Chibre Manager



Table des matières

[1 Analyse 4](#_Toc66442862)

[1.1 Introduction 4](#_Toc66442863)

[1.2 Organisation 5](#_Toc66442864)

[1.3 Cadre de travail 6](#_Toc66442865)

[1.3.1 Wavemind 6](#_Toc66442866)

[1.3.2 Ecole des Arche 6](#_Toc66442867)

[1.4 Durée du travail 6](#_Toc66442868)

[1.5 Planification initiale 6](#_Toc66442869)

[1.6 Choix des technologies 7](#_Toc66442870)

[1.6.1 Web 7](#_Toc66442871)

[**1.6.1.1** **Ruby on Rails** 7](#_Toc66442872)

[1.6.2 Mobile 7](#_Toc66442873)

[**1.6.2.1** **React Native** 7](#_Toc66442874)

[**1.6.2.2** **Expo** 7](#_Toc66442875)

[1.7 Stratégie de test 8](#_Toc66442876)

[**1.8** Environnement de travail 8](#_Toc66442877)

[1.8.1 Matériels Hardware. 8](#_Toc66442878)

[1.8.2 Logiciels et OS. 8](#_Toc66442879)

[**1.9** Objectif 9](#_Toc66442880)

[2 Conception 10](#_Toc66442881)

[***2.1*** Structure de du serveur web 10](#_Toc66442882)

[2.1.1 Ruby on Rails 10](#_Toc66442883)

[2.1.2 Base de données 11](#_Toc66442884)

[***2.2*** Structure de l’application mobile 12](#_Toc66442885)

[2.2.1 Expo 12](#_Toc66442886)

[2.2.2 React Native 12](#_Toc66442887)

[***2.3*** API 13](#_Toc66442888)

[***2.4*** Maquette graphique 14](#_Toc66442889)

[***2.5*** Schéma de navigation 15](#_Toc66442890)

[2.5.1 Mobile 15](#_Toc66442891)

[**2.6** Risques techniques 15](#_Toc66442892)

[***2.7*** Planification réelle 16](#_Toc66442893)

[3 Réalisation 17](#_Toc66442894)

[**3.1** Introduction 17](#_Toc66442895)

[**3.2** Ruby on Rails 17](#_Toc66442896)

[3.2.1 Configuration 17](#_Toc66442897)

[**3.2.1.1** **Serveur** 17](#_Toc66442898)

[3.2.1.2 Application 17](#_Toc66442899)

[**3.2.1.3** **Fonctionnalités** 17](#_Toc66442900)

[3.2.2 Mise en fonction des migrations et des modèles 17](#_Toc66442901)

[3.2.2.1 Les Migrations 17](#_Toc66442902)

[3.2.2.2 Les Modèles 18](#_Toc66442903)

[3.2.3 Les Routes 18](#_Toc66442904)

[3.2.3.1 API 18](#_Toc66442905)

[3.2.4 Les Contrôleurs 19](#_Toc66442906)

[3.2.5 Les Vues 19](#_Toc66442907)

[**3.3** React Native 20](#_Toc66442908)

[3.3.1 Configuration 20](#_Toc66442909)

[**3.3.1.1** **Application** 20](#_Toc66442910)

[**3.3.1.2** **Fonctionnalités** 20](#_Toc66442911)

[3.3.2 Les Vues 21](#_Toc66442912)

[**3.3.2.1** **Menu d’accueil** 21](#_Toc66442913)

[21](#_Toc66442914)

[**3.3.2.2** **Création des équipes et des joueurs** 22](#_Toc66442915)

[**3.3.2.3** **Qui commence et combien de points pour la partie ?** 23](#_Toc66442916)

[**3.3.2.4** **Partie** 23](#_Toc66442917)

[**3.3.2.5** **Lorsqu’une équipe gagne** 25](#_Toc66442918)

[**3.3.2.6** **Liste des parties crées** 26](#_Toc66442919)

[**3.3.2.7** **Option d’une partie** 27](#_Toc66442920)

[**3.3.2.8** **Atout** 28](#_Toc66442921)

[**3.3.2.9** **Les Annonces** 29](#_Toc66442922)

[**3.3.2.10** **Les Infos** 30](#_Toc66442923)

[**3.4** API 31](#_Toc66442924)

[3.4.1 Comment marche mon API 31](#_Toc66442925)

[**3.4.1.1** **Mobile** 31](#_Toc66442926)

[**3.4.1.2** **Serveur Web** 31](#_Toc66442927)

[***3.5*** Tests 33](#_Toc66442928)

[3.5.1 Erreurs restantes 33](#_Toc66442929)

[3.5.2 Application mobile 33](#_Toc66442930)

[**3.5.2.1** **Menu** 33](#_Toc66442931)

[**3.5.2.2** **Historique des parties** 33](#_Toc66442932)

[**3.5.2.3** **Option de la partie** 33](#_Toc66442933)

[**3.5.2.4** **Ecran de la partie** 34](#_Toc66442934)

[**3.5.2.5** **Création d’une équipe et des joueurs** 34](#_Toc66442935)

[**3.5.2.6** **Choix du joueur qui commence** 34](#_Toc66442936)

[**3.5.2.7** **Choix de l’atout** 34](#_Toc66442937)

[***3.6*** Liste des documents fournis 35](#_Toc66442938)

[4 Conclusions 35](#_Toc66442939)

[***4.1*** Conclusion personnelle 35](#_Toc66442940)

[***4.2*** Bilan des objectifs 35](#_Toc66442941)

[***4.3*** *Difficultés rencontrées* 36](#_Toc66442942)

[***4.4*** Suite du projet 36](#_Toc66442943)

[5 Annexes 37](#_Toc66442944)

[***5.1*** Résumé du rapport du TPI / version succincte de la documentation 37](#_Toc66442945)

[***5.2*** Sources – Bibliographie 37](#_Toc66442946)

[***5.3*** Journal de travail 37](#_Toc66442947)

[***5.4*** Manuel d'Installation 37](#_Toc66442948)

[***5.5*** Manuel d'Utilisation 37](#_Toc66442949)

[***5.6*** Archives du projet 37](#_Toc66442950)

Table des matières

[1 Analyse 6](#_Toc66443028)

[1.1 Introduction 6](#_Toc66443029)

[1.2 Organisation 7](#_Toc66443030)

[1.3 Cadre de travail 8](#_Toc66443031)

[1.3.1 Wavemind 8](#_Toc66443032)

[1.3.2 Ecole des Arche 8](#_Toc66443033)

[1.4 Durée du travail 8](#_Toc66443034)

[1.5 Planification initiale 8](#_Toc66443035)

[1.6 Choix des technologies 9](#_Toc66443036)

[1.6.1 Web 9](#_Toc66443037)

[1.6.2 Mobile 9](#_Toc66443038)

[1.7 Stratégie de test 10](#_Toc66443039)

[**1.8** Environnement de travail 10](#_Toc66443040)

[1.8.1 Matériels Hardware. 10](#_Toc66443041)

[1.8.2 Logiciels et OS. 10](#_Toc66443042)

[**1.9** Objectif 11](#_Toc66443043)

[2 Conception 12](#_Toc66443044)

[***2.1*** Structure de du serveur web 12](#_Toc66443045)

[2.1.1 Ruby on Rails 12](#_Toc66443046)

[2.1.2 Base de données 13](#_Toc66443047)

[***2.2*** Structure de l’application mobile 14](#_Toc66443048)

[2.2.1 Expo 14](#_Toc66443049)

[2.2.2 React Native 14](#_Toc66443050)

[***2.3*** API 15](#_Toc66443051)

[***2.4*** Maquette graphique 16](#_Toc66443052)

[***2.5*** Schéma de navigation 17](#_Toc66443053)

[2.5.1 Mobile 17](#_Toc66443054)

[**2.6** Risques techniques 17](#_Toc66443055)

[***2.7*** Planification réelle 18](#_Toc66443056)

[3 Réalisation 19](#_Toc66443057)

[**3.1** Introduction 19](#_Toc66443058)

[**3.2** Ruby on Rails 19](#_Toc66443059)

[3.2.1 Configuration 19](#_Toc66443060)

[3.2.2 Mise en fonction des migrations et des modèles 19](#_Toc66443061)

[3.2.3 Les Routes 20](#_Toc66443062)

[3.2.4 Les Contrôleurs 21](#_Toc66443063)

[3.2.5 Les Vues 21](#_Toc66443064)

[**3.3** React Native 22](#_Toc66443065)

[3.3.1 Configuration 22](#_Toc66443066)

[3.3.2 Les Vues 23](#_Toc66443067)

[23](#_Toc66443068)

[**3.4** API 33](#_Toc66443069)

[3.4.1 Comment marche mon API 33](#_Toc66443070)

[***3.5*** Tests 35](#_Toc66443071)

[3.5.1 Erreurs restantes 35](#_Toc66443072)

[3.5.2 Application mobile 35](#_Toc66443073)

[***3.6*** Liste des documents fournis 37](#_Toc66443074)

[4 Conclusions 37](#_Toc66443075)

[***4.1*** Conclusion personnelle 37](#_Toc66443076)

[***4.2*** Bilan des objectifs 37](#_Toc66443077)

[***4.3*** *Difficultés rencontrées* 38](#_Toc66443078)

[***4.4*** Suite du projet 38](#_Toc66443079)

[5 Annexes 39](#_Toc66443080)

[***5.1*** Résumé du rapport du TPI / version succincte de la documentation 39](#_Toc66443081)

[***5.2*** Sources – Bibliographie 39](#_Toc66443082)

[***5.3*** Journal de travail 39](#_Toc66443083)

[***5.4*** Manuel d'Installation 39](#_Toc66443084)

[***5.5*** Manuel d'Utilisation 39](#_Toc66443085)

[***5.6*** Archives du projet 39](#_Toc66443086)

Analyse

## Introduction

Dans le cadre de ma formation CFC en tant qu’informaticien, l’École des Arche, dans lequel j’ai étudié pendant 2 ans, nous impose un travail de fin d’année dans le but de confirmer mes études et ainsi obtenir mon CFC. Ce projet se fait dans le cadre de l’entreprise (Wavemind) dans laquelle je suis stagiaire depuis le début de l’année 2019. J’ai à ma disposition 80 heures ou 2 semaines de travail pour réaliser ce projet qui s'est déroulé du 1er mars au 15 mars 2021.

