# RUNEO DRIVE

Clément Sartoni - 2023


## Résumé

Projet de préparation au TPI consistant en la continuation du projet Runeo Drive, déjà débuté au P_Appro_1 et permettant aux chauffeurs du Paléo de recevoir les informations des courses à effectuer



Clément Sartoni

# 1. Analyse préliminaire

## *1.1.* Introduction

Ce projet a été initié par M. Carrel il y a 4 ans pour répondre aux besoins d'organisation des chauffeurs du Paléo. Il a été repris par la classe min-mid 4 de l'etml dans le cadre de la préparation au TPI, et les différentes tâches à réaliser ont été réparties entre les différents membres de la classe grâce à une organisation agile et une séparation en user stories.

L'application a déjà fait ses preuves et est déjà utilisée au paléo depuis plusieurs années. Elle est séparée en deux parties : l'application de bureau utilisée par les coordinateurs pour planifier les trajets, et l'application mobile utilisée par les chauffeurs pour recevoir les informations et prendre en charge les trajets.

Ce rapport détaillera l'analyse et la réalisation des tâches à la charge de Clément Sartoni, concernant toutes l'application mobile, Runeo-Drive. Chaque partie sera donc divisée pour parler de chaque tâche séparément.

## 1.2. Objectifs

Améliorations de l'application Runeo-Drive existante selon 3 axes :

### 1.2.1. Restreindre le choix de véhicule

Souvent (mais pas toujours), le type de véhicule est défini lors de la publication d'un run. Si c'est le cas, dans l'application actuelle, l'application propose de choisir un véhicule parmi tous les véhicules existants.

Il faut que ce choix soit restreint aux seuls véhicules du type fixé.

### 1.2.2. Adresse du backend

Il faut une liste déroulante qui permet de choisir avec le nom du festival et laisser une option "Autre..." qui permet d'introduire une valeur ('localhost:8000' par exemple).

### 1.2.3. Autres runs de mon artiste

Il arrive fréquemment qu'à la fin d'un run les personnes transportées posent des questions au chauffeur par rapport au retour à l'hôtel - par exemple - ou par rapport aux transports du lendemain.

Le chauffeur ne dispose pas (actuellement) d'un moyen efficace de répondre, il doit souvent dire à la personne de s'adresser au bureau des chauffeurs.

On veut donc qu'il puisse avoir un accès facile et rapide à l'information grâce à son app

## 1.2.4. Planification initiale

Le projet a débuté le lundi 20 mars et se terminera le vendredi 28 avril 2023.

Il y a deux semaines de vacances prévues du lundi 10 au vendredi 21 avril 2023, ainsi que deux jours d'absence supplémentaires : le lundi 27 mars, où je suis absent pour des raisons personnelles, et le vendredi 7 avril qui est un jour férié.

Dans une semaine de travail normale, le temps disponible pour travailler est organisé comme indiqué dans ce tableau :

| Jour | Lundi | Mardi | Mercredi | Jeudi | Vendredi | Total |
|---|---|---|---|---|---|---|
| **Périodes** | 8 | 0 | 9 | 4 | 9 | 30 |
| **Heures** | 6h | 0h | 6h45 | 3h | 6h45 | 22h30 |

En prenant donc en compte les vacances et les jours de congé, il y a à disposition pour ce projet 103 périodes ou 77 heures et 15 minutes.

## 1.2.5. Découpe en sprints

Vu le peu de temps à disposition pour ce projet, il a été choisi d'effectuer des sprints d'une semaine afin d'avoir un suivi plus pertinent et efficace. Cela a le désavantage de passer plus de temps sur des aspects organisationnels mais cela est nécessaire pour ne pas manquer de retours sur le travail qui est en cours, et si le travail de gestion des sprints est fait efficacement le coût se limite à quelques heures par semaine. Les sprints reviews se feront le jeudi à 15h00, et les sprints commenceront donc le vendredi matin de chaque semaine. Le dernier vendredi sera consacré à l'impression du rapport et aux dernières retouches qui seront potentiellement discutées lors de la dernière sprint review.

### *1.2.5.1. Sprint 1*

Les objectifs de ce sprint sont les suivants :

- L'analyse préliminaire est terminée.
- Les sprints sont définis sur IceScrum ainsi que dans la documentation, et les dates des sprint reviews sont définies.
- Le journal de travail reflète bien la réalité des activités et les bonnes habitudes sont instaurées pour rester durant les sprints suivants.
- Le product backlog contient suffisamment de stories pour pouvoir procéder au planning des sprints suivants. Ceci inclut pour chaque user story une description détaillée, les tests d'acceptance ainsi qu'une première ébauche des tâches à réaliser.

