81 <TextInput value={telephone} onChangeText={({telephone}) => setTelephone(telephone)} placeholder="Téléphone" style={styles.input} placeholderTextColor='white' />
82 <View style={AppStyles.rowContainer}>
83   <Text style={AppStyles.text}>Déjà un compte ? </Text>
84   <InlineTextButton text="S'identifier" onPress={() => navigation.navigate("Login")} />
85 </View>
86 <Text style={{color:'red'}}>{error && 'Erreur de saisie'}</Text>
87 <TouchableOpacity style={styles.button} onPress={handleRegister}>
88   <Text style={{color:'white', fontSize:15, fontWeight:'bold'}}>Valider</Text>
89 </TouchableOpacity>
90 </View>
91 <View>
92
93 </View>
94 </ImageBackground>

```

Figure E

```

101 const styles = StyleSheet.create({
102   input: {
103     alignSelf: 'stretch',
104     padding: 2,
105     backgroundColor: '#3A61FF',
106     borderBottomColor: 'white',
107     borderBottomWidth: 1,
108     marginVertical: 2,
109   },
110   button: {
111     width: 80,
112     height: 80,
113     // marginTop: 2,
114     justifyContent: 'center',
115     alignItems: 'center',
116     color: 'white',
117     borderWidth: 2,
118     borderColor: 'white',
119     borderRadius: 100,
120     backgroundColor: '#3051E0',
121     elevation: 20,
122     shadowColor: 'black',
123     shadowOffset: { width: 0, height: 2 },
124     shadowOpacity: 0.8,
125     shadowRadius: 2,
126   }
127 })

```

Figure D, des lignes 15 à 24, je créer les constantes qui prendrons les valeurs de chaque `<TextInput />` du formulaire (**figure D**, des lignes 65 à 81) pour les enregistrer dans le **State**.

Figure D, ligne 74 `secureTextEntry` va cacher la saisie du mot de passe.

Figure D, ligne 84 j'affiche le bouton « S'enregistrer », component importé **figure B, ligne 7**. `onPress` dirige l'Utilisateur vers le page 'Login'.

Dans le cas d'une erreur dans la saisie des informations, un message d'erreur s'affichera sous le formulaire (**figure D, ligne 86**). Par défaut, son état est défini comme `false` (**figure C, ligne 17**).

Figure D, des lignes 87 à 89 `<TouchableOpacity />` va définir un bouton « Valider » pressable qui, au clique, appliquera la fonction `handleRegister` (**figure D, ligne 87**).

Cette fonction va demander l'enregistrement de l'Utilisateur à la base de données par **Firebase Authentication** (**figure C, ligne 28**) et dans la base de données **Firestore** (**figure C, ligne 43**) :

- Pour l'enregistrement de l'utilisateur dans **Firebase Authentication**, j'utilise la méthode `createUserWithEmailAndPassword` (**figure C, ligne 28**) que j'importe de la bibliothèque `firebase/auth` (**figure A, ligne 4**). Dans cette méthode je renseigne `auth` qui est la constante que j'importe de `firebase-config` (**figure A, ligne 3**) et qui autorise l'accès à **Firebase Authentication**. Puis je renseigne `email` qui reprend la valeur du state définie par `setEmail` (**figure C, ligne 15**), et enfin `password` qui reprend la valeur du state définie par `setPassword` (**figure C, ligne 16**).

`.then` (**ligne 33, figure C**) : l'authentification est validée, je donne à `user` les données de l'utilisateur renseignées dans la base de données.

`.catch` (**ligne 31, figure C**) : il y a une erreur lors de l'authentification, `setError` donne la valeur `true` au state (**figure C, ligne 18**). Le message d'erreur (**ligne 59 de la figure D**) s'affichera.

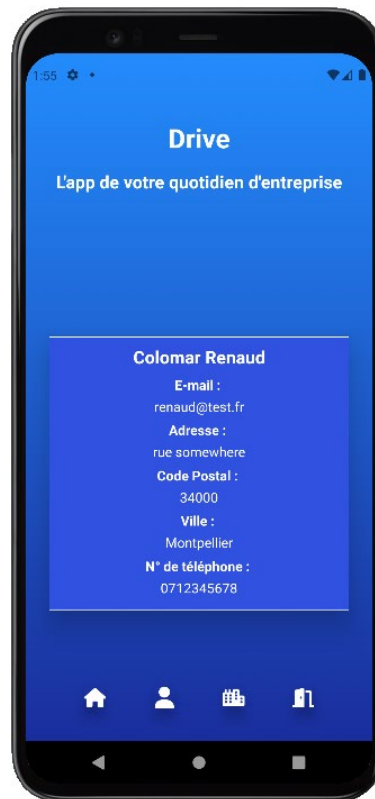
- Pour l'enregistrement de l'utilisateur dans **Firestore**, j'utilise `addDoc` (**figure C, ligne 43**). J'indique que je souhaite enregistrer les données dans la collection (que l'on peut comparer à une table en SQL) `'Utilisateurs'`. Je renseigne `db` qui est la constante que j'importe de `firebase-config` (**figure A, ligne 3**) et qui autorise l'accès à **Firestore**. Puis je renseigne les `attributs` avec les valeurs du `State` correspondantes (**figure C, lignes 44 à 50**).

`.then` (**figure C, ligne 51**) : L'inscription est validée, message dans le console.

`.catch` (**figure C, ligne 54**) : L'utilisateur est redirigé vers la page de connexion.

Une fois enregistré, l'Utilisateur est connecté et automatiquement dirigé sur la page d'accueil de l'application.

d) Consultation d'un compte Utilisateur



Une fois connecté, l'utilisateur a accès à ses informations via la barre de navigation en bas d'écran.

Figure A