Arrivant à la fin de mon stage, mon responsable, Alain Fresco grand joueur de cartes et surtout de chibre (déviante du jass), m’a proposé un projet à réaliser et à présenter à mon école en rapport avec ce jeu.

Le projet de mon TPI consiste à créer une application mobile qui va aider l’utilisateur à compter les points de deux équipes durant une partie de chibre bel et bien réel. Le comptage des pointes au chibre n’est pas si facile et des erreurs de comptage peuvent vite arriver.

Ce projet m’aidera surement à mieux approfondir mes connaissances Ruby, en Javascript, les API et la relation entre le backend et le frontend. C’est un projet complet qui comporte toutes les techniques auquel j’ai pu étudier durant mon stage.

Pour pouvoir mener a bien ce travail, l’on m’a conseillé de suivre la méthode des 6 pas. Cela consiste à planifier ma façon de traiter certaines tâches, cette méthode consiste à suivre 6 étapes.

1. S’informer
2. Planifier
3. Décider
4. Réaliser
5. Contrôler
6. Evaluer

Bien sûr-il ce peut que certaine étape ne soit pas nécessaire pour aller au bien de mon projet.

## Organisation

**Apprenti** Téo Sesti

079 963 04 97

[sesti.teo@wavemind.ch](mailto:sesti.teo@wavemind.ch)

**Entreprise** Wavemind Sàrl

Y-Parc – Swiss Technopole

Rue Galilée 15

1400 Yverdon-les-Bains

024 552 00 77

**Responsable de stage** Wavemind Sàrl

Alain Fresco

024 552 00 77

**2ème Responsable de stage**Wavemind Sàrl

Romain

024 552 00 77

alain.fresco@wavemind.ch

**Expert 1** Gilbert Gruaz

gilbert.gruazduvaud.ch

**Expert 2** Nicolas Borboen

[nicolas.borboen@epfl.ch](mailto:nicolas.borboen@epfl.ch)

## Cadre de travail

### Wavemind

Le travail devait se faire dans les bureaux de l’entreprise Wavemind Sàrl, mais à cause de la pandémie du coronavirus, j’ai dû effectuer ce travail depuis chez moi.

L’entreprise Wavemind est spécialisée dans le domaine du développement web et mobiles depuis quelques années déjà. C’est dans cette société que j’ai eu le plaisir de pouvoir en apprendre davantage sur la profession d’informaticien.



### Ecole des Arche

Je suis rentré à l’Ecole des Arche à Lausanne en 2017. J’ai passé 2 ans à suivre des cours d’informatique mais aussi d’anglais, culture générale ou encore mathématique.

J’ai aussi eu la chance de pouvoir assister à des cours bloque durant mes deux années de stage pratique.

**

## Durée du travail

Mon projet de CFC informaticien se fait sur une durée 2 semaines, doit 10 jours à 8 heures par journée, cela fait l’équivalent d’environ 80 heures. Cela se déroulera du 1 mars 2021 jusqu’au 15 mars 2021.

## Planification initiale

Annexe

## Choix des technologies

### Web

#### **Ruby on Rails**

Le framework Ruby on Rails sera choisi pour ce travail. Il s’agit d’une contrainte imposée par mon responsable de stage, Alain Fresco. Etant donné que c’est sur framework que j’ai appris durant mes deux ans de stage, cela me posera aucun problème pour réaliser mon projet.



### Mobile

Le framework React Native m’a lui aussi été imposés pour ce projet. Etant donné que j’ai aussi appris ce langage durant ma formation, cela me posera aucun problème.

#### **React Native**

React Native a été inventé par Facebook et se base sur les principes de React.js.

Avantages :



* Open source
* Gain de temps
* Grande communauté

#### **Expo**

J’ai choisi expo car elle permet d’avoir une solution rapide pour proposer deux versions d’un même code. Pas besoin d’apprendre Xcode ou Android Studio pour configurer ou déployer notre application.

Expo permet d’avoir une émulation rapide et qui permet aussi de facilement partager une application ou juste un bout de code avec quelqu’un d’autre.

Avantages :

* Facile à prendre en main.
* Déployer des app rapidement et facilement.
* Très accessible

## Stratégie de test

Les tests seront effectués par moi-même pour assurer la stabilité de l’application mobile et web.

Pour la partie web, je vais m’assurer que tous les requetés API me retournent un tableau d’information, s’assurer que les restrictions de données marchent bien ou encore que les erreurs API soit bel et bien géré par le serveur web.

Pour la partie mobile, je vais effectuer mes tests sur mon smartphone Samsung Note 10. Je vais aussi faire une série de tests que je décrirais en détail plus bas dans ce rapport. Pour résumé je vais tester les différentes actions possibles dans tous les écrans différents en essayant même de fermer l’application et de la rouvrir pour voir comme elle va réagir.

Environnement de travail

### Matériels Hardware.

Voici la liste matériel hardware sur lequel j’ai réalisé mon projet :

* Processeur : Intel Core i9-9900K, 3800 MHz
* Carte mère : ASUS ROG Strix Z390-E Gaming
* RAM : 32G DDR 4
* Carte graphique : MSI 2070 Super RTX
* SSD
* HDD
* Clavier & Souris
* Ecrans

### Logiciels et OS.

Voici la liste des logiciels et de l’os que j’ai pu utiliser durant la réalisation de mon projet.

* RubyMine
* PostMan
* NotePad
* Swagger.io
* Draw.io
* Figma
* Git
* Chrome
* Windows 10

Objectif

1. La qualité du repository Git : messages de commis explicites et lisibles, permettant de retracer l'évolution du code (plusieurs commis par jour, création de branches de fonctionnalités), fichier README md présentant le projet et son déploiement.
2. Le code suit le principe de DRY (Dont Repeat Yourself), ce qui implique un code exempt de sections dupliquées et en respectant le style de programmation des langages utilisés (Rubocop et ES Lint).
3. Guide d’installation précis et reproductible (ReadMe)
4. Les différentes méthodes HTTP sont implémentées à bon escient en fonction de l'action réalisée sur la ressource indiquée.
5. Le rapport démontre que le candidat a étudié le modèle des données : un diagramme entité-association (ERD) est présent dans le rapport. Le candidat décrit et critique le diagramme et les différentes tables.
6. L'étudiant fournit une documentation de l'API, qui explique les types de données, les valeurs de retour, les différentes possibilités d'interactions avec l'API. L'utilisation d’outil tel que Swagger est recommandée.
7. Du côté mobile l’application doit gérer proprement les erreurs du serveur auront dans le cas où le serveur ne serait pas disponible que dans le cas où le serveur rendrait une erreur.

Conception

Structure de du serveur web

### Ruby on Rails

La technologie au niveau du Backend est imposée par mon responsable de stage, Alain Fresco. Le choix du framework Ruby on Rails est donc obligatoire.

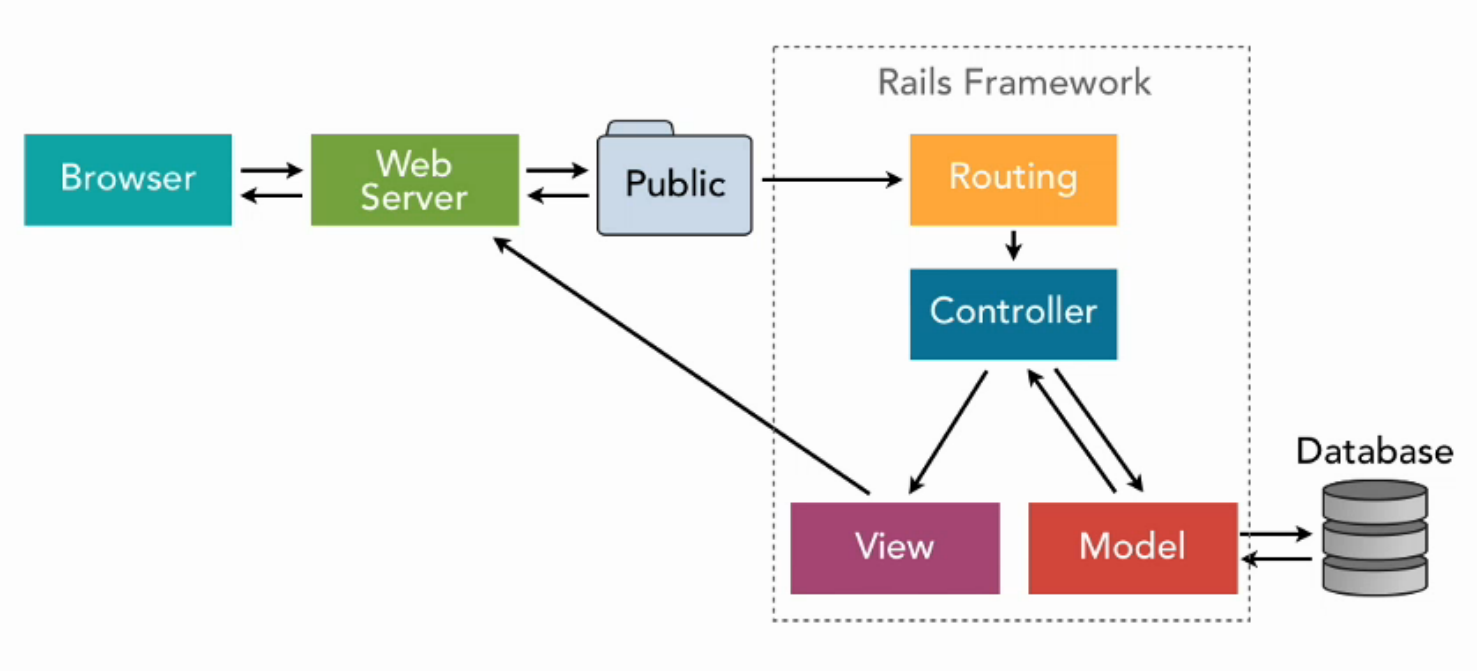
Un framework veut littéralement dire « cadre de travail ». L’objectif principal d’un framework est de simplifier la programmation en proposant des méthodes déjà toute prête et utile pour le projet en question. Les frameworks comportent plusieurs avantages comme ceux-ci :