La sprint review se fera le jeudi 23 mars 2023 à 15h00.

#### *1.2.5.2. Sprint 2*

L'objectif de ce sprint sera principalement de réaliser la user story concernant l'implémentation du choix de l'adresse du backend lors de l'ouverture de l'application. Celle-ci a été choisie comme première tâche car c'est potentiellement la plus complexe à réaliser et que la fonctionnalité qu'elle apporte est essentielle.

Il faudra aussi prévoir de documenter la réalisation et les points de design spécifiques de cette user story.

La sprint review se fera le jeudi 30 mars 2023 à 15h00.

#### *1.2.5.3. Sprint 3*

L'objectif de ce sprint sera de réaliser la user story consistant à créer la fonctionnalité « Autres runs de mon artiste » et de commencer la dernière, ayant pour but de restreindre le choix du véhicule. Ces user stories sont un peu moins complexes car il est probable que certaines pages déjà existantes puissent être adaptées aux nouveaux besoins assez simplement. Tout comme pour la fonctionnalité précédente, il faudrait aussi avoir déjà documenté la réalisation ainsi que les points de design spécifiques.

La sprint review se fera le jeudi 6 avril 2023 à 15h00.

#### *1.2.5.4. Sprint 4*

Pour ce dernier sprint, l'objectif sera de finaliser la user story de restriction du choix du véhicule ainsi que de terminer la documentation.

La sprint review se fera le jeudi 27 avril 2023 à 15h00, afin d'avoir encore une journée de travail pour effectuer des retouches avant le TPI si nécessaire.

# 2. Analyse / Conception

## *2.1.* Analyse fonctionnelle

### 2.1.1. Autres runs de mon artiste

En tant que chauffeur, je veux accéder rapidement à la liste des runs de l'artiste que je transporte pour répondre à questions.

Tests d'acceptance :

| | |
|---|---|
| **Présence du bouton** | Quand on est sur la page d'un run, juste après les informations sur les runners, se trouve un bouton intitulé "Autres runs de l'artiste", comme sur la maquette de la page d'un run (Figure 1). |
| **Clic sur le bouton** | Sur la page d'un run, quand on clique sur le bouton "Autres runs de l'artistes", on est redirigé vers une page contenant tous les runs de l'artiste, comme sur la maquette "Autres runs de l'artiste" (Figure 2). |
| **Clic sur un run** | Sur la page "Autres runs de l'artiste", quand on clique sur un run, la page du run en question s'ouvre correctement et le bouton "autres runs de l'artiste" n'est plus visible. |
| **Retour depuis un run** | Sur la page du run ouvert après qu'on ait cliqué dessus sur la page "autres runs de l'artiste", quand on clique sur la petite flèche de retour en haut à droite, on retourne bien en arrière sur la page "autres runs de l'artiste". |

