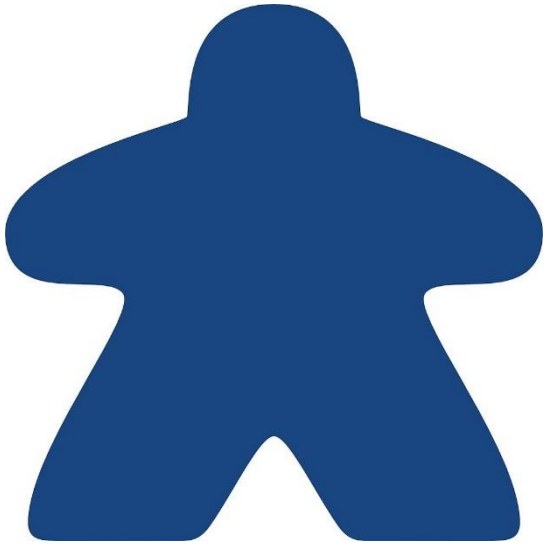


Board Club

Projet de fin de formation

CODING ACADEMY 06/2020

Sébastien CHOUBRAC

Vincent CONTAMINE

Martin HEWITT

SOMMAIRE

1 – Présentation du projet

1.1 - Qu'est-ce que Board Club ?

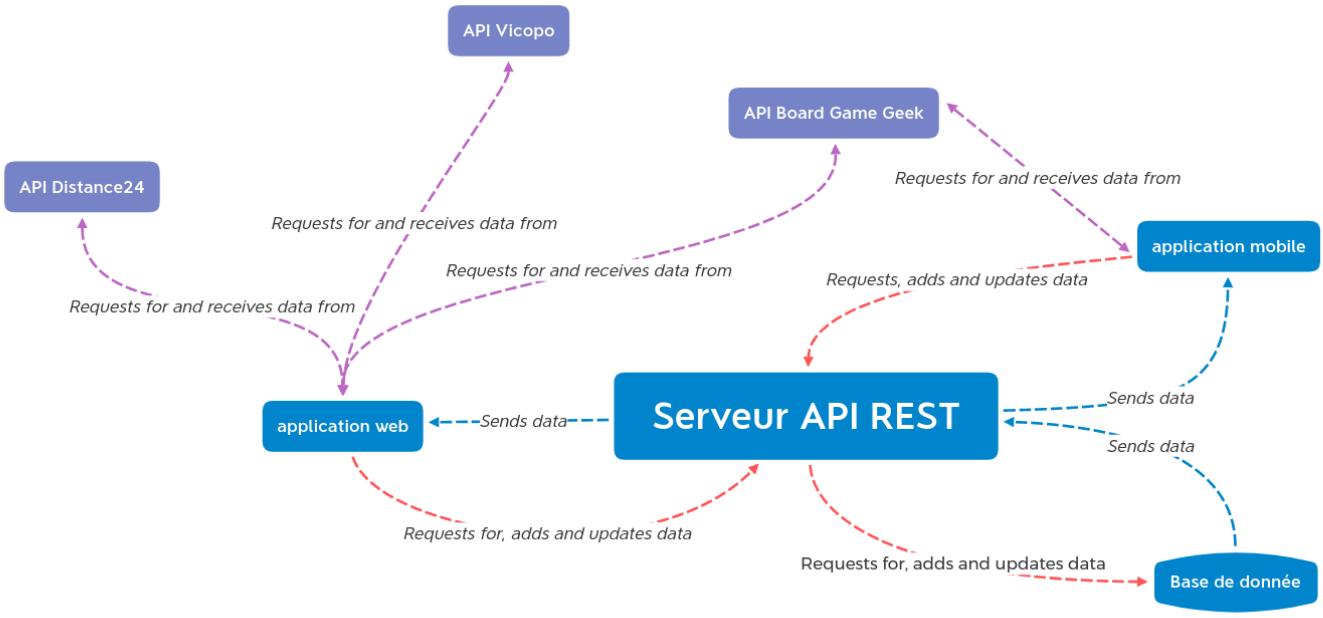
Board Club est un ensemble site et application mobile permettant aux passionnés de jeux de société d'organiser et de trouver facilement des parties de leur passe-temps favori.

Le site propose la visualisation rapide sur sa page d'accueil de toutes les futures parties prévues ; mais offre également la possibilité de ne voir que les parties organisées ou jouées par ses utilisateurs favoris, celles portant sur ses jeux préférés ainsi que les parties les plus proches de chez eux.