* **Un code qui peut être réutiliser à plusieurs endroits.**
* **Une standardisation du code**
* **Gain de temps pour la programmation**
* **Mise en place rapide de la structure de base de l’application**

Le framework Ruby on Rails utilise l’architecture MVC qui possède trois noyaux principaux :

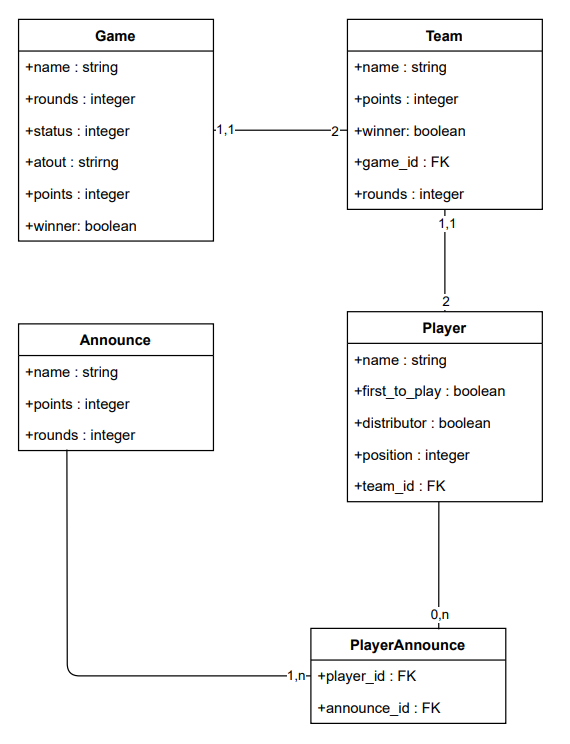
* **Modèle**
* **Vue**
* **Contrôleur**

Pour résumé ces trois parties, cela va nous permettre de bien distinguer le code par rapport à leur utilisation. Cela va premièrement nous permettre de rendre le code plus lisible mais aussi plus compréhensible. Comme dit avant chaque noyau à dont un but précis, d’abord le **modèle** qui va communiquer avec la base donnée avec les différentes méthodes définit à l’intérieur. La **vue** va interpréter tout le code HTML, CSS pour afficher quelque chose à l’écran de l’utilisateur et pour finir il y a le **contrôleur**, qui va communiquer avec le **modèle** et la **vue**.



### Base de données

Voilà le fichier SQL sur lequel j’ai basé toute mon application.

****

Description des entités du fichier SQL dans le tableau ci-dessous :

|  |  |
| --- | --- |
| Entités | Explication |
| Game | Regroupe toutes les parties de l’application. Permet de stocker le nom, les manches, le statuts et l’atout de la partie. |
| Team | Comportes toutes les équipes d’une partie. Permettre de définir le nom, les points et la partie à la quel elle appartient. |
| Player | Comportes tous les joueurs d’une partie.  Permet de définir le nom, le distributeur, le premier à jouer et la team a la quel elle appartient. |
| PlayerAnnounce | Table intermédiaire qui permet de faire le lien des annonces par rapport à joueur et vice verse. |
| Announce | Regroupe toutes les annonces d’une partie.  Permet de définir le nom, les points et les manches d’une annonces. |

Structure de l’application mobile

### Expo

Expo est une plate-forme qui permet de créer des applications pour Android, iOS et pour le Web grâce au JavaScript et à React. Expo possède un ensemble d’outils et de composant déjà prêt autour de React Native.

Il y a deux étapes ici, quand vous lancez « expo start » dans votre console, le démarrage de l’application avec Expo CLI ce fait. Vous exécutez aussi Expo Développent Server et Metro.



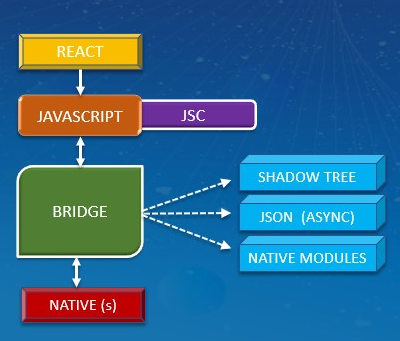
J’utilise la version **4.0.7** d’expo.

### React Native

React Native est un framework qui nous offre un gain de temps avec la possibilité de livrer deux versions de votre application mobile. Elle permet d’optimiser les performances pour une bonne fluidité et une bonne réactivité au niveau de l’interface graphique. React Native s’appuie sur React.js et est écrit dans le langage de programmation JavaScript.

Il y a quatre sections principales :

* Le code React que le programmeur écrit.
* Le JS qui est interprété
* Une série d’éléments connus sous le nom de « Brige »
* Le coté Native de l’application



La liste ci-dessous liste les différents dossiers de la structure de l’application ainsi qu’un texte explicatif sur l’utilisation de ses différents dossiers. J’utilise la version **0.63.2** de React Native.

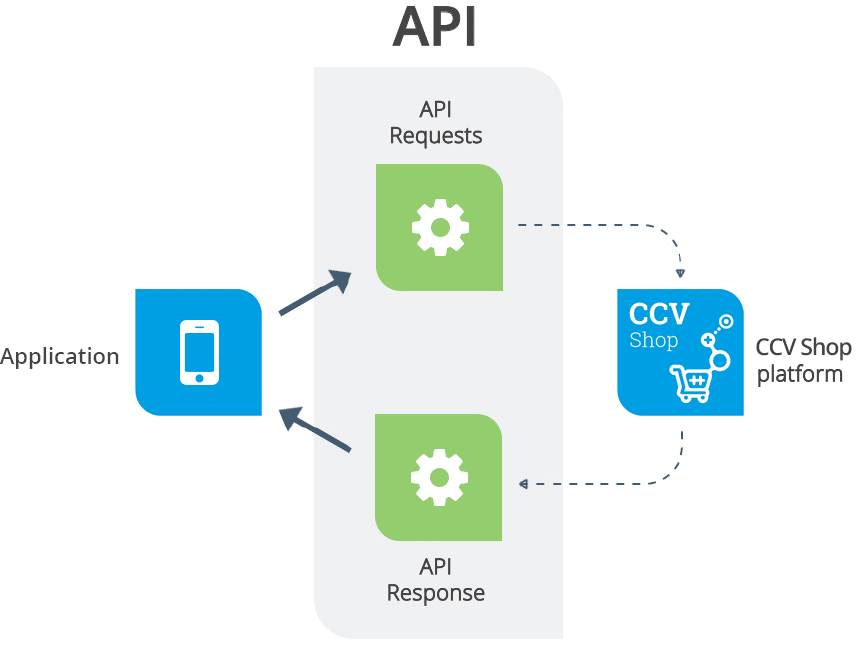
* **« assets » :** C’est ici que sont stocké toutes les images, ou autres médias qui seront utilisés dans l’application.

* **« src/common » :** C’est ici que sont stockés tous les fichiers communs (méthodes, constantes). Cela nous permet de définir une seule fois quelques valeurs pour ensuite pouvoir les réutiliser par la suite dans d’autre fichiers sans les réécrire à chaque fois
* **« src/components » :** Les composants sont utilisés dans les différentes vues dans le but d’afficher quelque chose à l’écran de l’utilisateur.
* **« src/screens » :** Les écrans ou « screens » sont la finalité et l’assemblage des composants.**-+**

React Native utilise la convention PascalCase pour le nommage de ses différentes vues. Le PascalCase consiste à Mettre une majuscule pour chaque début de mot. Pour les noms de ses composants l’on va utiliser la convention kebab-case, c’est-à-dire que l’on ne pas utiliser de majuscule et mettre un tiret entre chaque mot .Il y a aussi le camelcase qui consiste à mettre une majuscule à partir du 2e mot dans notre variable

API

Une API (Application Programming Interface) est un ensemble normalisé de classes, de fonctions ou encore de méthode avec lesquels a y va pouvoir communiquer diverses informations. On parle d’API au moment où une entité informatique agit avec un autre système tiers.



Dans ce travail de fin d’année, il y a une communication entre mon serveur Rails et mon application React Native qui se fait par le biais d’une AP

Mon serveur web propose une API **REST** (Representational State Transfer) ce qui me permet d’utiliser les verbes **GET**, **POST**, **PUT**, **DELETE** et **UPDATE**.