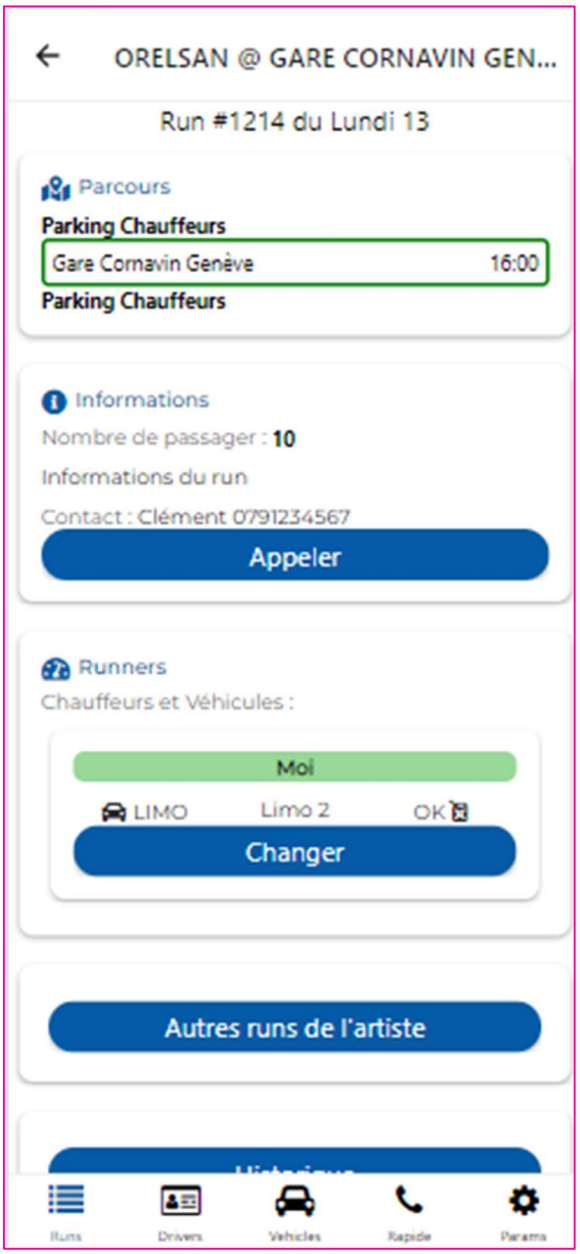


**Figure 1 : Maquette de la page d'un run**

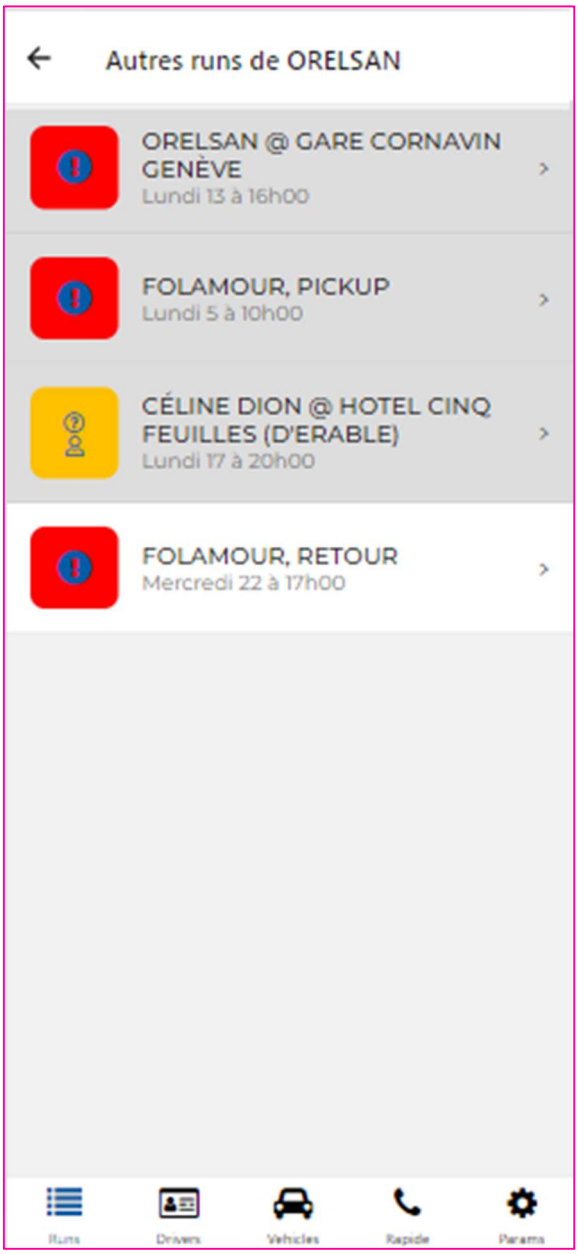


**Figure 2 : Maquette de la page "Autres runs de l'artiste"**

### 2.1.2. Adresse du backend

En tant que chauffeur, je veux pouvoir choisir à quel backend mon application se connecte pour pouvoir travailler à Belfort et à Paléo sans changer d'app.

Tests d'acceptance :

| | |
|---|---|
| **Présence de la dropdown** | Sur la première page de l'application permettant de se connecter, en dessus du champ permettant d'entrer le token, une liste déroulante est présente et accepte les valeurs "Paléo", "Belfort" ou "autre". Paléo est sélectionné par défaut |
| **Champ autre** | Sur la première page, quand la valeur "autre" est sélectionnée dans la liste déroulante, un champ texte apparait au-dessous et permet de rentrer un URL. |
| **Connexion à la bonne API** | Sur la première page, quand le festival ou l'URL et un token valide sont renseignés et que l'on clique sur le bouton "connexion", les runs du festival sélectionné apparaissent. |
| **Erreur réseau** | Sur la première page, alors que le smartphone ne peut pas atteindre le backend Quand on clique sur le bouton "connexion", une erreur avertit l'utilisateur qu'il y a un problème avec son réseau ou le site du festival sélectionné. |

### 2.1.3. Restreindre le choix de véhicule

En tant que chauffeur, je veux que l'app me propose de choisir un véhicule du type fixé pour me simplifier la manipulation et éviter des erreurs.

