CHIBRE MANAGER

Table des matières

1 Analyse

1.1 Introduction

Arrivé au terme de mon CFC en tant qu'informaticien à l'école des Arches, j'ai dû effectuer un travail de fin d'année afin 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 réelle. Le comptage des pointes au Chibre n'est pas 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 que j'ai eu l'opportunité d'étudier durant mon stage.

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

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

Il se peut que certaines étapes ne soient pas nécessaires pour l'aboutissement de mon projet.

1.2 Organisation

Apprenti	Téo Sesti 079 963 04 97 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 Therisod 024 552 00 77 romain.therisod@wavemind.ch
Expert 1	Gilbert Gruaz gilbert.gruazduvaud.ch
Expert 2	Nicolas Borboen nicolas.borboen@epfl.ch

1.3 Cadre de travail

1.3.1 Wavemind

Ce travail aurait dû se faire dans les bureaux de l'entreprise Wavemind Sàrl, mais à cause de la pandémie du coronavirus, j'ai dû effectuer ce TPI 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.

Figure 1 : Logo Wavemind

1.3.2 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, de culture générale et de mathématiques. J'ai aussi eu la chance de pouvoir assister à des cours bloques durant mes deux années de stage pratique.

Figure 2 : Logo Ecole des Arches

1.4 Durée du travail

Mon projet de CFC informaticien se fait sur une durée 2 semaines, soit 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.

1.5 Planification initiale

Le document « Planification initiale » figure dans les fichiers annexe.

1.6 Choix des technologies

1.6.1 Web

1.6.1.1 Ruby on Rails

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

Figure 3 : Logo Ruby on Rails

1.6.2 Mobile

Le framework React Native m'a également été imposés pour ce projet. Etant donné que j'ai aussi appris ce langage durant ma formation, cette contrainte n'a pas été un obstacle dans l'élaboration de mon TPI.

1.6.2.1 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

Figure 4 : Logo React Native

1.6.2.2 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

Figure 5 : Logo Expo

1.7 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 requêtes API me retournent un tableau d'information, m'assurer que les restrictions de données marchent bien ou encore que les erreurs API soit gérés 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 décrirai en détail plus bas dans ce rapport. Pour résumé, je vais tester les différentes actions possibles sur des écrans différents en essayant même de fermer l'application et de la réouvrir afin de voir comment elle va réagir.

1.8 Environnement de travail

1.8.1 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

1.8.2 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.io
- Git
- Chrome

- Windows 10

1.9 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 recommandé.

7) Du côté mobile, l'application doit gérer proprement les erreurs du serveur ; aussi bien le cas où le serveur ne serait pas disponible que dans le cas où le serveur rendrait une erreur.

2 Conception

2.1 Structure de du serveur web

2.1.1 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 toutes prêtes et utiles 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 a un but précis, d'abord le **modèle** qui va communiquer avec la base de donnée avec les différentes méthodes définies 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**.