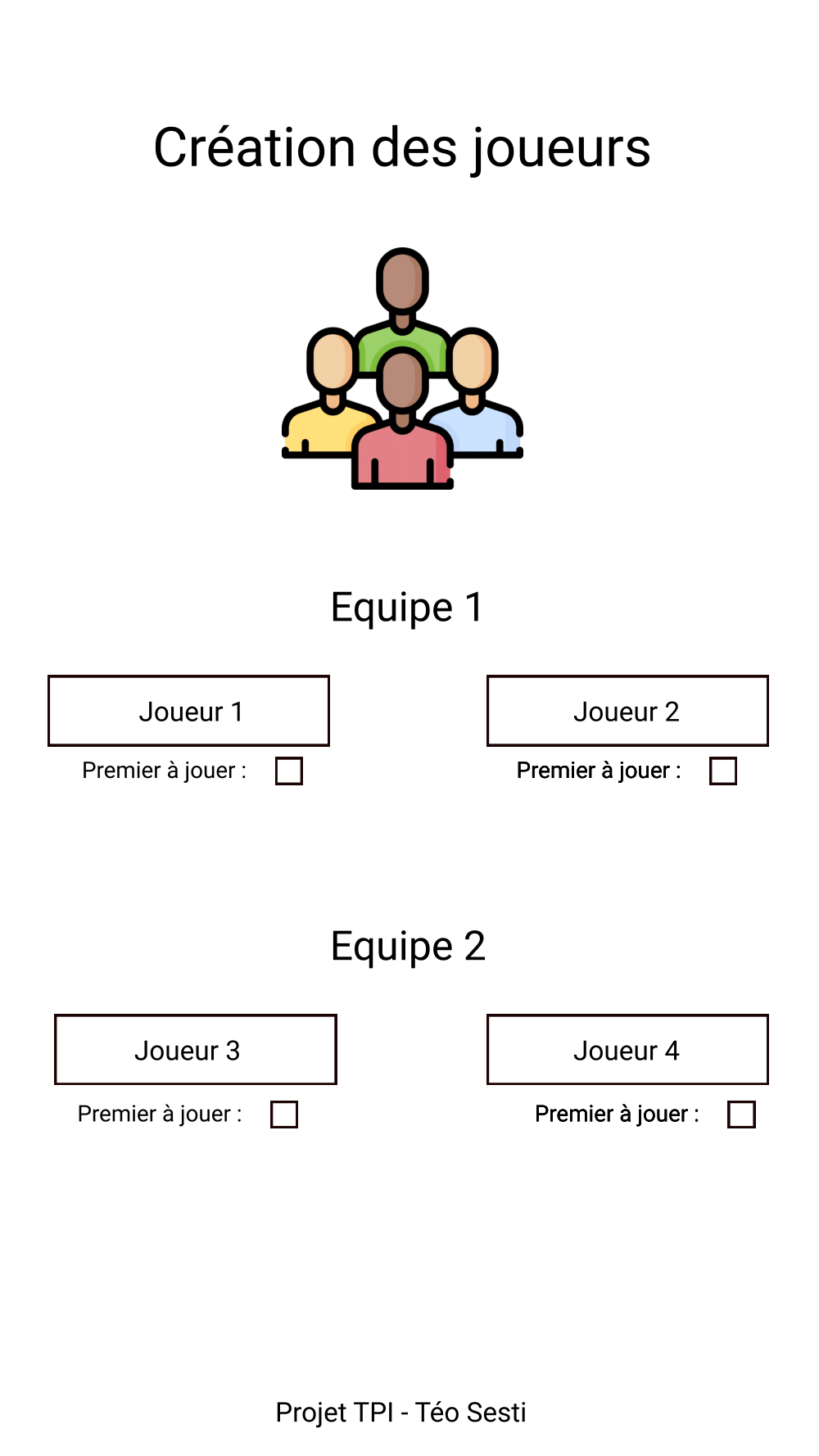
Toutes les routes disponibles dans mon application sont disponibles. Ces routes ont été généré à partir de l’outil Swagger.

Pour pouvoir profiter d’une vue d’ensemble, vous pouvez vous rendre à l’adresse suivante : <http://editor.swagger.io> et y mettre le code que vous trouverez dans le fichier API.txt dans la zone pour le **YAML.** Il est difficile d’offrir un format papier pour ce document car la vue d’ensemble sur l’éditeur est interactive.

Maquette graphique

J’ai utilisé le site web https://www.figma.com/ pour pouvoir réaliser ses différentes maquettes. L’outil est vraiment simple à prendre en main et j’ai pu facilement avoir un résultat qui me plaisait.



****

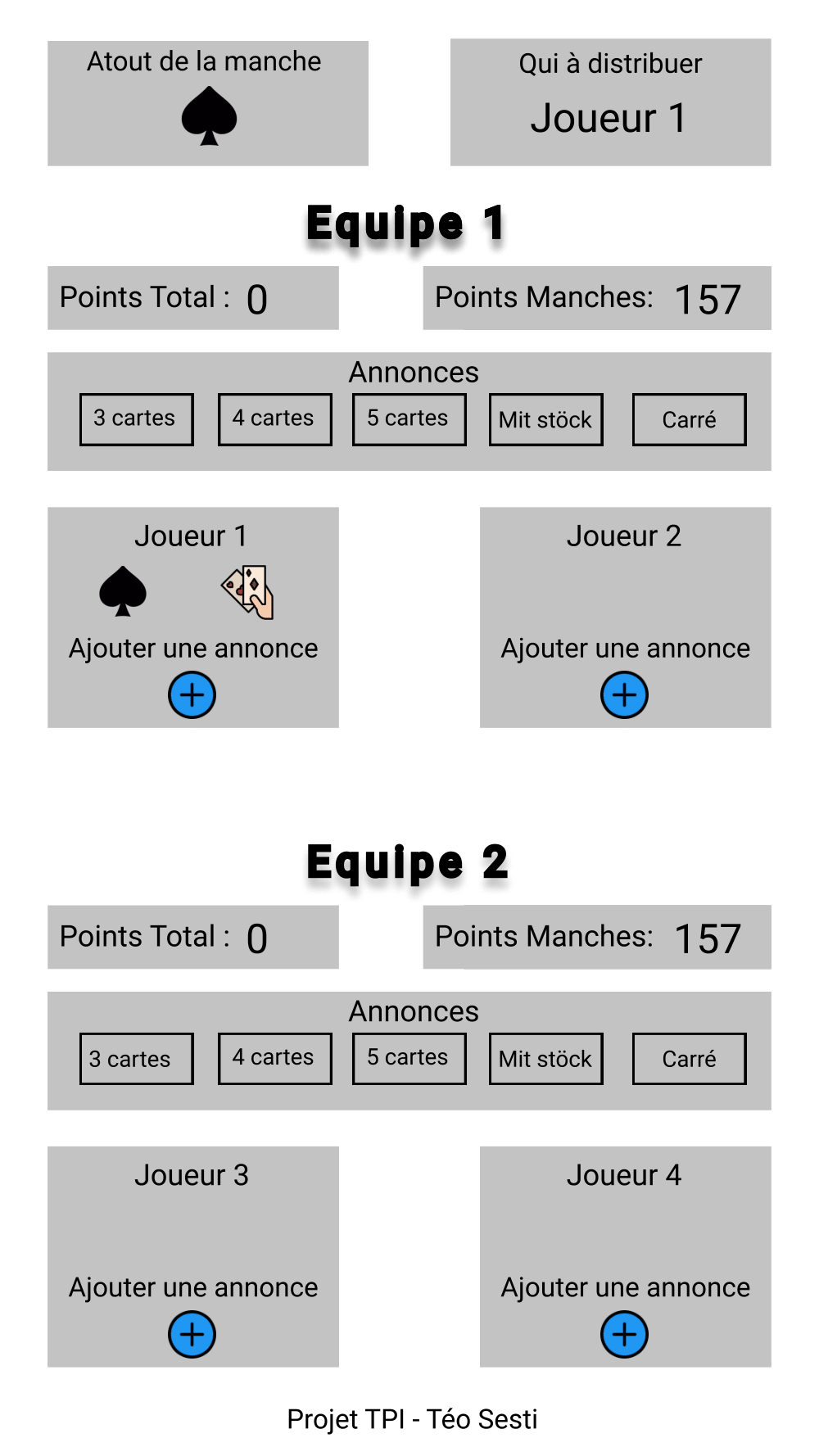
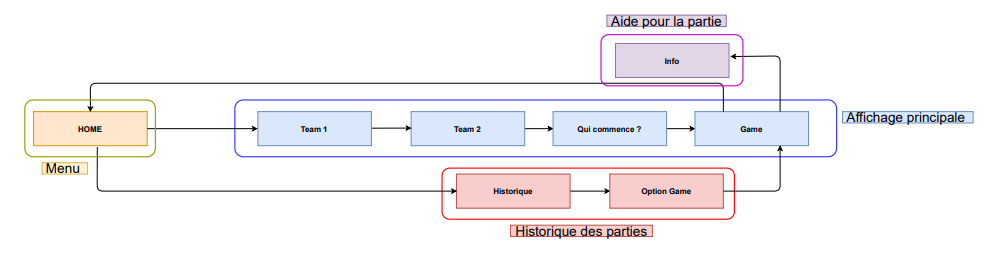
****

Schéma de navigation

### Mobile

Ce schéma vous présente la navigation sur la partie frontend de l’application, les carrés sont les différents écrans disponibles et les flèches vous permettent de savoir par quel écran vous pouvez où y rendre.



Risques techniques

J’ai beaucoup travaillé sur des sites web durant mon stage chez Wavemind et c’est seulement après 1 ans que je me suis mis au Ruby et seulement y a quelques mois à React native. Je n’ai donc pas toutes les normes acquises ou encore tous les reflexe pour appliquer toutes les conventions pour le React Native par exemple. J’ai heureusement eu la chance de développer une app sur React native avant de faire celle-ci. J’ai peur que mon app ne soit pas pensé de façon à ce qu’elle peut être améliorée facilement par la suite.