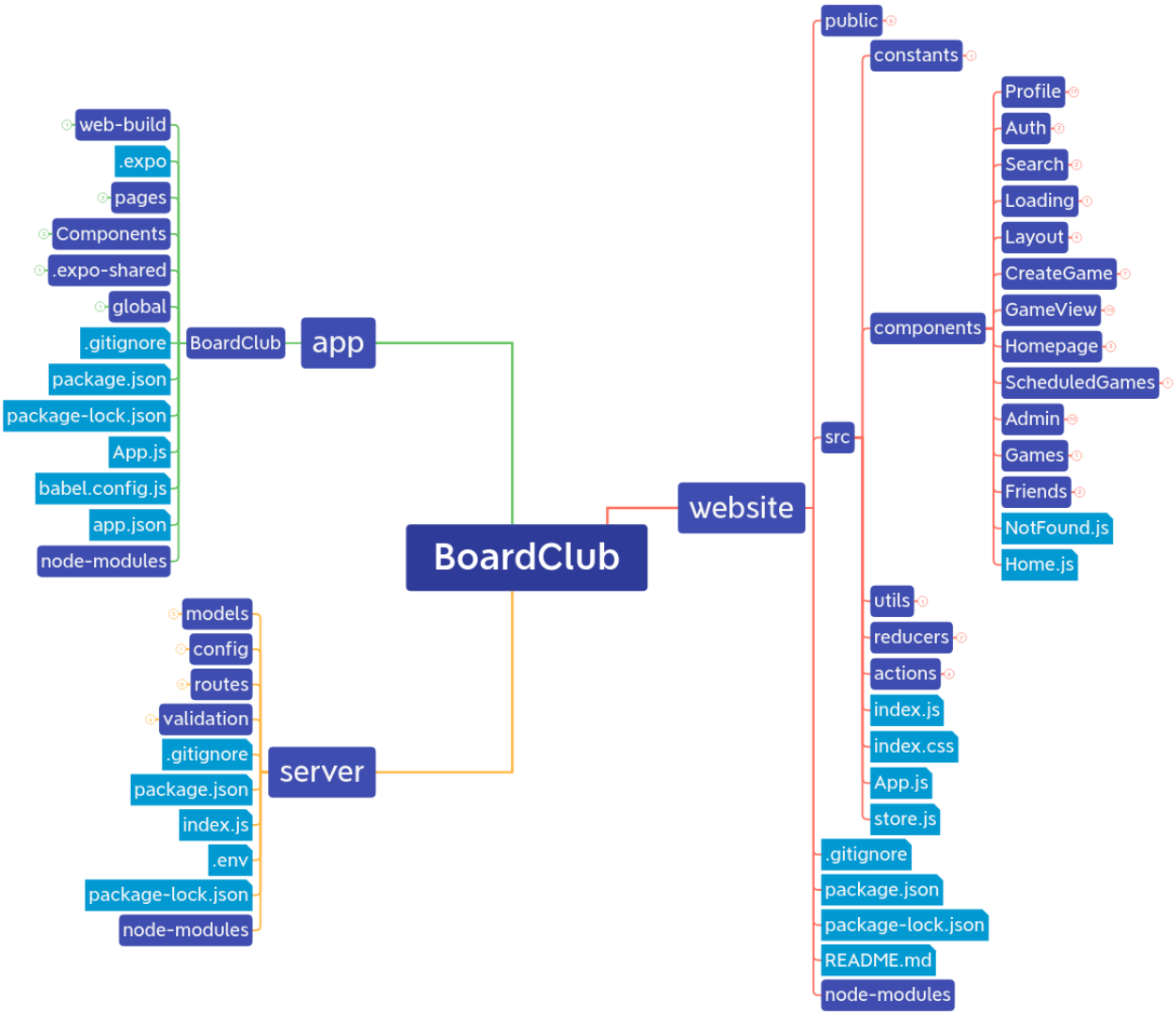
1.2 - Technologies utilisées

Ce projet utilise les technologies du stack MERN : MongoDB, Express React et Node.js ; ainsi que React Native pour l’application mobile. Il sera donc nécessaire d’être familier avec ces technologies pour travailler sur ce projet.

1.3 - Architecture du site



1.4 - Arborescence des fichiers



2 – Installation du projet

2.1 - Prérequis

Pour installer et travailler sur Board Club, vous pouvez utiliser le système d'exploitation de votre choix.

Afin de pouvoir développer autour de notre projet, vous aurez cependant besoin d'installer au préalable les logiciels suivants :

- Git :

Vous aurez besoin de Git, un logiciel de gestion de versions qui vous permettra de récupérer le projet. Pour installer Git, vous pouvez aller directement sur la page officielle du logiciel (https://git-scm.com/downloads).