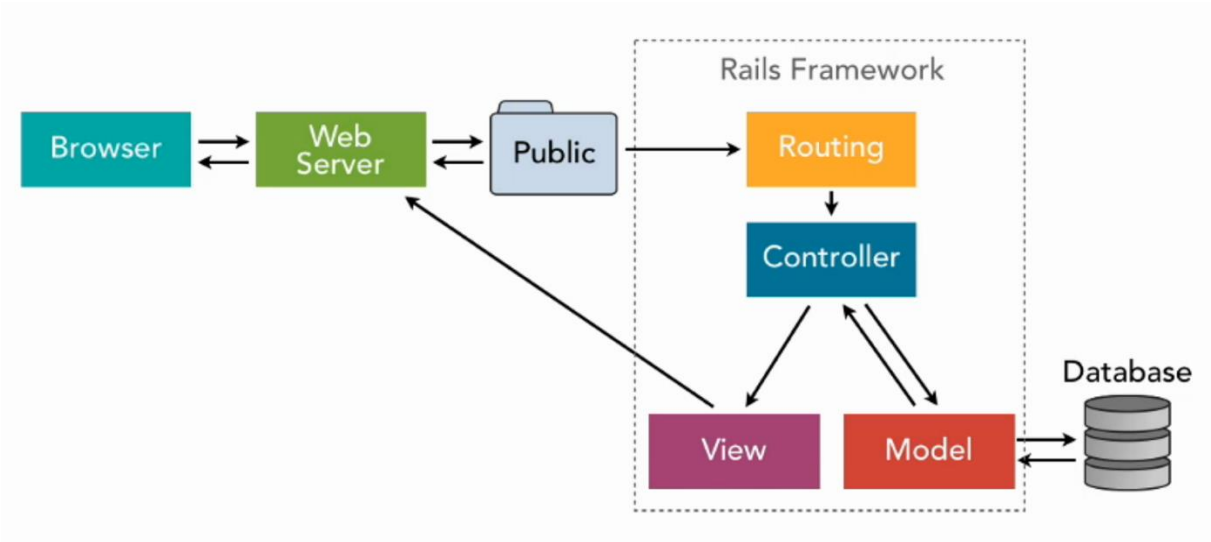

Figure 6 : Cheat sheet Rails

2.1.2 Base de données

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

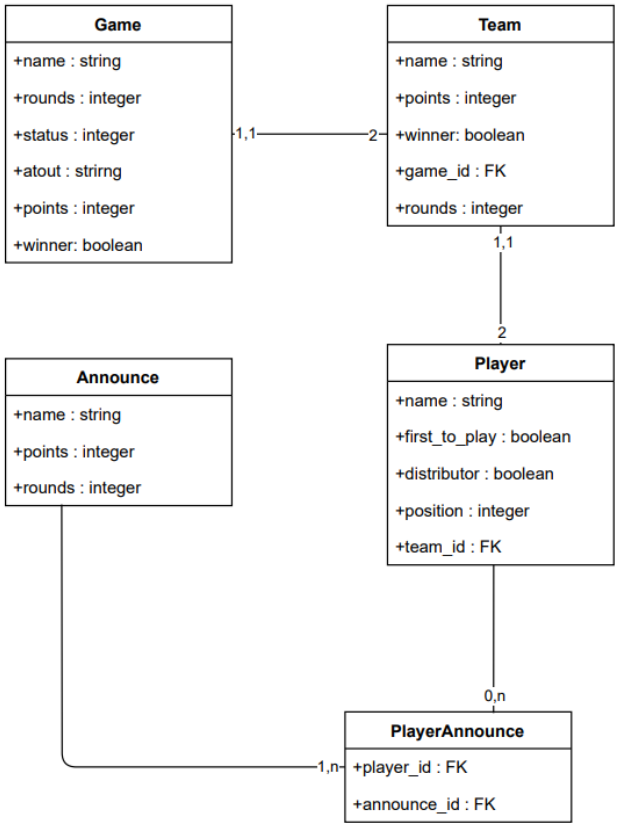

Figure 7 : Base de données SQL

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	Comporte toutes les équipes d'une partie. Permettre de définir le nom, les points et la partie à laquelle elle appartient.
Player	Comporte tous les joueurs d'une partie. Permet de définir le nom, le distributeur, le premier à jouer et la team à laquelle elle appartient.
PlayerAnnounce	Table intermédiaire qui permet de faire le lien des annonces par rapport au joueur et vice versa .
Announce	Regroupe toutes les annonces d'une partie. Permet de définir le nom, les points et les manches d'une annonce.

2.2 Structure de l'application mobile

2.2.1 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 composants déjà prêts 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.

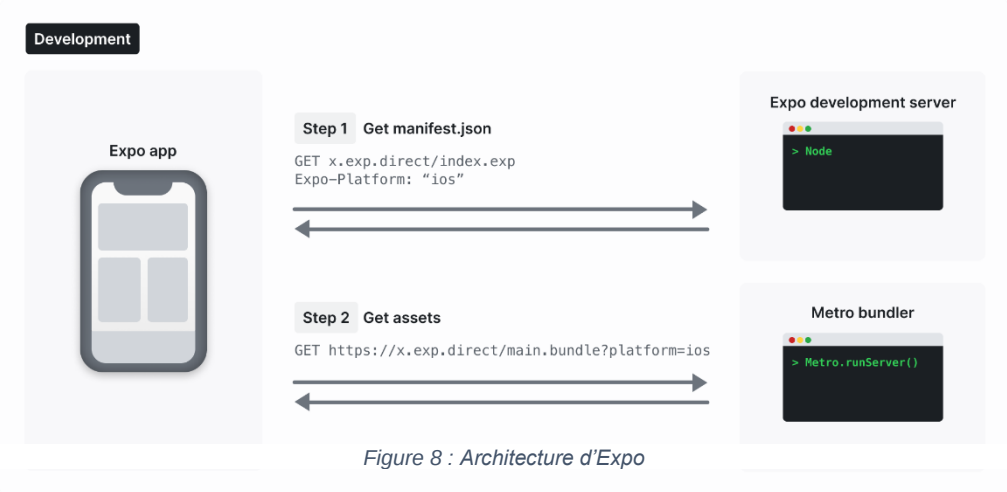

Figure 8 : Architecture d'Expo

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

2.2.2 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 « Bridge »
- Le coté Native de l'application

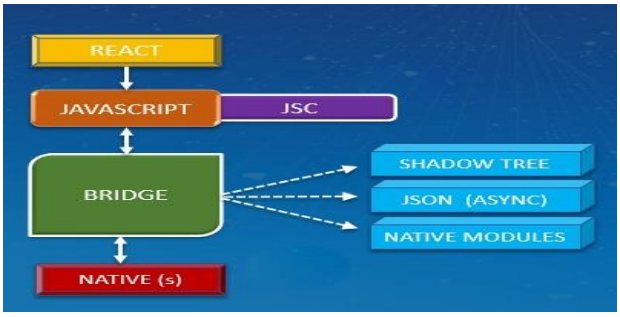

Figure 9 : Architecture React Native

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 stockées 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 nommer les 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

2.3 API

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

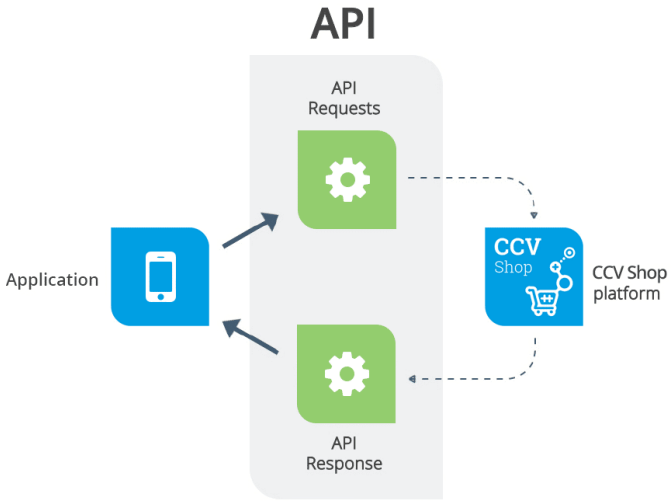

Figure 10 : Architecture API

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 API
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 générées 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.

2.4 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.

Figure 11 : Maquette 1

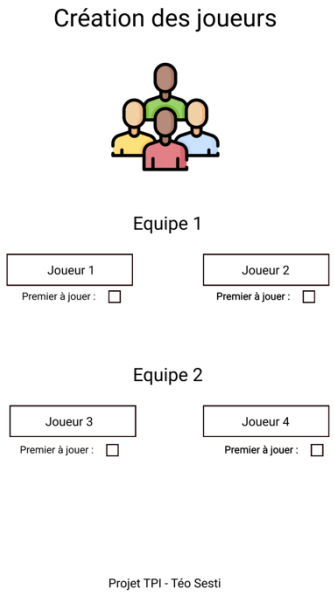

Figure 12 : Maquette 2

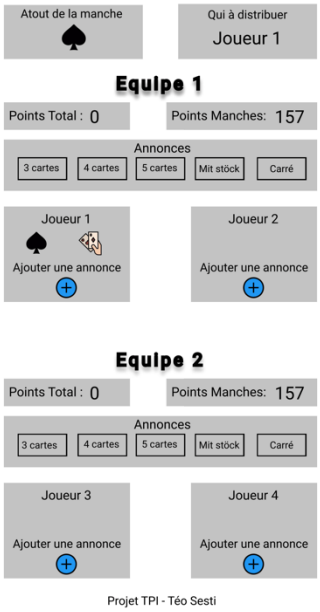

Figure 13 : Maquette 3

2.5 Schéma de navigation

2.5.1 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 vous y rendre.