J’avais peur de ne pas avoir les compétences nécessaires et j’ai dû aller m’informer sur quelque pratique que je n’avais pas eu le temps de voir pour mon projet. Comme la mise à jour du State ou le **useEffect** donc je n’ai pas encore tout compris à vrai dire

* *risques techniques (complexité, manque de compétences, …).*

*Décrire aussi quelles solutions ont été appliquées pour réduire les risques (priorités, formation, actions, …).*

Planification réelle

Annexe -> Voir Planification

Réalisation

Introduction

Une des premières étapes, lors de la mise en place d’une application rails, est de mettre en route les migrations et les modèles.

Ruby on Rails

### Configuration

#### **Serveur**

* Serveur : Heroku
* OS : Ubuntu
* DB : PostgreSQL

Application

* Framework : Ruby on Rails
* Version : 3.0.0
* URL : https://chibre-manager.herokuapp.com/

#### **Fonctionnalités**

* Game
  + Création
  + Modification
  + Listing
  + Suppression
* Announce
  + Création
  + Suppression
* API

### Mise en fonction des migrations et des modèles

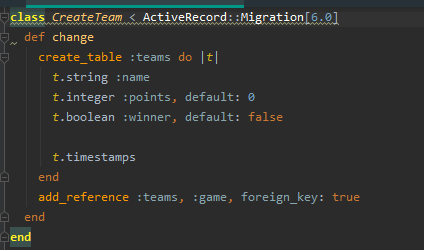
Une des premières étapes, lors de la mise en place d’une application rails, est de mettre en route les migrations et les modèles.

Les Migrations

Pour générer les migrations, il faut lancer la commande suivante dans votre console :

rails db:migrate

Tous ces fichiers vont crées les tables ainsi que les champs et les associations des différents tableaux. Voilà à quoi ressemble un fichier de migration



Le « t.string : name » va créer un champ de type « string » nommé « name » et ainsi de suite pour le reste des lignes de code. Le « add\_reference » va annoncer qu’il y a une relation entre la table « Team » et « Game ». On appelle ça en anglais « Foreign Key »

Les Modèles

Une fois les migrations faites, il vous faut créer des modèles. Ça va nous permettre d’effectuer des requêtes sur notre serveur web grâce à l’ActiveRecord de Ruby on Rails.

ActiveRecord est une façon de lire des données d’une base de données. Les différents champs d’une table ou d’une vue sont mis dans une classe. Ainsi l’objet créé est lié à un tuple de la base.

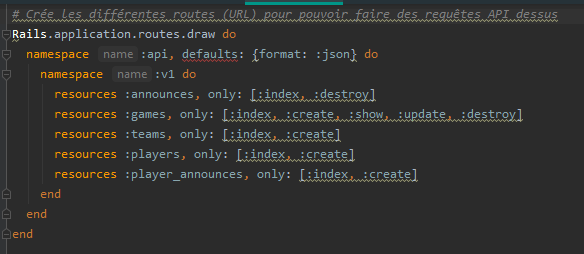
Les modèles vont permettre de définir les relations des différentes tables, ainsi que des validations sur des paramètres. L’on peut aussi définir certaines actions qui vont agir selon notre demande.

### Les Routes

Je n’ai défini que des routes pour mes API car toute mon application sera visible seulement depuis l’application React Native.

API

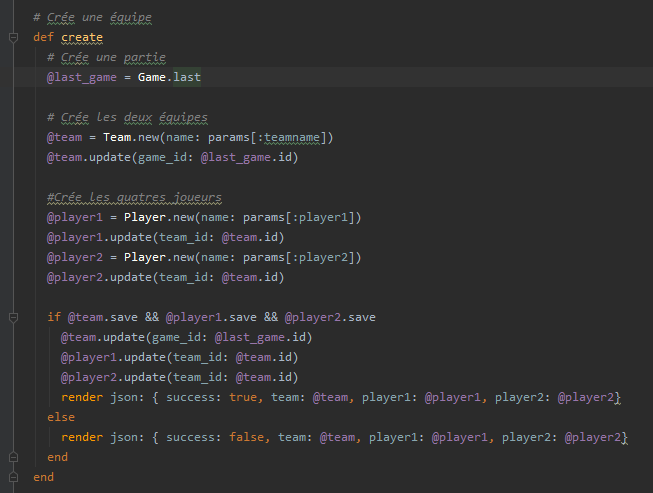
Voilà les routes que j’ai pu configurer pour mon application Chibre Manager.

**

J’ai créé une route « /api/v1/ » grâce à une des premières lignes de commande et à partir de cette adresse, toutes les autres adresses pour « announces, teams, games, player et player\_announces » sont ainsi créés. L’option « only » indique qu’il ne faut créer que des adresses pour ces méthodes seulement.

### Les Contrôleurs

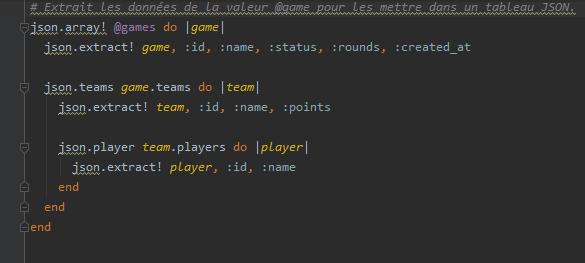
Les contrôleurs vont nous permettre de créer, supprimer ou encore mettre à jour une classe. Voici à quoi ressemble la méthode « create » définit dans le contrôleur « Team ».

**

Le contrôleur va d’abord créé une partie, deux équipes et pour finir les quatre joueurs avec les paramètres passés. Il va ensuite s’assurer que tout est correct avec la commande « .save » et ainsi sauvegardé toutes ses actions.

### Les Vues

Les vues vont me permettre de renvoyer un tableau en format JSON pour pouvoir ensuite utiliser ses informations dans mon application React Native. Voici un exemple d’une vue.



Les lignes suivantes vont extraire les données de la variable « game » ainsi que toutes les équipes et joueur liées à cette partie.

React Native

### Configuration

#### **Application**

* Framework : React Native
* Version :
* URL APK : http://gofile.me/6NSuo/AhpWDppRx

#### **Fonctionnalités**

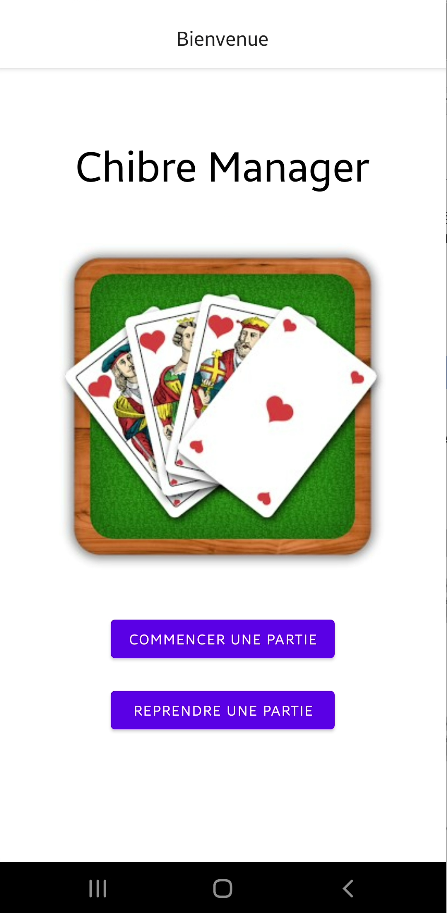
* Création des joueurs
* Créations des équipes
* Création d’une partie
* Possibilité de reprendre une partie
* Comptage des points
* Atout de la manche
* Respecter l’ordre des joueurs à la manche suivante
* Lier une annonce à un joueur
* Définit qui gagne une partie
* Définit qui doit distribuer
* Définit qui doit commencer

### Les Vues

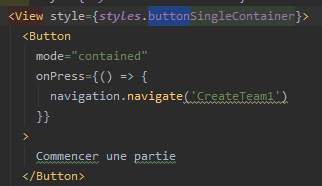
#### **Menu d’accueil**

Lorsque l’utilisateur va commencer à utiliser l’application Chibre Manager, il va forcément arriver sur menu là. C’est à partir du menu que l’utilisateur va pouvoir soit commencer une nouvelle partie soit tout simplement reprendre une partie non terminée ou simplement consulter les informations d’une ancienne partie.

En tant donnée que mon application compte les points d’un jeu de cartes, j’ai donc mis naturellement une image de ce genre.



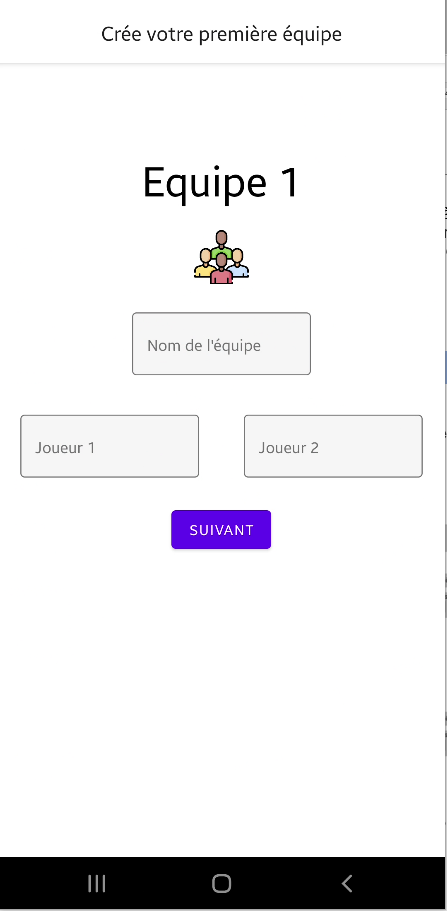
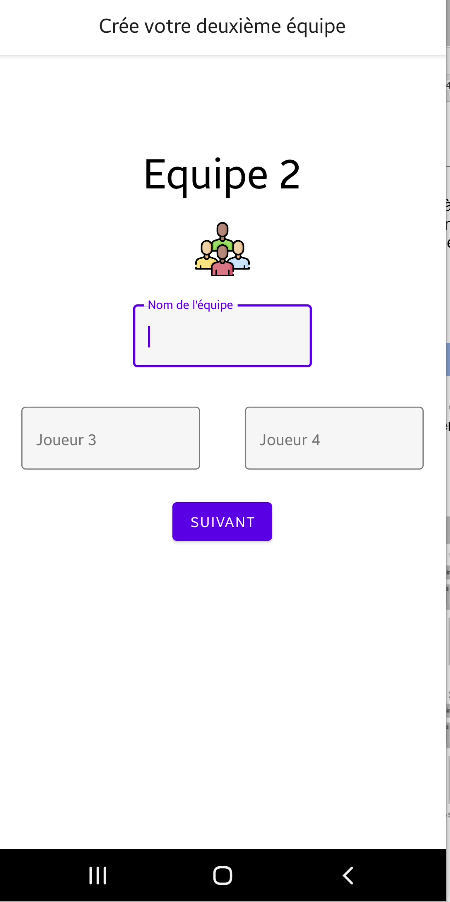
### 