Test d'acceptance :

| | |
|---|---|
| **Sélection des véhicules** | Dans la page d'un run, quand le type de véhicule est fixé, lorsque je clique sur le bouton pour sélectionner le ou changer de véhicule, seuls les véhicules du type fixé apparaissent. |

## *2.2.* Concept

Le site web gère la base de données mysql et met à disposition une API pour que l'application mobile puisse récupérer les informations.

Pour mieux comprendre le contexte du travail présenté dans ce document, le fonctionnement de l'application sera détaillé ici au moyen de différents schémas déjà existants. Ils ont été créés lors des étapes antérieures du projet par d'autres personnes.

Le but premier de l'application est de gérer les runs, ou courses, qui sont concrètement les demandes ou besoins de transport des artistes ou des personnes les accompagnant. Ces runs sont planifiés par les coordinateurs soit avant le festival soit pendant, avec toutes les informations nécessaires comme les heures de rendez-vous et le type de véhicule par exemple. Ensuite, ils apparaissent sur la page principale de l'application mobile et les chauffeurs peuvent se désigner pour effectuer les runs tout en sélectionnant leur véhicule.

### 2.2.1. Diagrammes

À la page suivante se trouve le MCD de la base de données. Les deux tables les plus importantes sont les utilisateurs et les runs.

Le processus complet de gestion d'un run est détaillé en page 8 grâce à un schéma expliquant les différents états par lesquels un run doit passer.