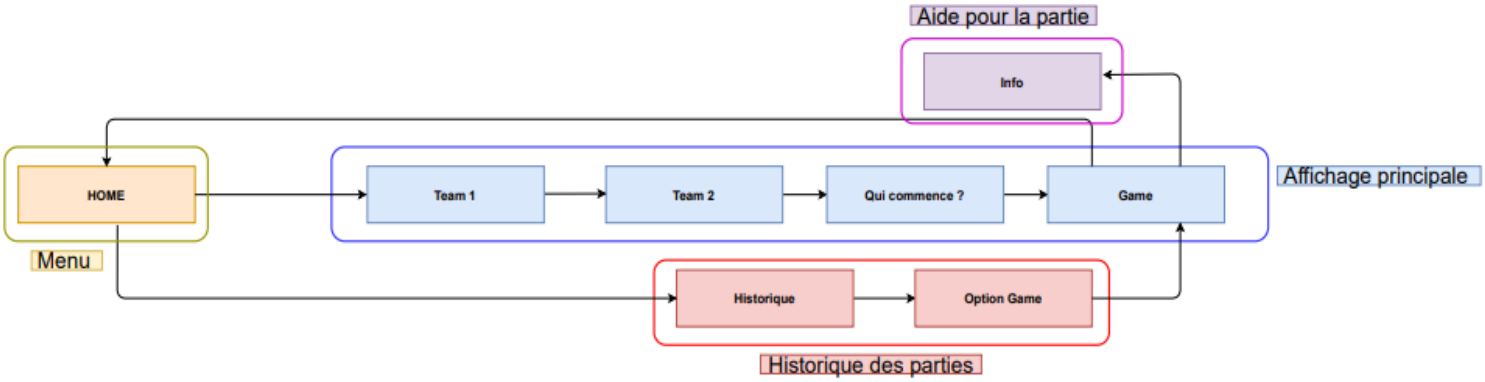

Figure 14 : Schéma de navigation

2.6 Risques techniques

J'ai beaucoup travaillé sur des sites web durant mon stage chez Wavemind. C'est seulement après 1 ans que je me suis mis au Ruby et seulement depuis quelques mois à React native. Je n'ai donc pas toutes les normes acquises ou encore tous les réflexes pour appliquer toutes les conventions pour 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 pensée de façon à ce qu'elle ne puisse être améliorée facilement par la suite.
J'avais quelques craintes de ne pas avoir les compétences nécessaires et j'ai dû aller m'informer sur certaines pratiques que je n'avais pas eu le temps de voir pour mon projet. Comme la mise à jour du State ou le **useEffect** que je ne maîtrise pas encore.

2.7 Planification réelle

Le document « Planification réelle » figure dans les fichiers annexe.

3 Réalisation

3.1 Ruby on Rails

3.1.1 Configuration

3.1.1.1 Serveur

- Serveur : Heroku
- OS : Ubuntu
- DB : PostgreSQL

3.1.1.2 Application

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

3.1.1.3 Fonctionnalités

- Game
  - Création
  - Modification
  - Listing
  - Suppression
- Announce
  - Création
  - Suppression
- API

3.1.2 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.

3.1.2.1 Les Migrations

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

```
rails db:migrate
```

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

```ruby
class CreateTeam < ActiveRecord::Migration[6.0]
  def change
    create_table :teams do |t|
      t.string :name
      t.integer :points, default: 0
      t.boolean :winner, default: false

      t.timestamps
    end
    add_reference :teams, :game, foreign_key: true
  end
end
```

Figure 15 : Migration rails

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 »

3.1.2.2 Les Modèles

Une fois les migrations faites, il faut créer des modèles qui vont 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.

3.1.3 Les Routes

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

3.1.3.1 API

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

```ruby
# Crée les différentes routes (URL) pour pouvoir faire des requêtes API dessus
Rails.application.routes.draw do
  namespace name :api, defaults: {format: :json} do
    namespace name :v1 do
      resources :announces, only: [:index, :destroy]
      resources :games, only: [:index, :create, :show, :update, :destroy]
      resources :teams, only: [:index, :create]
      resources :players, only: [:index, :create]
      resources :player_announces, only: [:index, :create]
    end
  end
end
```

Figure 16 : Routes Rails

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.

3.1.4 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 ».

```ruby
# Crée une équipe
def create
  # Crée une partie
  @last_game = Game.last

  # Crée les deux équipes
  @team = Team.new(name: params[:teamname])
  @team.update(game_id: @last_game.id)

  #Crée les quatres joueurs
  @player1 = Player.new(name: params[:player1])
  @player1.update(team_id: @team.id)
  @player2 = Player.new(name: params[:player2])
  @player2.update(team_id: @team.id)

  if @team.save && @player1.save && @player2.save
    @team.update(game_id: @last_game.id)
    @player1.update(team_id: @team.id)
    @player2.update(team_id: @team.id)
    render json: { success: true, team: @team, player1: @player1, player2: @player2}
  else
    render json: { success: false, team: @team, player1: @player1, player2: @player2}
  end
end
```

Figure 17 : Contrôleurs Rails

Le contrôleur va d'abord créer 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.