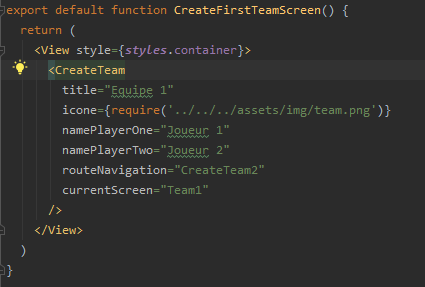
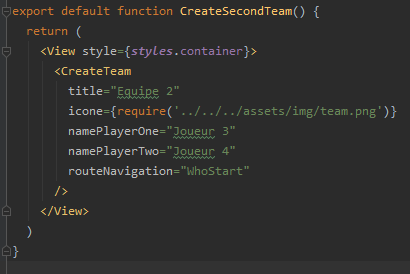
Le bouton « Commencer la partie » va amener l’utilisateur sur un formulaire pour pouvoir créer tous les éléments pour pouvoir commencer une partie. Le deuxième bouton « Reprendre une partie » va lui nous faire changer d’écran pour aller sur une liste de partie déjà créée auparavant. Voilà à quoi ressemble bouton pour commencer une partie.

#### **Création des équipes et des joueurs**

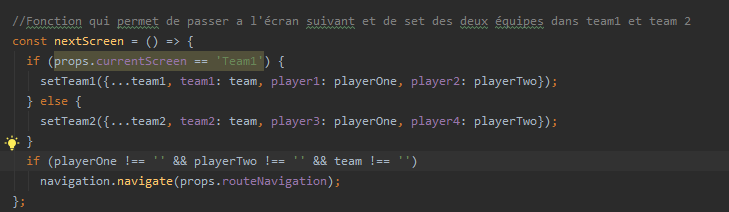
Lorsque l’utilisateur va vouloir créer une nouvelle partie il va devoir définir les quatre joueurs et les définir dans les deux équipes. Comme cité dans les placeholder, la création de l’équipe se fera grâce au champ « Nom de l’équipe » pour les joueurs ce sera dans les deux champs restants (J1 et J2 ou J3 et J4).



La première vérification se fait au niveau des valeurs que vous avez mises dans les différents champs. En effet il ne sera pas possible de continuer sans que chacun des champs ne soit rempli par au moins une lettre. Cette vérification me permet de m’assurer que toutes les informations nécessaires sont bel et bien présentes pour la création d’une partie. Pour pouvoir afficher ces deux écrans j’ai utilisé le composant « team-view » avec des informations différentes à chaque fois.

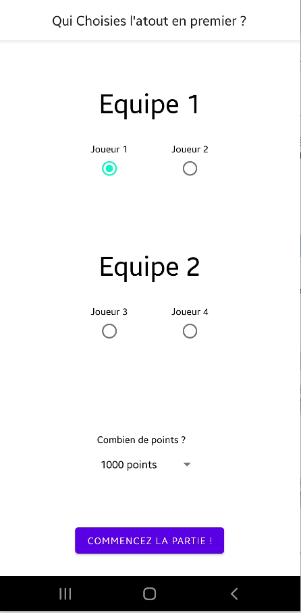


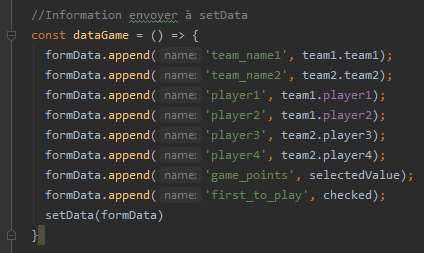
Voilà la méthode « nextScreen » qui va préparer le futur tableau de données pour pouvoir l’envoyer avec notre requête API.



#### **Qui commence et combien de points pour la partie ?**

Arriver sur cet écran l’utilisateur va devoir choisir le joueur qui devrait commencer en choisissant en premier l’atout de la première manche. L’utilisateur devra s’assurer de mettre le nombre de points maximaux pour la partie créée pour qu’une équipe puisse gagner.





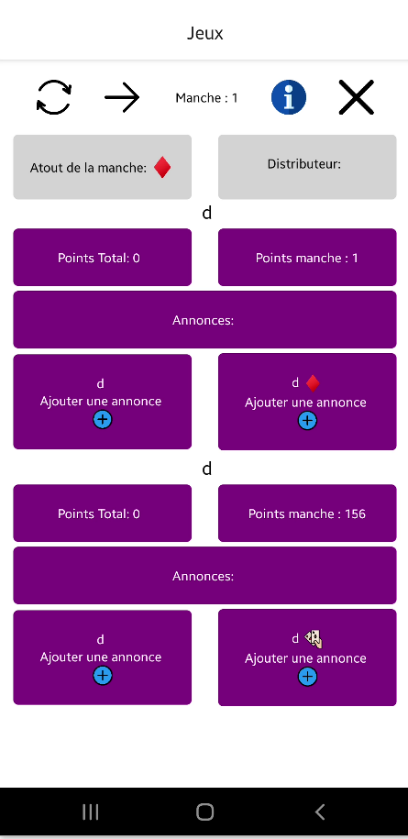
Une fois toutes les options mise en place, l’utilisateur va envoyer toutes ces données avec une requête API qui se fera une fois que le bouton « Commencez la partie » sera appuyé par l’utilisateur. Voilà à quoi ressemble le tableau envoyé par notre requête

#### **Partie**

Cette vue est la vue principale de l’application, c’est ici que l’on va pouvoir utiliser cette application pour nous aider dans le comptage des points au chibre. Tout d’abord l’utilisateur devra rafraichir l’application à l’aide du bouton « rafraichir ».

Cela va initier toutes les données pour le commencement de la partie. Ensuite l’utilisateur aura la possibilité d’ajouter des annonces qu’il pourra attribuer à un joueur d’une équipe.

Une fois les tous les annonces mise, l’utilisateur n’aura plus qu’à rentrer les points de la manche dans l’input prévu à cet effet. Une fois toutes les actions faites, l’utilisateur devrai passer à la manche suivante en clique sur la flèche qui va additionner tous les points des équipes pour les mettre dans points total.

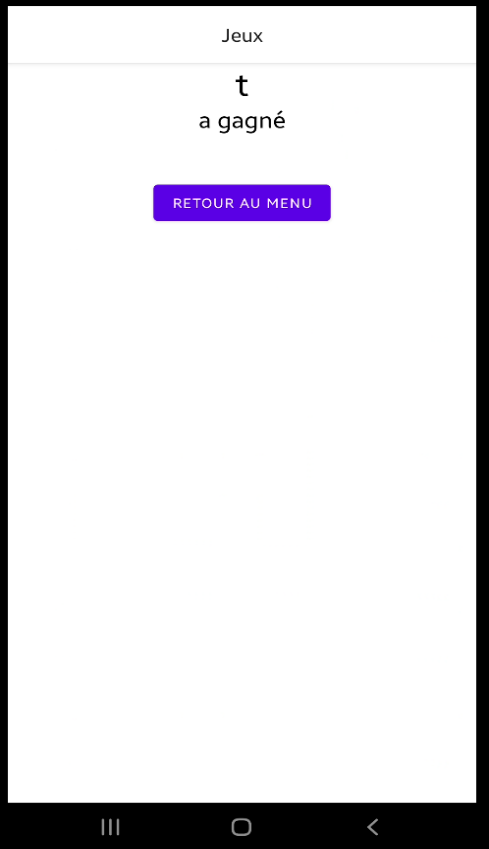


Le faite de rafraichir va faire une requête API avec la méthode GET. La méthode pour passer à la manche suivante va faire une autre requête API mais avec la méthode PUT pour pouvoir faire une mise à jour de la partie actuelle avec les différentes valeurs de la manche. Tout cela est possible grâce aux différentes méthodes définit sur le serveur web. Le bouton bleu permet d’attribuer des annonces en appelant un autre écran qui vous sera présenté plus loin dans ce dossier. La partie va aussi détecter automatiquement quand une équipe aura dépassé le seuil des points défini par l’utilisateur. L’icône à coté de carte à côté du joueur signifie que c’est a lui de distribuer tant dit que l’icone en forme d’atout à côté du joueur signifie que c’est à lui choisir l’atout pour la manche courante.

#### **Lorsqu’une équipe gagne**

Cet affichage apparaît quand une des deux équipes à dépasser 1000 ou 1500 points.

Le nom de l’équipe gagnante est affiché et un bouton apparaît pour revenir au menu principal de l’application.