```
1 import React, {useState} from 'react';
2 import {auth, db} from '../utils/firebase-config';
3 import {onAuthStateChanged} from "firebase/auth";
4 import { View, Text, ImageBackground } from 'react-native';
5 import { doc, getDoc } from 'firebase/firestore';
6
```

Figure B

```

7 import AppStyles from '../styles/AppStyles';
8
9 import Header from '../components/Header';
10 import NavBar from '../components/NavBar';
11

```

Figure C

```

15
16 // Vérifier si l'utilisateur est connecté
17 const [user, setUser] = useState(null);
18 onAuthStateChanged(auth, (currentUser) => {
19   setUser(currentUser);
20 });
21
22
23
24 const [userName, setUserName] = useState(null);
25 const [userPrenom, setUserPrenom] = useState(null);
26 const [userEmail, setUserEmail] = useState(null);
27 const [userTel, setUserTel] = useState(null);
28 const [userAdresse, setUserAdresse] = useState(null);
29 const [userVille, setUserVille] = useState(null);
30 const [userCodePostal, setUserCodePostal] = useState(null);
31
32 // Récupère l'Id de l'utilisateur connecté
33 const userId = user?.uid;
34
35 // Récupère l'Id de l'utilisateur dans la bdd Firestore
36 const userDoc = doc(db, `Utilisateurs/${userId}`);
37
38 // On récupère les données de l'utilisateur sur Firestore
39 getDoc(userDoc).then(docData => {
40   if (docData.exists()) {
41
42     const user = docData.data();
43     setUserName(user.nom_user);
44     setUserPrenom(user.prenom_user);
45     setUserEmail(user.mail_user);
46     setUserTel(user.tel_user);
47     setUserAdresse(user.adresse_user);
48     setUserVille(user.ville_user);
49     setUserCodePostal(user.cp_user);
50
51   } else {
52     console.log("No such document!");
53   }
54 }).catch(error => {
55   console.log("Error getting document:", error);
56 });

```

Figure D