3.1.5 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.

```ruby
# Extrait les données de la valeur @game pour les mettre dans un tableau JSON.
json.array! @games do |game|
  json.extract! game, :id, :name, :status, :rounds, :created_at

  json.teams game.teams do |team|
    json.extract! team, :id, :name, :points

    json.player team.players do |player|
      json.extract! player, :id, :name
    end
  end
end
```

Figure 18 : Vues Rails

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

3.2 React Native

3.2.1 Configuration

3.2.1.1 Application

- Framework : React Native
- URL APK : http://gofile.me/6NSuo/VTcrcfUQm

3.2.1.2 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

3.2.2 Les Vues

3.2.2.1 Menu d'accueil

Lorsque l'utilisateur va commencer à utiliser l'application Chibre Manager, il va forcément arriver sur ce 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.
Étant donné que mon application compte les points d'un jeu de cartes, j'ai donc mis naturellement une image de ce genre.

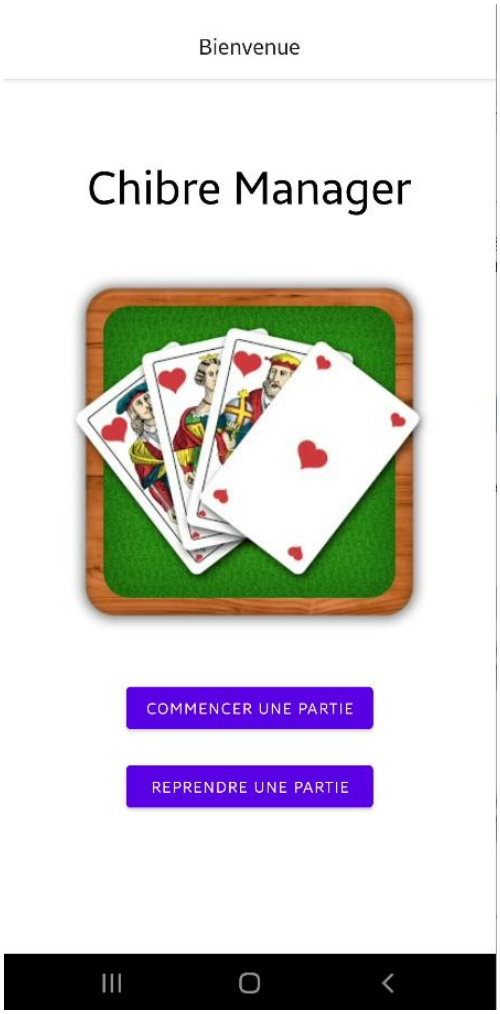

Figure 20 : Ecran d'accueil

```
<View style={styles.buttonSingleContainer}>
  <Button
    mode="contained"
    onPress={() => {
      navigation.navigate('CreateTeam1')
    }}
  >
    Commencer une partie
  </Button>
```

Figure 19 : Code bouton

Le bouton « Commencer la partie » va amener l'utilisateur sur un formulaire pour pouvoir créer tous les éléments afin de débuter une partie. Le deuxième bouton « Reprendre une partie » va lui nous faire changer d'écran pour aller sur une liste de parties déjà créées auparavant.

3.2.2.2 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).

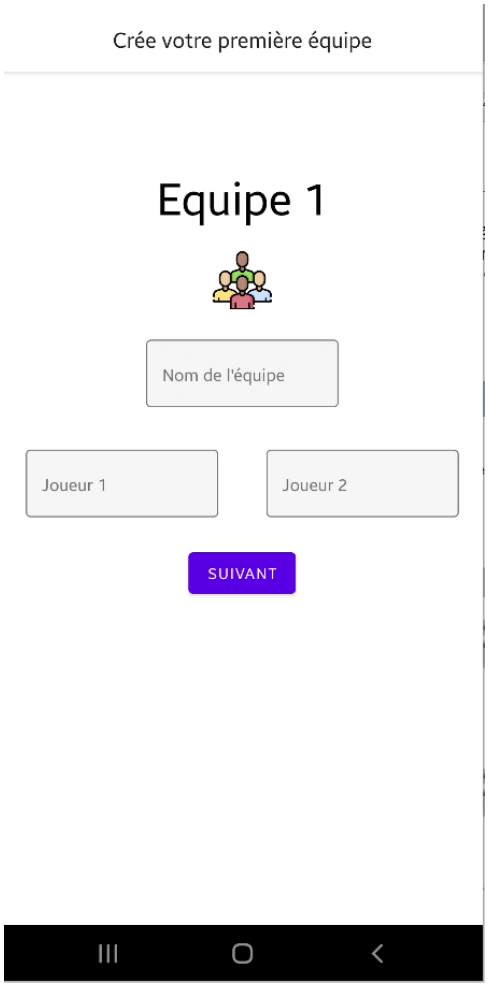

Figure 22 : Ecran équipe 1

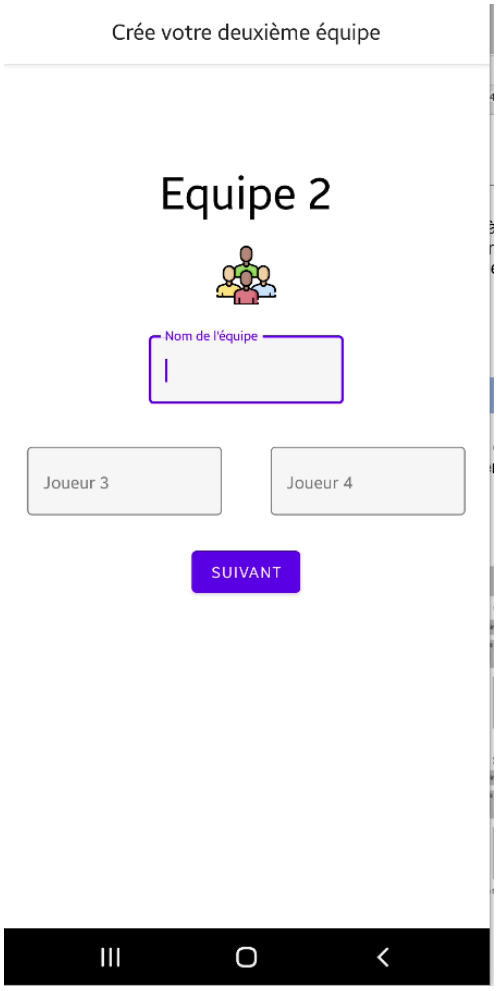

Figure 21 : Ecran équipe 2

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 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. L'utilisation des deux méthodes création des joueur et d'équipe est utilisé dans cette vue.

```
export default function CreateFirstTeamScreen() {
  return (
    <View style={styles.container}>
      <CreateTeam
        title="Equipe 1"
        icone={require('../../../assets/img/team.png')}
        namePlayerOne="Joueur 1"
        namePlayerTwo="Joueur 2"
        routeNavigation="CreateTeam2"
        currentScreen="Team1"
      />
    </View>
  )
}
```