******

Ce code permet d’afficher la vue ci-dessus, j’aurais pu en faire un composant mais j’avais besoin de la valeur gameData pour vérifier si une équipe avait gagné. Dans ce code vous pouvez voir l’affichage du nom de l’équipe gagnante ainsi que qu’un bouton généré par la bibliothèque React Native Paper.

******

#### **Liste des parties crées**

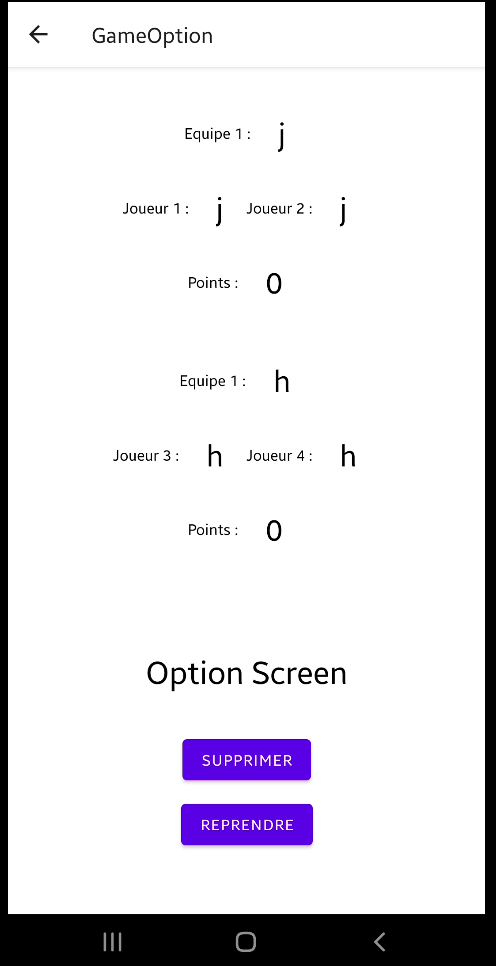
Cette vue nous liste toute la partie créée auparavant. Elle nous indique le nom de la partie, le nom des deux équipes et la date de création.



Toute forme de liste sous React Native est gérée grâce au composant Flatlist qui va nous permettre de créer facilement une liste selon un tableau d’objet. Il aurait été plus judicieux de ma part de faire des cartes pour lister les différentes parties car il n’y a pas beaucoup de place à l’horizontale. La Flatlist va nous afficher le nom de la partie, le nom des deux équipes ainsi que la date de création de celle-ci. La Flatlist est générer grâce à une requête API avec la méthode ‘GET’ qui va nous renvoyer toutes les parties crée dans l’application.

#### **Option d’une partie**

Cette vue nous liste toute la partie créée auparavant. Elle nous indique le nom de la partie, le nom des deux équipes et la date de création. Un bouton va permettre à l’utilisateur de pouvoir supprimer cette partie ainsi qu’un autre bouton pour pouvoir reprendre une partie qui ne serait pas encore terminée.



Cette vue nous liste toute la partie créée auparavant. Elle nous indique le nom de la partie, le nom des deux équipes et la date de création. Un bouton va permettre à l’utilisateur de pouvoir supprimer en envoyant une requête API avec la méthode ‘DELETE’ ainsi que l’id de la partie. Un autre bouton va permettre à l’utilisateur de pouvoir reprendre une partie, ayant l’id de la partie stockée, il suffit de faire une requête GET avec l’id de celle-ci pour pouvoir récupérer toutes les informations de la partie à reprendre en s’assurant que la partie n’a pas de gagnant en vérifiant la valeur : winner d’une partie.

#### **Atout**

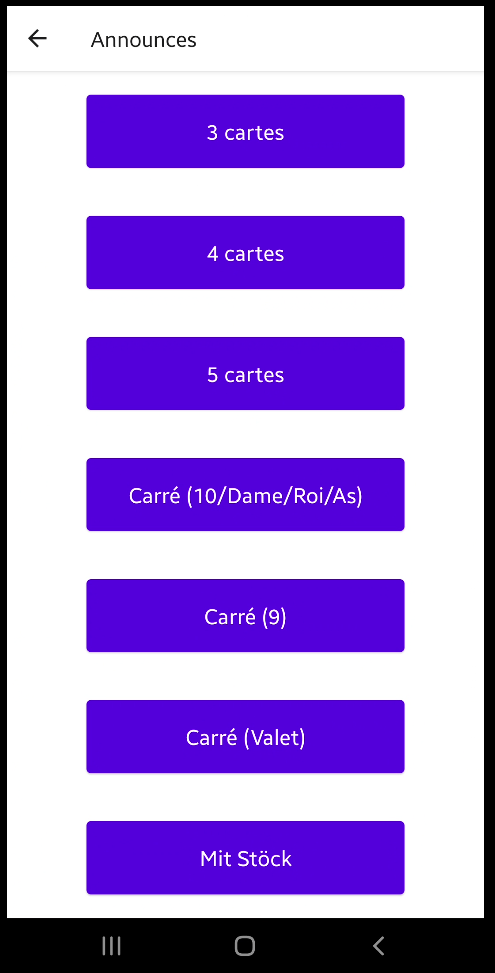
Cette vue va nous permettre de définir l’atout pour la manche courante. Le fait de chibre va donner le tour à votre coéquipier et c’est celui qui va donc choisir l’atout à la place du joueur initial.



En cliquant sur l’un des atouts à l’écran une requête API avec la méthode ‘PUT’ va être envoyée en prenant en compte l’atout choisit et s’il y a chibre ou pas. Ensuite l’utilisateur sera renvoyé dans l’écran principal de la partie, il lui restera plus qu’a rafraichir pour voir les nouvelles informations apparaître à l’écran.

#### **Les Annonces**

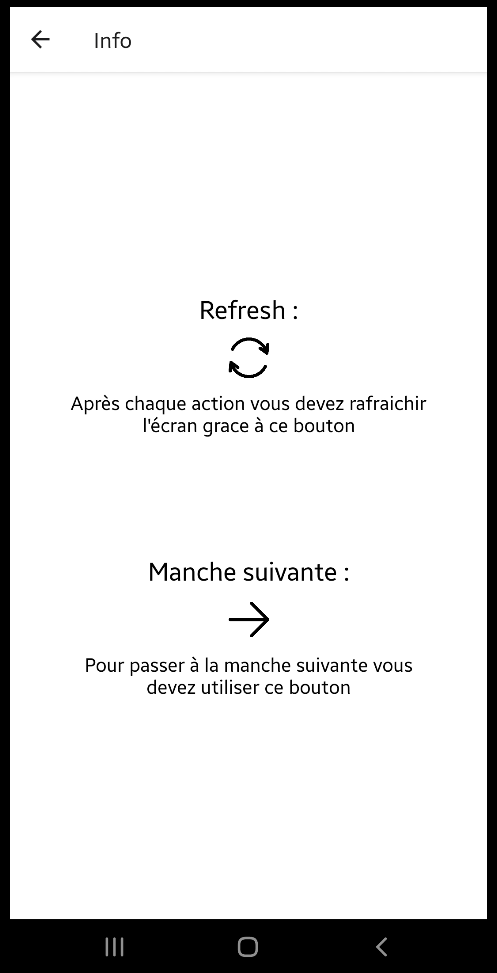
Cet affichage va proposer plusieurs annonces à l’utilisateur qu’il va pouvoir sélectionner selon sa demande. Il devra ensuite revenir sur l’écran principal de la partie et cliquer sur le bouton rafraichir pour pouvoir afficher les nouvelles informations sur son écran.

**

Pour pouvoir afficher ses différents éléments, j’ai créé un fichier AnnounceScreen qui lui va appeler plusieurs fois le même fichier, « Announce », mais pas avec les mêmes informations à chaque fois. Une fois l’annonce sélectionnée, une requête API avec la méthode ‘POST’ va être envoyée sur notre serveur web.

#### **Les Infos**

Cette vue va nous donner les différentes informations concernant les boutons qui sont présents durant une partie.

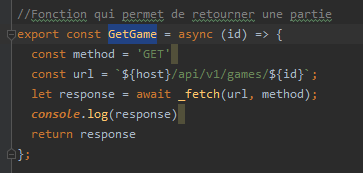
**

API

### Comment marche mon API

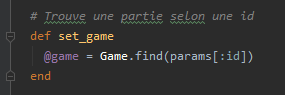
#### **Mobile**

La constante getGame va retourner un tableau au format JSON avec toutes les informations nécessaire grâce à l’id qu’il recevras. Voici le code ci-dessous.

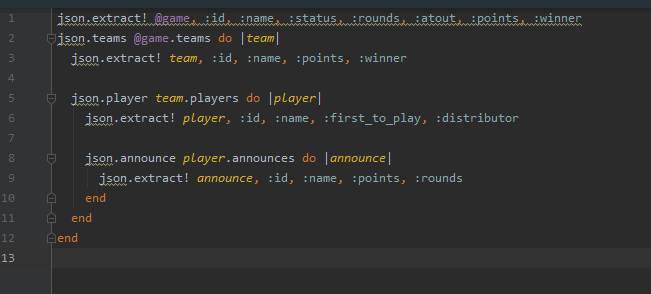


#### **Serveur Web**

Grace a l’id reçu, le serveur web va pouvoir rechercher la partie concerner et ainsi retourner toute l’information définit dans la vue à l’application. Voilà la méthode « set\_game » qui fera ce que j’ai dit plus haut.

******

Voilà a quoi ressemble la vue sur le serveur web, c’est ici que l’on va définir les différents champ a renvoyer. Comme décrit sur cette image l’on peut voir que le tablea json va contenir l’id, le nom, les manches, l’atout, les points et s’il y a un gagnant pour la partie crée. Il y aura aussi toutes les équipes et joueur lier a cette partie.