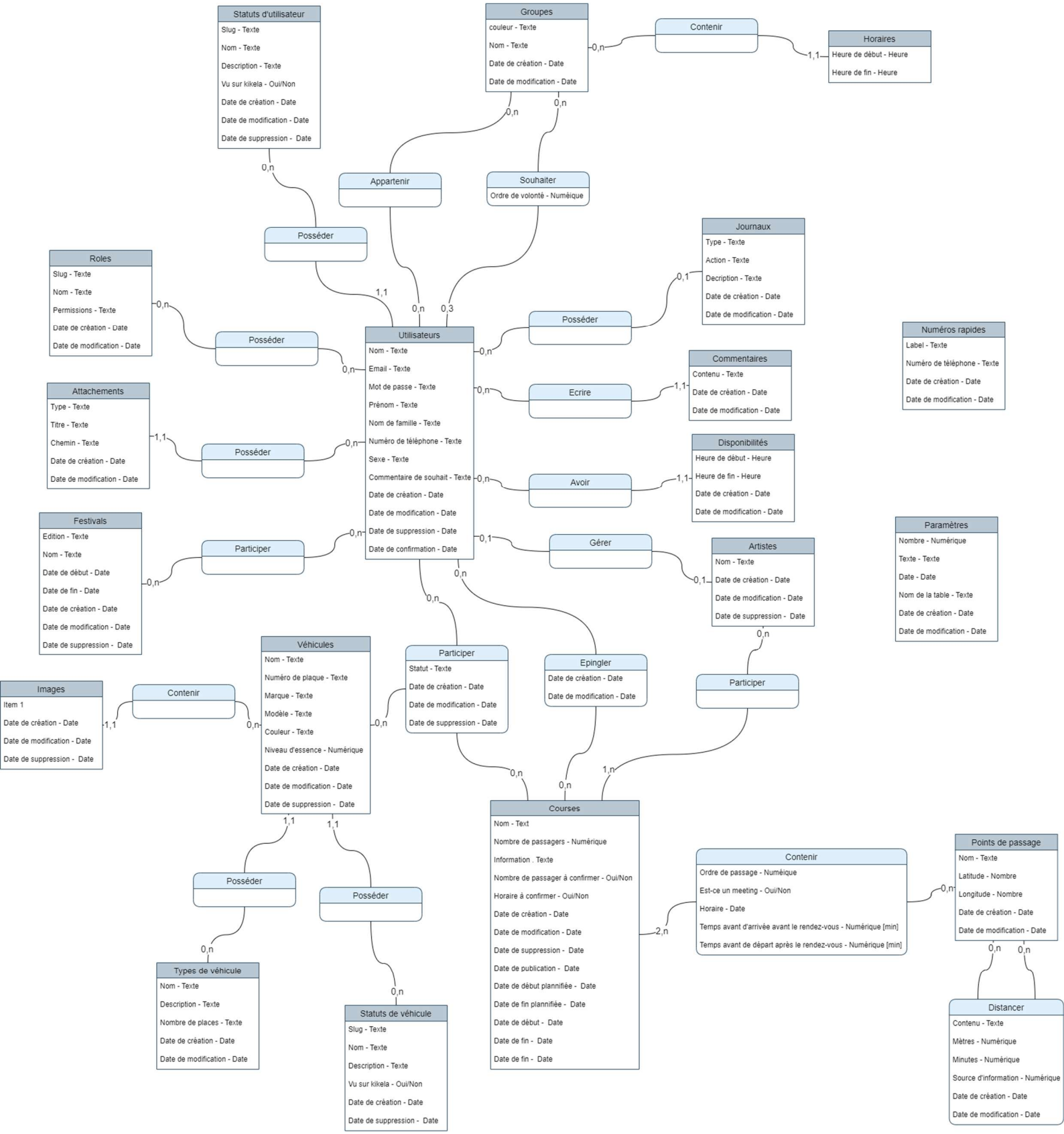


**Figure 3 : MCD de la base de données**

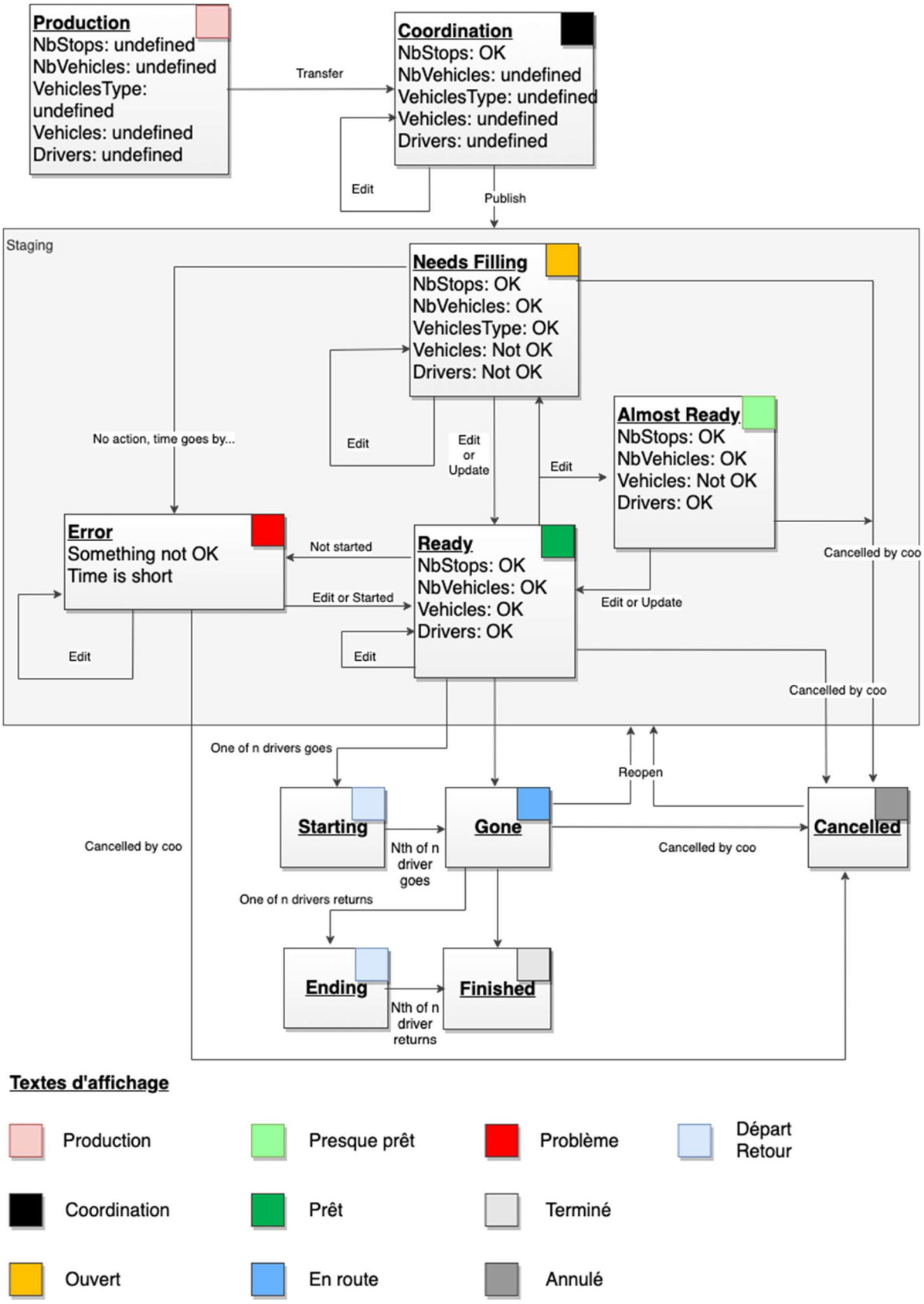


**Figure 4: Schéma détaillant les différents états par lesquels passe un run**

### 2.2.2. Fonctionnement de l'application mobile

Les chauffeurs téléchargent l'application depuis le site web ou se font envoyer un lien de téléchargement via WhatsApp. Ils reçoivent aussi de la part des coordinateurs un token de connexion leur permettant de s'identifier sur la première page de l'application et d'avoir accès aux runs.