Figure 24 : Composant « CreateTeam » 2

```
export default function CreateSecondTeam() {
  return (
    <View style={styles.container}>
      <CreateTeam
        title="Equipe 2"
        icone={require('../../../assets/img/team.png')}
        namePlayerOne="Joueur 3"
        namePlayerTwo="Joueur 4"
        routeNavigation="WhoStart"
      />
    </View>
  )
}
```

Figure 23 : Composant « CreateTeam » 2

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

```
//Fonction qui permet de passer a l'écran suivant et de set des deux équipes dans team1 et team 2
const nextScreen = () => {
  if (props.currentScreen == 'Team1') {
    setTeam1({...team1, team1: team, player1: playerOne, player2: playerTwo});
  } else {
    setTeam2({...team2, team2: team, player3: playerOne, player4: playerTwo});
  }
  if (playerOne !== '' && playerTwo !== '' && team !== '')
    navigation.navigate(props.routeNavigation);
};
```

Figure 25 : Fonction "nextScreen"

3.2.2.3 Qui commence et combien de points pour la partie ?

Arrivé sur cet écran l'utilisateur va devoir choisir le joueur qui devra 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.

```
//Information envoyer à setData
const dataGame = () => {
  formData.append( name: 'team_name1', team1.team1);
  formData.append( name: 'team_name2', team2.team2);
  formData.append( name: 'player1', team1.player1);
  formData.append( name: 'player2', team1.player2);
  formData.append( name: 'player3', team2.player3);
  formData.append( name: 'player4', team2.player4);
  formData.append( name: 'game_points', selectedValue);
  formData.append( name: 'first_to_play', checked);
  setData(formData)
}
```

Figure 27 : Constante « dataGame »

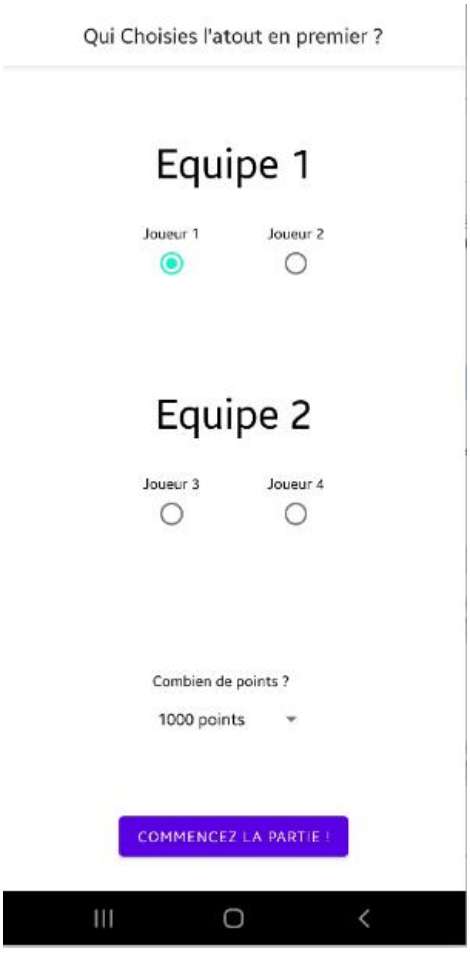

Figure 26 : Ecran pour le premier joueur

Une fois toutes les options misent 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. L'utilisation de la méthode création de partie et savoir qui choisit l'atout sont utilisé dans cette partie du code.

3.2.2.4 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 initialiser toutes les données pour débuter une partie. Ensuite l'utilisateur aura la possibilité d'ajouter des annonces qu'il pourra attribuer à un joueur d'une équipe.
Une fois toutes les annonces mises, l'utilisateur n'aura plus qu'à rentrer les points de la manche dans le champ prévu à cet effet. Une fois toutes les actions faites, l'utilisateur devra passer à la manche suivante en cliquant sur la flèche qui va additionner tous les points des équipes pour les mettre en points totaux

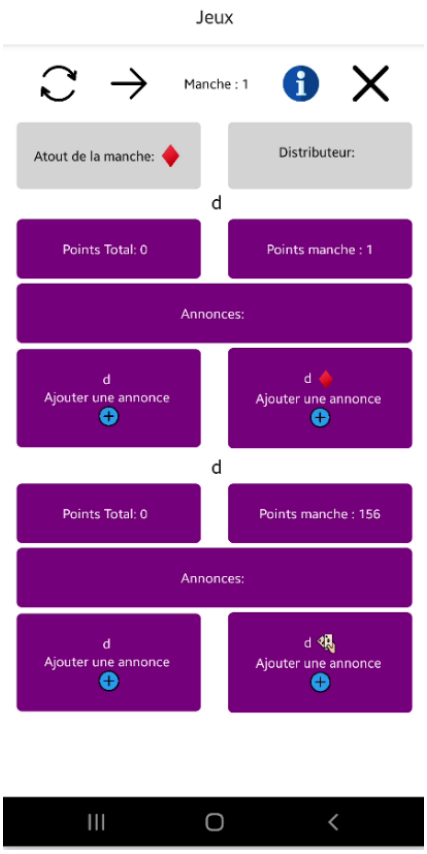

Figure 28 : Ecran de la partie

L'action 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 définies 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 définis par l'utilisateur. L'icône à côté de la carte de celle du joueur signifie que c'est à lui de distribuer tandis que l'icône en forme d'atout à côté du joueur signifie que c'est à lui de choisir l'atout pour la manche. L'utilisation des méthodes, comptage de points, changement d'atout, attribuer une annonce a un joueur ou encore définir quelle équipe à gagner sont utilisées.