**

*Décrire la réalisation "physique" de votre projet*

* *Le logiciel ce situe dans mon repos github (*[*https://github.com/nours33/Chibre-manager/tree/master*](https://github.com/nours33/Chibre-manager/tree/master)*) La section serveur dans backend et la section application dans le dossier frontend*
* *Backend : Dossier App -> Dossier Controller, Dossier Model, Dossier Vue. Dossier db -> Dossier migrate.*
* *Frontend : Dossier src -> Dossier Common, Dossier components, Dossier Screens. Dossier assets.*
* *J’ai utiliser Windows 10 64bits, avec l’IDE RubyMine de Jetbrains, Postman pour me requete API, Swagger pour l’information de l’api, Figma Pour la realisation des maquette, Github pour avoir une trace de mon code.*
* *les répertoires où le logiciel est installé*
* *la liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)*
* *les versions des systèmes d'exploitation et des outils logiciels*
* *la description exacte du matériel*
* *le numéro de version de votre produit !*
* *programmation et scripts: librairies externes, dictionnaire des données, reconstruction du logiciel - cible à partir des sources.*

*NOTE : Evitez d’inclure les listings des sources, à moins que vous ne désiriez en expliquer une partie vous paraissant importante. Dans ce cas n’incluez que cette partie…*

Tests

Les tests suivants ont été fait à partir de mon smartphone Samsung Note 10.

### Erreurs restantes

* 1. Lorsqu’il y a un match pour la 2ème équipe, les 257 points ne sont pas correctement attribués. A la place ils reçoivent 157 points, comme si la valeur match n’était pas prise en compte. Gros problème car le comptage des points sera forcément faux.
  2. Pas d’affichage de confirmation quand on efface une partie dans l’écran « Historique des parties ».
  3. Rien ne nous indique qu’il faut absolument remplir les champs Joueur et Equipe pour pouvoir continuer sur l’écran suivant.
  4. Besoin de rafraîchir à chaque que l’on veut mettre à jour une information sur l’écran en cliquant sur le bouton prévu à cet effet.
  5. Invalidité des dates sur l’écran « Historique des parties »

### Application mobile

#### **Menu**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Cliquez sur le bouton « Commencer une partie » | Changement d’écran pour aller sur la création d’équipe numéro une. | ok |
| Cliquez sur le bouton « Reprendre une partie » | Changement d’écran pour aller sur l’historique des parties crée | ok |

#### **Historique des parties**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Affichage de la liste des parties lorsqu’on arrive sur la page | Affichage de la liste des parties | ok |
| Clique sur une partie dans la liste | Changement d’écran pour aller sur l’historique de la partie sélectionner | ok |

#### **Option de la partie**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Affichage des différentes informations lors qu’on arrive sur la page | Affichage du nom des deux noms des équipes ainsi que le nom des quatre joueurs et les points des deux équipes | ok |
| Cliquez sur le bouton « Supprimer » | Suppression de la partie sélectionné | ok |
| Cliquez sur le bouton « Reprendre » | Changement d’écran pour aller sur l’écran d’une partie. | ok |

#### **Ecran de la partie**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Clique sur le bouton rafraichir | Actualiser les informations sur l’écran de la partie. Atout, nouveau distributeur, points des deux équipes, celui qui doit choisir l’atout | ok |
| Clique sur le bouton pour aller à la manche suivante | Actualise la partie courante. | ok |
| Clique sur le bouton information | Changement d’écran avec les différentes informations pour la partie. | ok |
| Clique sur le bouton croix | Quitte la partie et dirige l’utilisateur au menu de l’application | ok |
| Clique sur l’atout de la manche | Ouvre un écran qui affiche tous les atouts possible | ok |
| Clique sur les points de la manche de l’équipe 1 | Change d’écran et permet à l’utilisateur de changer les points de la manche courante | ok |
| Clique sur le bouton bleu avec une croix dessus | Change d’écran et affiche toutes les annonces possible | ok |

#### **Création d’une équipe et des joueurs**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Remplir tous les champs | Redirige l’utilisateur à l’écran suivant. | ok |
| Remplit seulement le champ pour le nom de l’équipe | Rien ne se passe | ok |
| Remplit seulement les deux noms de l’équipe | Rien ne se passe | ok |
| Ne remplit rien du tout | Rien ne se passe | ok |

#### **Choix du joueur qui commence**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Choisi le joueur 1 en tant que premier joueur | Le joueur 1 est la personne qui choisit l’atout | ok |
| Choisi le joueur 2 en tant que premier joueur | Le joueur 2 est la personne qui choisit l’atout | ok |
| Choisi le joueur 3 en tant que premier joueur | Le joueur 3 est la personne qui choisit l’atout | ok |
| Choisi le joueur 4 en tant que premier joueur | Le joueur 4 est la personne qui choisit l’atout | ok |
| Choisit 1000 points pour la partie | 1000 points sont attribués à la partie | ok |
| Choisit 1500 points pour la partie | 1500 points sont attribués à la partie | ok |

#### **Choix de l’atout**

Tests Résultat attendu Résultat

|  |  |  |
| --- | --- | --- |
| Choisis l’atout de trèfle | L’atout de trèfle est sélectionné | ok |
| Choisis l’atout de carreaux | L’atout de carreaux est sélectionné | ok |
| Choisis l’atout de pique | L’atout de pique est sélectionné | ok |
| Choisis l’atout de cœur | L’atout de cœur est sélectionné | ok |
| Choisis de chibre | Le coéquipier de la personne qui chibre est choisit | ok |
| Choisis de ne pas chibre | C’est au joueur sélectionner qui a choisi l’atout | ok |

Liste des documents fournis

*Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions*

* *le rapport de projet*
* *le manuel d'Installation (en annexe)*
* *le manuel d'Utilisation avec des exemples graphiques (en annexe)*
* *autres…*

Conclusions

Conclusion personnelle

Premièrement ce travail m’a permis d’enrichir tous mes connaissances concernant les technologies Rails et React Native. C’est aussi durant ce stage de 2ans que j’ai travaillé avec Rails et React Native pour la première fois. J’ai eu aussi la chance de beaucoup travailler avec Wordpress, malheureusement je n’ai pas eu la chance de pouvoir appliquer ce que j’ai appris pour ce projet. Mais grâce à ce projet j’ai réellement pu avoir me challenger car ça ne faisait pas longtemps que j’avais appris le langage React native.

Deuxièmement, je ne connaissais pas du tout le Chibre, cela m’a donc motivé à l’apprendre et à connaitre toutes les règles nécessaires pour pouvoir y jouer. Cela n’a pas été facile car je ne suis pas un grand amateur de cartes de bases mais j’avoue que j’ai eu un réel plaisir à pratiquer quelques parties avec des amis.

Finalement ce projet servira surement à tous les amateurs de chibre et je te tiens a remercié Wavemind pour leur soutien durant ces 2 ans de stage

Bilan des objectifs

Repo GitHub : Il est vrai que tous mes messages commit ne sont pas tout explicite par rapport à ce que j’ai fait.

Principe de DRY (Dont Repeat Yourself) : Le principe même de React Native est de pouvoir écrire une seule fois un composant pour pouvoir le réutiliser plusieurs fois ailleurs dans notre application. Il me semble que c’est un choix judicieux concernant cette technologie. Il est vrai que j’ai du répéter du code dans la partie serveur web.

Guide Readme : Il y a un guide d’installation présent sur le repo GitHub. Cela vous explique les différentes étapes à réaliser pour pouvoir installer l’application sur votre machine.

Méthodes http : Les différentes méthodes **GET**, **PUT**,**POST**,**DELETE** ont été utilisé à bon escient et en fonction de l’action à réaliser.

Diagramme ERD : Présent dans le repo GitHub. Ce document vous explique la structure de la base de données utilisé pour cette application.

Swagger : Il y a un fichier API.txt sur le repo GitHub.

Erreur serveur : Les erreurs du serveur sont gérées grâce au code renvoyé dans le tableau json

Dans l’état générale l’application fait ce que le cahier des charges mais je reconnais très volontiers que l’application n’est pas du tout optimale. Le problème et le fait qu’il faut rafraichir à chaque nouvelle action pour pouvoir voir les nouvelles informations à l’écran. De plus l’interface utilisateur est à revoir car je trouve qu’il n’est pas vraiment fait pour compter les points au chibre.

.

*Difficultés rencontrées*

Les difficultés que j’ai pu rencontrer sont celle avec React Native. React Native est un langage informatique avec le quel je ne suis pas du tout habitué. Il m’a fallu donc du temps pour réaliser des tâches qu’il ne l’était pas vraiment. De plus j’ai dû apprendre toute la partie des requête API car c’était une des premièrement fois que je faisais cela. Heureusement que la prise en main de React Native est facile. L’une des autres difficultés était de comprendre tout le système de comptage de points du chibre. Je ne connaissais pas du tout ce système avant de réaliser ce travail. Pour la partie technique le moment on j’ai dû perdre le plus de temps était lors de la création de l’écran principale, c’est en effet un gros bloque de programmation et c’est la que j’ai pris le plus de temps.

Suite du projet

Pour la suite du projet, je vais devoir améliorer le fait qu’une partie se rafraichisse sans passer par un bouton. Il faudra aussi que j’améliore l’interface utilisateur car elle n’est pas du tout optimale pour pouvoir afficher ce genre de partie. De plus pour la partie programmation, il m’est arrivé de me répéter quelque fois au niveau du code, ce qui n’est pas du tout optimale pour l’application.

Annexes

Résumé du rapport du TPI / version succincte de la documentation

Sources – Bibliographie

*Liste des livres utilisés (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur)… Et de toutes les aides externes (noms)*

Journal de travail

## 

Manuel d'Installation

Manuel d'Utilisation

Archives du projet

*Media, … dans une fourre en plastique*