## 2.3. Stratégie de test

En raison d'un problème concernant le build de l'application, il ne sera pas possible pour ce projet d'appliquer la procédure de test qui serait idéale et devrait normalement être utilisée. Ce chapitre sera donc séparé en deux.

### 2.3.1. Stratégie de test théorique

Il faudrait normalement installer complètement l'application sur un smartphone Android et sur un iPhone à chaque session de test pour vérifier que les modifications fonctionnent sur un environnement semblable à celui qui sera utilisé en production.

Pour ce faire, il est théoriquement possible de build l'application puis de la distribuer sur un canal de test sur le Play Store (Android) ainsi que sur l'App Store (iOS). Cependant, il est pour cela nécessaire d'avoir auparavant configuré des comptes permettant de publier des applications sur les deux systèmes. Dans l'état actuel, le compte développeur sur le Play Store est presque configuré et prêt à publier l'application, mais il n'y a rien de prévu pour iOS.

### 2.3.2. Stratégie de test utilisée

À chaque sprint review, M. Carrel effectuera un pull de la branche « develop » de GitHub sur son poste. Ensuite, l'application sera lancée en utilisant l'application « Expo Go » sur son smartphone. Les tests mentionnés dans l'analyse des user stories qui auront été terminées pourront ensuite être effectués.

Changer ainsi d'environnement permet de vérifier que les modifications effectuées dans l'environnement de développement fonctionnent une fois déployées ailleurs.

Concernant les données de test, tant que personne n'utilise la base de données de production de paléo, donc tant que les données de l'année passée n'ont pas été effacées, il est possible de créer des runs de tests depuis le site web afin d'utiliser le même backend qu'en production.

Il est possible que dans le futur la production ne soit plus disponible pour des tests, ce qui impliquerait de devoir lancer en plus de l'application mobile l'application Runeo-Desk localement et d'utiliser une base de données locale dans laquelle des runs de test pourront être créés.

## *2.4.* Risques techniques

Les risques techniques sont assez légers pour ce projet. Le principal est le manque de formation concernant le React et son fonctionnement dans une application en React Native, qui est le framework utilisé dans l'application mobile. Cependant, cela reste un langage connu et relativement maitrisé à la suite d'un projet précédent sur la même application.

Un autre risque potentiel est celui de reprendre du code déjà existant et codé par d'autres personnes. Cela rend plus complexe de prédire combien de temps la réalisation des fonctionnalités va prendre, étant donné que selon la manière dont l'application a été réalisée, il sera possible de plus ou moins réutiliser des éléments ou alors il pourrait ne pas être possible en l'état d'ajouter la fonctionnalité ce qui demanderait de remodeler des fonctionnalités déjà existantes.

## Choix techniques

Ces choix techniques ont été effectués lors de la création du projet et aucune modification n'a donc été appliquée à ces derniers.

| | |
|---|---|
| Outil de gestion des versions | Github |
| Framework du site web Runeo Desk | Laravel 9.16.0 |
| Framework JS de l'application mobile | React Native version 14.4 |
| Kit de développement logiciel de l'application mobile | Expo SDK version 43 |

# 2.5. Points de design spécifiques

## 2.5.1. Changement de l'adresse du backend

### *2.5.1.1. Champ URL caché*

Pour réaliser la fonctionnalité, utiliser un champ « dropdown » était nécessaire, et il devait accepter une valeur « Autre », qui serait donc entrée via un champ texte.

Plutôt que de gérer les deux champs dans le formulaire, il a été choisi de reporter les valeurs de la dropdown dans le champ texte à chaque changement de sélection. Grâce à une fonction appelée en même temps, le champ texte peut être caché lorsqu'il contient une valeur entrée depuis la dropdown, et affiché uniquement quand l'utilisateur doit lui-même renseigner son URL après avoir sélectionné « autre.

```
//#region dropdown config
const data = [
  { label: 'Paléo', value: 'http://runeo.paleo.ch/api' },
  { label: 'Belfort', value: 'http://belfort.festival.temp/api' },
  { label: 'Autre', value: '' }
];
const [selected, setSelected] = useState(data[0]);

const [urlVisible, setUrlVisible] = useState(false);

function onPress(item: { label: string; value: string }) : void {
    setUrlVisible(item.value == '')
}
//#endregion

const authContainer = AuthContainer.useContainer();
const initialValues = {
    token: "",
    url: selected.value
};
```

**Figure 5 : début de la déclaration des variables et fonctions nécessaires au fonctionnement du formulaire (fichier src\auth\TokenAuth.component.tsx)**

Ici, on peut observer peut voir la déclaration des variables « selected » et « urlVisible », qui utilisent la méthode « useState » de react, permettant de recharger tout ce qui dépend des variables à chaque fois qu'elles sont modifiées en utilisant leur fonction « set ».