3.2.2.5 Lorsqu'une équipe gagne

Cet affichage apparaît quand une des deux équipes a dépassé 1000 ou 1500 points. Le nom de l'équipe gagnante est affiché et un bouton apparaît pour revenir au menu principal de l'application.

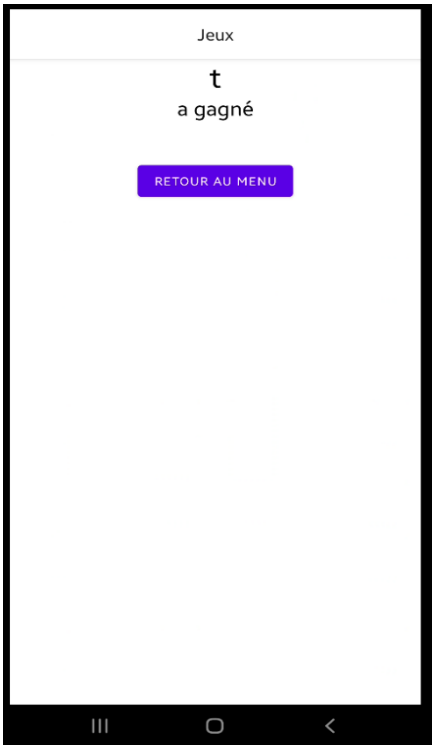

Figure 29 : Ecran de l'équipe gagnante

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.

```
if (gameData.teams[0].winner) {
  return (
    <View style={styles.container}>
      <View style={styles.containerText}>
        <Text style={styles.winnerTitle}>{gameData.teams[0].name}</Text>
        <Text style={styles.title}>a gagné</Text>
      </View>
      <View style={styles.containerButton}>
        <Button
          mode={"contained"}
          onPress={() => {
            {
              updateStatusGame();
              navigation.navigate("Home")
            }
          }}
        >
          Retour au menu
        </Button>
      </View>
    </View>
  )
}
```

Figure 30 : Retourne une vue

3.2.2.6 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.

Figure 31 : Liste des parties

Toutes formes de listes sous React Native sont gérées grâce au composant Flatlist qui permet de réaliser facilement une liste selon un tableau d'objets. 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 générée grâce à une requête API avec la méthode 'GET' qui va nous renvoyer toutes les parties crées dans l'application.

3.2.2.7 Option d'une partie

Cette vue nous répertorie toute la partie conçue 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.

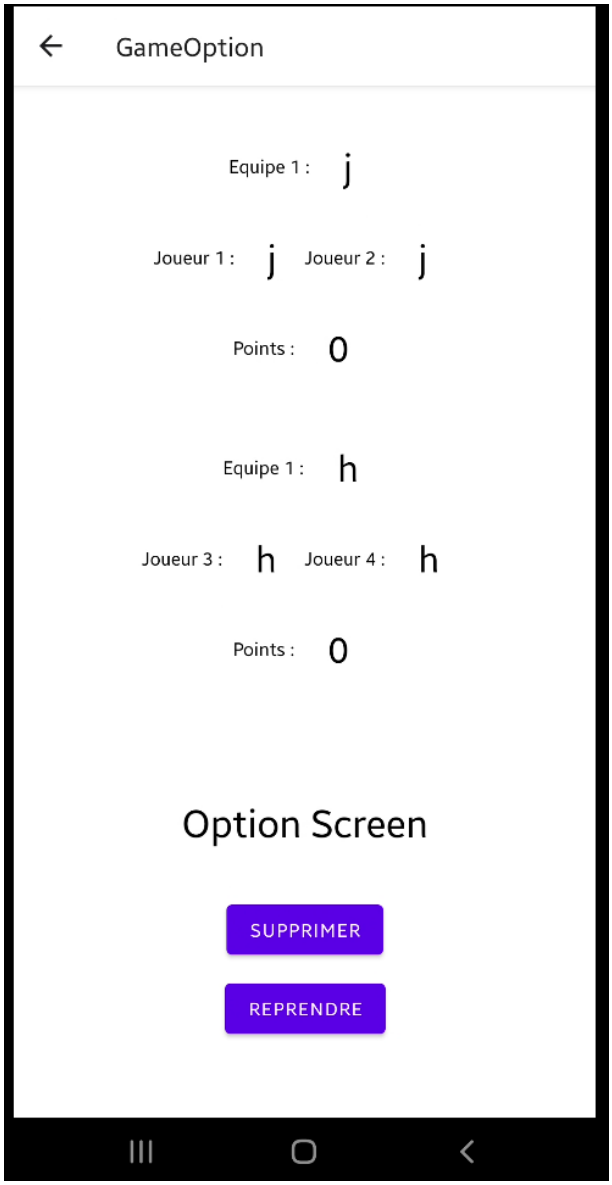

Figure 32 : Ecran d'option d'une partie

Cette vue nous répertorie toute la partie élaborée 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.

3.2.2.8 Atout

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

Figure 33 : Ecran des atouts

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.

3.2.2.9 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.