```

64 <Header />
65 <View style={AppStyles.content}>
66   <View style={AppStyles.liste}>
67     <Text style={AppStyles.titleH2}>{userName} {userPrenom}</Text>
68     <Text style={AppStyles.titleH3}>E-mail : </Text>
69     <Text style={AppStyles.text}>{userEmail}</Text>
70     <Text style={AppStyles.titleH3}>Adresse : </Text>
71     <Text style={AppStyles.text}>{userAdresse}</Text>
72     <Text style={AppStyles.titleH3}>Code Postal : </Text>
73     <Text style={AppStyles.text}>{userCodePostal}</Text>
74     <Text style={AppStyles.titleH3}>Ville : </Text>
75     <Text style={AppStyles.text}>{userVille}</Text>
76     <Text style={AppStyles.titleH3}>N° de téléphone : </Text>
77     <Text style={AppStyles.text}>{userTel}</Text>
78   </View>
79 </View>
80 <NavBar />
81

```

Figure D, lignes 17 à 20 : avec la méthode `onAuthChanged` que j'importe de `firebase/auth` (figure A, ligne 3), je vérifie si l'utilisateur est connecté. **Auth** autorise l'accès à la base de donnée et `setUser` va me permettre d'enregistrer dans le **state** les données de l'utilisateur.

Des lignes 24 à 30 de la figure C, je défini les constantes qui me permettront d'enregistrer les données souhaitées de l'utilisateur récupérées dans **Firestore**, dans le **state**.

Ligne 33 de la figure C, je donne à la constante **userId** l'id de l'utilisateur si celui-ci existe dans **Firebase Authentication**.

La constante **userDoc** (**figure C, ligne 36**) sera le chemin d'accès aux données souhaitées : **db** m'autorise l'accès à la base de données **Firestore**, '**Utilisateurs**' est la collection (ou table en sql) dans laquelle se trouvent les données des utilisateurs, et je cible l'utilisateur connecté grâce à l'id de l'utilisateur « stocké » dans la constante **userId**.

Pour la suite, explication de la hiérarchie d'une base de données **Firestore** :

Base de données -> collections (tables) -> documents(id) -> datas(attributs)

Afin de récupérer les données de l'utilisateur, j'utilise la méthode **getDoc** de **firebase/firestore** (**figure C, ligne 39**) avec le chemin renseigné par notre constante **userDoc**.

.then (figure C, ligne 39) : une fois l'utilisateur trouvé dans la base de données (le **document** trouvé), j'utilise la méthode **docData** pour accéder aux datas.

If (figure C, ligne 40) : si des datas existent dans la base de données de l'utilisateur alors je les enregistre dans le **state** de **user** (**ligne 42**) et je récupère celles que je souhaite avec les différents **setUser** (**des lignes 43 à 49**).

S'il n'existe pas de datas pour l'utilisateur, un message d'erreur s'affichera dans la console (**figure C, ligne 52**).

S'il y a une erreur dans l'accès à la récupération des datas, un message d'erreur s'affichera dans la console (**figure C, ligne 55**).

Dans **la figure D, des lignes 63 à 74**, nous récupérons les informations stockées dans le **state** par les constantes des **lignes 43 à 49 de la figure C**. Je les affiche dans les balises **<Text />** en appelant les bonnes constantes entre **{}**.

11) Conclusion

De travailler sur ce projet ambitieux durant mes deux mois de stage a été l'occasion de me confronter aux réalités du métier de développeur web et web mobile. J'ai dû m'adapter à des contraintes qui m'ont été

imposées par l'employeur, travailler en totale autonomie, découvrir et apprendre un nouveau langage à développer, faire face à des difficultés et rester bloquer plusieurs jours sur un code. Mais à force de persévérances, de curiosité, de recherches et de travail, j'ai trouvé les solutions et avancé dans le développement de l'application avec une grande satisfaction à chaque étape franchie.

La formation que j'ai suivie à l'Adrar m'a enseigné les bases du métier de développeur. Mais les formateurs m'ont également appris à apprendre, à chercher et à progresser.

Le stage au sein de l'entreprise Instadrone m'a conforté dans mon choix, le métier de développeur répond à mes attentes professionnelles et personnelles.

J'ai hâte d'entamer cette nouvelle carrière au sein d'une entreprise afin de prendre de l'expérience aux côtés d'autres développeurs plus expérimentés, de mener à son terme le projet Drive pour l'entreprise Instadrone en tant qu'indépendant et par la suite me lancer dans de nouveaux défis en freelance.