Ce fonctionnement est utilisé pour les deux fonctionnalités requises du système : la variable « selected » permet de reporter les données sélectionnées dans la dropdown dans le formulaire en l'utilisant dans l'objet « initialValues » du formulaire, et la variable urlVisible définit l'affichage du champ URL en étant définie à chaque fois qu'une nouvelle valeur est sélectionnée (fonction « onPress »).

```jsx
<Formik
    onSubmit={onSubmit}
    validationSchema={
        yup.object().shape({
            token: yup.string().min(5).required(),
            url: yup.string().min(1).required()
        })}
    initialValues={initialValues}
    enableReinitialize={true}>
    {(formik) => (
        <View>
            <Text style={{fontFamily: 'Montserrat-ExtraBold', marginLeft: 10}}>FESTIVAL</Text>
            <Dropdown label={selected.label} data={data} onSelect={setSelected} onPress={onPress}/>

            <View style={urlVisible ? {} : {display: "none"}}>
                <Text style={{fontFamily: 'Montserrat-ExtraBold', marginLeft: 10}}>URL</Text>
                <TextInputComponent
                name={"url"}
                formik={formik}
                inputProps={{
                    placeholder: "Ex: https://192.168.241.121:8000/api"
                }}/>
            </View>
```

**Figure 6 : composants JSX du formulaire, de la dropdown et du champ URL (fichier src\auth\TokenAuth.component.tsx)**

On peut voir dans cette capture d'écran comment les variables influencent sur les composants.

- Le paramètre « enableReinitialize » renseigné dans le formulaire Formik permet de réinitialiser les valeurs à chaque fois que les « initialValues » sont modifiées.
- Les paramètres « onSelect » et « onPress » de la dropdown permettent à la dropdown, qui a été importée et adaptée depuis un projet tiers récupéré sur internet, de mettre à jour la valeur sélectionnée et de déclencher la fonction onPress.
- Le champ de l'URL et son label sont regroupés dans une View pour pouvoir gérer leur affichage avec un opérateur ternaire dépendant de la valeur « urlVisible ».


**Figure 7 : Rendu graphique de la fonctionnalité**

### *2.5.1.2. Données d'url*

Les données concernant les différentes options des festivals et des adresses qui y sont associées sont pour l'instant renseignées « en dur » en haut du fichier, comme elles l'étaient auparavant dans le fichier « App.tsx ». À l'endroit où étaient renseignées les adresses auparavant, un commentaire a été inséré pour indiquer le nouvel emplacement :

```
8
9  // Le paramétrage concernant les URLs par défaut utilisés par l'application a été déplacé dans le fichier suivant :
10 // './src/auth/TokenAuth.component.tsx'
11
```

**Figure 8 : commentaire à l'ancien emplacement de l'URL de l'API dans le fichier "App.tsx"**

Il suffit ensuite de modifier les informations du tableau « data » pour inclure de nouveaux festivals ou modifier leurs adresses (capture d'écran en page 15).

### *2.5.1.3. Gestion des erreurs*

Les tests d'acceptance spécifient le besoin de faire une distinction entre les erreurs d'accès à l'url spécifiée et entre celles liées à un token invalide. Auparavant, le code gérait déjà les exceptions mais n'exploitait pas les informations contenues dans les messages liés à celles-ci. Il a donc du fallu faire quelques tests pour analyser les informations à disposition puis modifier légèrement le fonctionnement pour gérer cette donnée.

```
try {
  Axios.defaults.baseURL = url;
  const user = await getAuthenticatedUserApi();
  await AsyncStorage.setItem(USER_STORAGE_KEY, JSON.stringify(user));
  setAuthenticatedUser(user);
} catch (e) {
  console.log(e);
  await logout();
  throw new Error(e.message);
}
```

**Figure 9 : gestion d'erreur lors de la tentative de récupération des données (fichier src\common\container\Auth.container.ts)**

Cette partie du fichier gère les erreurs pouvant provenir de la tentative de connexion à l'API :