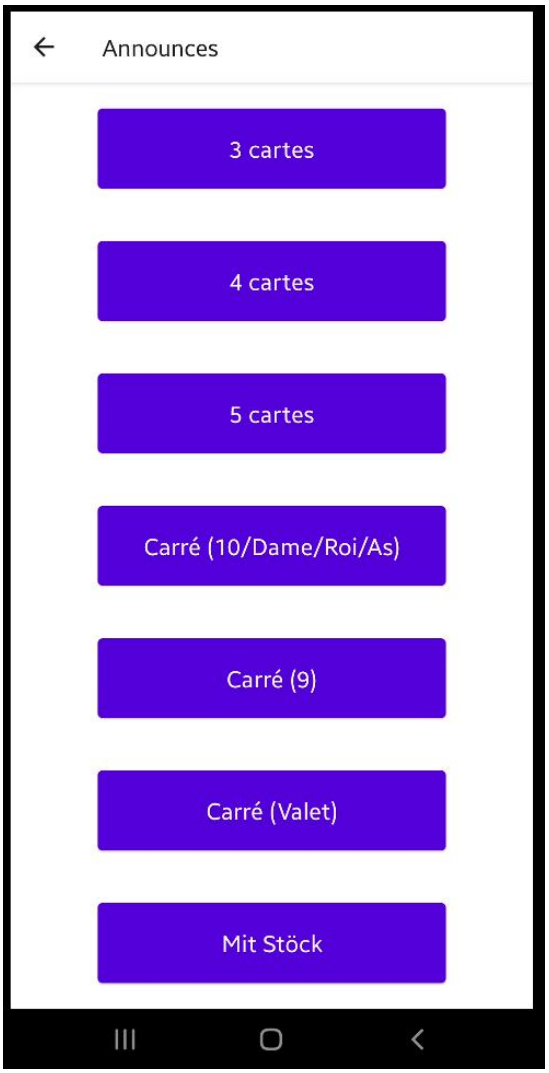

Figure 34 : Ecran des annonces

Pour pouvoir afficher ses différents éléments, j'ai créé un fichier AnnounceScreen qui 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.

3.2.2.10 Les Infos

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

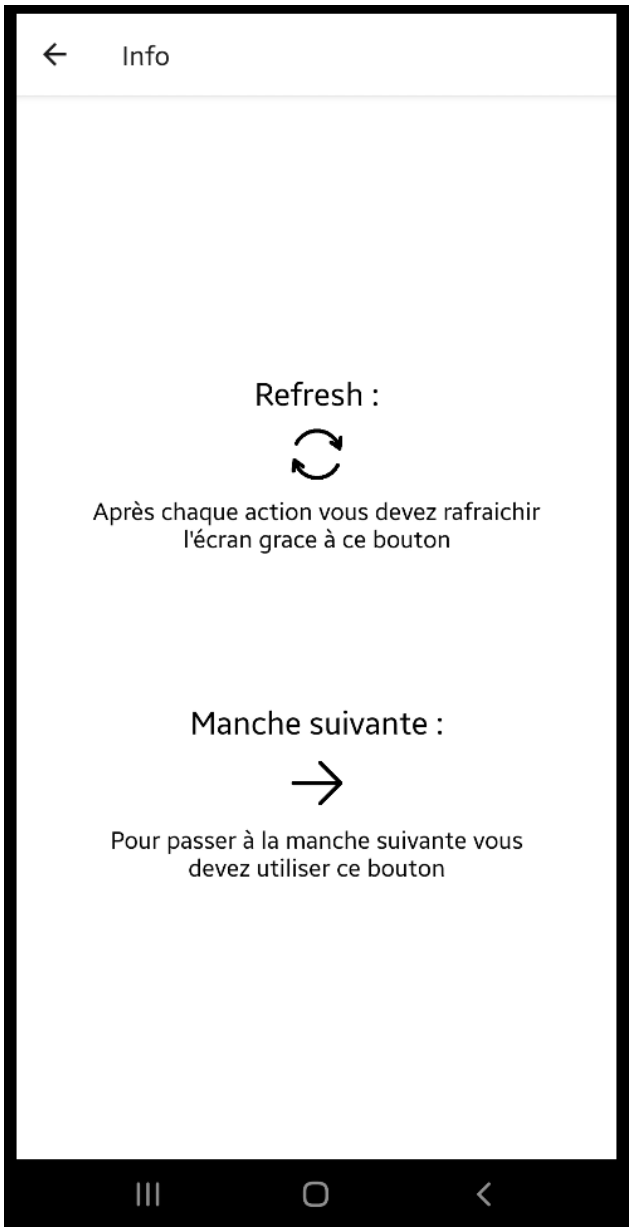

Figure 35 : Ecran d'information

3.3 API

3.3.1 Comment marche mon API

3.3.1.1 Mobile

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

```
//Fonction qui permet de retourner une partie
export const GetGame = async (id) => {
  const method = 'GET'
  const url = `${host}/api/v1/games/${id}`;
  let response = await _fetch(url, method);
  console.log(response)
  return response
};
```

Figure 36 : Méthode API

3.3.1.2 Serveur Web

Grâce à l'id reçu, le serveur web va pouvoir rechercher la partie concernée et ainsi retourner toute l'information définie dans la vue à l'application. Voilà la méthode « set_game » qui fera ce que j'ai expliqué plus haut.

```
# Trouve une partie selon une id
def set_game
  @game = Game.find(params[:id])
end
```

Figure 37 : Méthode Rails

Voilà à quoi ressemble la vue sur le serveur web, c'est ici que l'on va définir les différents champs à renvoyer. Comme décrit sur cette image l'on peut voir que le tableau 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 joueurs liés à cette partie.

```
1  json.extract! @game, :id, :name, :status, :rounds, :atout, :points, :winner
2  json.teams @game.teams do |team|
3    json.extract! team, :id, :name, :points, :winner
4
5    json.player team.players do |player|
6      json.extract! player, :id, :name, :first_to_play, :distributor
7
8      json.announce player.announces do |announce|
9        json.extract! announce, :id, :name, :points, :rounds
10     end
11   end
12 end
13
```

Figure 38 : Tableau JSON Rails

3.4 Structure des fichiers

Dans cette partie du dossier je vais vous expliquer comment est organisée toute la structure de mon repo Git Hub.

- Le logiciel se situe dans mon repos GitHub, à l'adresse suivante : https://github.com/nours33/Chibre-manager/tree/maste.

- Le dossier backend_api contient tous les fichiers concernant le serveur web et la base de données PostgreSQL.

    - App : c'est ici que tous les fichiers principaux de l'application sont stockés.
        - Les contrôleurs : permettent de communiquer entre le modèle et la vue
        - Les modèles : permettent de communiquer avec la base de données
        - Les vues : permettent d'afficher des informations sur un écran ou dans notre cas renvoyé un tableau en format JSON pour notre API.

- Le dossier frontend_mobile contient tous les fichiers concernant l'application Mobile.