- Il a fallu ajouter ici l'assignation de l'URL à Axios (le client de requêtes HTTP/HTTPS), car il essaie de s'y connecter directement après que l'url soit renseignée.
- Auparavant, le message de l'erreur n'était pas transmis lors de la création de la nouvelle erreur, qui est transmise à la fonction supérieure.

```
try {
    await authContainer.authenticate(values);
} catch (e) {
    if(e.message == "Network Error")
    {
        if(urlVisible)
        {
            setFieldError("url", "Erreur de connexion, vérifie ton accès à internet et l'URL que tu as entré.");
        }
        else
        {
            setFieldError("token", "Erreur de connexion, vérifie ton accès à internet.");
        }
    }
    else
    {
        setFieldError("token", "Erreur de token, vérifie que le token que tu as entré est bien valide pour le festival sélectionné.")
    }
    setSubmitting(false);
}
```

**Figure 10 : Transformation de l'erreur en information « user-friendly »**
**(fichier src\auth\TokenAuth.component.tsx)**

Ici, l'erreur envoyée depuis le fichier Auth.container.ts est récupérée et transformée en un message « user-friendly » (compréhensible par l'utilisateur), puis affichée en dessous des bons champs du formulaire.

Pour ce faire, le message de l'erreur a d'abord été analysé en affichant directement « e.message » dans le formulaire et en testant plusieurs scénarios d'erreur. Il a été remarqué que le message « Network Error » était retourné quand le site ne pouvait être atteint et que le message « 401- unauthorized » apparaissait quand uniquement le token était invalide. Il était ensuite facile d'utiliser ces informations pour configurer la condition et personnaliser le message.

En bonus, une distinction supplémentaire a été faite entre deux scénarios : si l'utilisateur a entré lui-même son URL, il est plus probable que le problème vienne de là, et donc il faut indiquer le message d'erreur sous le champ URL. Mais s'il a sélectionné un des festivals par défaut, il a plus probablement un problème avec sa connexion internet ou alors le site du festival sélectionné est down (moins probable). Étant donné que le champ URL est masqué il faut alors indiquer l'erreur sous le champ token comme normalement.

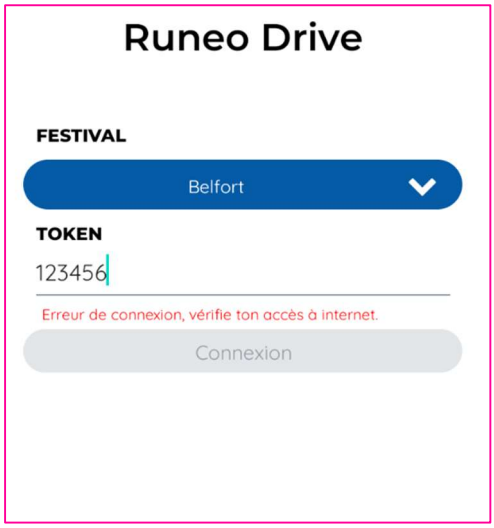


**Figure 12 : Pas d'accès internet**

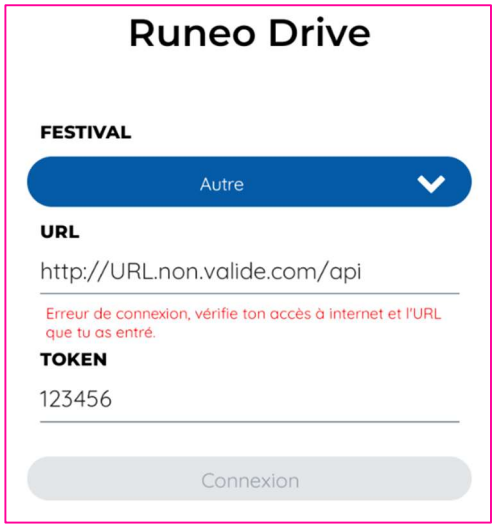


**Figure 13 : URL invalide**

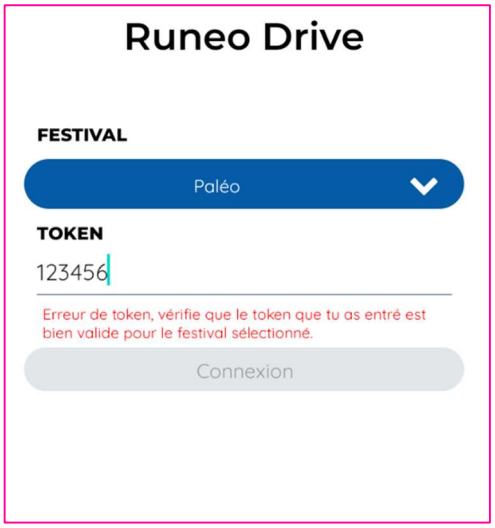


**Figure 11 : Token invalide**