    - App.js : C'est là qu'est écrit toute l'application mobile
    - Src :
        - Common : permet de définir des constantes que l'on peut utiliser partout dans notre application mobile.
        - Components : définit les différents composants de notre application
        - Screens : c'est l'affichage qui apparaît sur notre mobile

3.5 Tests

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

3.5.1 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 fois 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 »

3.5.2 Application mobile

3.5.2.1 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

3.5.2.2 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

3.5.2.3 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

3.5.2.4 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

3.5.2.5 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

3.5.2.6 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

3.5.2.7 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

3.6 Liste des documents fournis

Vous trouverez en annexe les fichiers suivants :

- Planification détaillée.pdf
- Planification initiale.pdf
- Planification réelle.pdf
- Résumé pour Chibre Manager.pdf
- Journal de travail.pdf
- API.txt

4 Conclusions

4.1 Conclusion personnelle

Ce travail m'a permis d'enrichir mes connaissances en technologie Rails et React Native. C'est durant ces deux ans de stage que j'ai travaillé avec Rails et React Native pour la première fois. J'ai eu également la possibilité de travailler avec Wordpress, malheureusement je n'ai pas eu l'opportunité de pouvoir appliquer ce que j'avais assimilé pour ce projet.
Mais grâce à cette étude j'ai réellement pu réaliser un challenge car, cela ne faisait pas longtemps que j'avais appris le langage React native.

Je ne connaissais pas du tout le Chibre, cela m'a donc motivé à l'apprendre et à en connaitre toutes les règles nécessaires pour pouvoir y jouer. Cela n'a pas été facile n'étant pas un grand amateur de jeux de cartes, mais j'avoue avoir éprouvé un réel plaisir à disputer quelques parties avec des amis.

Finalement ce projet servira sûrement à tous les amateurs de Chibre et je te tiens à remercier Wavemind pour leur soutien durant ces 2 ans de stage

4.2 Bilan des objectifs

Repo GitHub : Il est vrai que tous mes messages commit ne sont pas tous explicites 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 l'application. Il me semble que c'est un choix judicieux concernant cette technologie. Il est vrai que j'ai dû 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 utilisées 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

En règle générale l'application fait ce que le cahier des charges demande. Toutefois je reconnais que l'application n'est pas optimale. Le problème réside dans 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.

.

4.3 Difficultés rencontrées

Les difficultés que j'ai pu rencontrer sont celles avec React Native. React Native est un langage informatique avec lequel je ne suis pas du tout habitué. J'ai dû revoir certaines bases pour avancer dans mon projet. Il m'a fallu apprendre toute la partie des requête API car c'était une des premières 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 perdu le plus de temps était lors de la création de l'écran principal, c'est en effet un gros bloque de programmation et c'est cela qui m'a demandé le plus de temps.

4.4 Suite du projet

Pour la suite du projet, je vais devoir corriger 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.

5 Annexes

5.1 Résumé du rapport du TPI

Le document « Résumé de Chibre Manager » figure dans les fichiers annexe.

5.2 Manuel d'Installation

5.2.1.1 Application mobile

Vous trouverez à cette adresse la guide d'installation pour l'application mobile React Native.

https://github.com/nours33/Chibre-manager/blob/master/front_mobile/chibre-manager/Readme.md

5.2.1.2 Serveur web

Vous trouverez à cette adresse le guide d'installation pour la mise en place du serveur web.

https://github.com/nours33/Chibre-manager/edit/master/backend_api/chibre-manager/README.md

5.3 Manuel d'Utilisation

5.3.1.1 Application mobile

Vous trouverez à cette adresse le guide d'utilisation pour l'application mobile React Native.

https://github.com/nours33/Chibre-manager/blob/master/front_mobile/chibre-manager/Readme.md

5.3.1.2 Serveur web

Vous trouverez à cette adresse le guide d'utilisation pour le serveur web.

https://github.com/nours33/Chibre-manager/edit/master/backend_api/chibre-manager/README.md

5.3.1.3 Chibre manager

Vous trouverez à cette adresse le guide d'utilisation pour l'application Chibre Manager.

https://github.com/nours33/Chibre-manager/blob/master/Readme.md

6 Table des illustrations

7 Glossaire

Termes	Définitions
TPI	Le TPI (Travaux Pratiques d'informatique) est un travail à présenter pour confirme la fin de ses études de CFC informaticien.
Ruby	Ruby est un langage informatique libre, il est orienté objet.
Ruby on Rails	Est un framework web écrit en Ruby.
Expo	Un système qui permet aux utilisateurs de créer des applications mobiles pour IOS et Android
Framework	Ensemble cohérent de bibliothèque logicielle qui va nous aider à réaliser plus facilement certaines tâches
Open Source	Est un code mis a disposition gratuitement pour une éventuelle modification et redistribution.
API	Application Programming Interface ou interface de programmation applicative en français est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.
Git	Git est un logiciel de gestion de versions.
ERD	Diagramme entité-association ou en anglais « Entity-Relationship Diagram »
PostMan	PostMan est un logiciel permettant de faire des requêtes API sur une adresse url.
NotePad	NotePad est un logiciel pour traiter du texte.
Swagger.io	Swagger est une application web permettant l'élaboration de document interactif concernant les API.
Draw.io	Draw est un logiciel web permettant de créé des graphiques ou dessins facilement.
RubyMine	RubyMine est un IDE pour ruby.
SQL	Le SQL est un langage de définitions des données permettent de créé et de modifier des données
GET	Protocole internet qui permet de communiquer différentes informations.
POST	Protocole internet qui permet de communiquer différentes informations.
YAML	YAML est un format de représentation de données par sérialisation Unicode.
useEffect	useEffect est un hook dans React Native permettant de déclencher certaine action a certain moment.
ActiveRecord	ActiveRecord est une approche pour lire les données d'une base de données.
Placeholder	Traduction français « espace réservé ».

8 Sources

https://reactnative.dev/
https://callstack.github.io/react-native-paper/index.html
https://guides.rubyonrails.org/
https://fr.wikipedia.org/wiki/Framework
https://fr.wikipedia.org/wiki/React_Native
https://fr.wikipedia.org/wiki/Ruby_on_Rails
https://fr.wikipedia.org/wiki/Ruby
https://www.chibre.ch/forum/viewtopic.php?t=638
https://www.jeuxdecartes.net/jeux-cartes/chibre/