- Node.js :

Allez sur la page officielle de Node (https://nodejs.org/en/download/) afin d'installer à la fois Node.js qui vous permettra d'exécuter du JavaScript, mais également npm, le gestionnaire de paquets officiel de Node.

- MongoDB :

Installez également MongoDB (https://docs.mongodb.com/manual/installation/) afin de pouvoir gérer la base de données du projet.

2.2 - Installation

Il faudra d'abord récupérer l'ensemble des fichiers sur le Github du projet. Pour cela, ouvrez le terminal et positionnez-vous dans le dossier où vous souhaitez installer le projet puis rentrez la commande suivante :

```
git clone https://github.com/mrtnhwtt/BoardClub.git
```

Pour plus de facilité dans la suite de l'explication, nous considérerons le dossier où se trouve le projet comme étant */BoardClub* .

Une fois le projet récupéré, il vous faudra installer toutes les dépendances utilisées en exécutant dans les dossiers */BoardClub/server*, */BoardClub/website* et */BoardClub/app* la commande :

```
npm install
```

Il est possible si vous le souhaitez d'installer une base de données toute prête avec des utilisateurs, parties et commentaires afin de tester rapidement le fonctionnement du site et de l'application. Pour cela, rendez-vous dans le dossier */BoardClub/Documentation/DB_Test* et entrez les commandes suivantes :

```
mongoimport --db boardclub --collection users < users.json
```

```
mongoimport --db boardclub --collection games < games.json
```

```
mongoimport --db boardclub --collection comments < comments.json
```

Attention à ce que votre serveur Mongo soit bien démarré au préalable, suivant le type d'installation choisi il est possible qu'il se

lance automatiquement ou qu'il faille le lancer manuellement (voir https://docs.mongodb.com/manual/tutorial/ pour plus d'informations).

Tous les utilisateurs présents dans la base de données de test ont le même mot de passe "testtest" ; et un utilisateur nommé "Admin" a tous les droits pour accéder à la page d'admnistration du site.

Pour accéder à l'application mobile vous aurez besoin d'installer au préalable un émulateur Android.

Vous pouvez obtenir un émulateur Android avec l'environnement de développement Android : Android Studio. (cliquez ici pour installer Android Studio). Une fois installé, suivez cette procédure pour créer une machine virtuelle d'un téléphone Android avec Android Pie 9.0.

Alternativement vous pourrez utiliser votre propre téléphone mobile Android pour tester l'application avec l'application Expo sur Android. Il est cependant conseillé d'utiliser une machine virtuelle.

3 – Lancement du projet

3.1 - Serveur

Avant toute chose, assurez-vous que votre serveur Mongo est bien démarré avec la procédure appropriée pour votre OS sur le port 27017. Si vous préférez le lancer sur un port différent, vous pouvez spécifier le port voulu dans le fichier */BoardClub/server/.env* .

Pour lancer le serveur gérant toutes les requêtes à la base de données, il suffit maintenant d'aller dans le dossier */BoardClub/server* à partir du terminal et d'exécuter la commande suivante :

```
npm start
```

Le serveur se lancera alors sur le port 5000.

Il sera également nécessaire pour la bonne communication avec les API externes utilisées par le projet de se rendre dans le dossier */BoardClub/server/node_modules/cors-anywhere* et d'exécuter la ligne de commande suivante :

```
node server.js
```

CORS Anywhere s'exécutera alors à l'adresse 0.0.0.0.8080. Si ce n'est pas le cas, vous pouvez modifier manuellement le port configuré dans le fichier server.js.

3.2 - Site

Une fois le serveur lancé, il vous suffit d'aller dans le dossier /BoardClub/website et d'exécuter la commande suivante :

```
npm start
```

Le site se lancera automatiquement dans votre navigateur sur le port 3000.

3.3 - Application mobile

Pour lancer l'application mobile, rendez-vous sur */BoardClub/app/BoardClub* et exécutez la commande suivante dans le terminal :

```
expo start
```

Une fois le chargement terminé, une page web s'ouvrira dans votre navigateur et des instructions apparaitront dans le terminal. Si vous utilisez une machine virtuelle, tapez "a" dans le terminal ou cliquez sur "Run on Android device/emulator" une fois le code QR apparu dans la page web. Si vous utilisez votre mobile avec l'application Expo, scannez le code QR dans le terminal ou la page web.

Si c'est votre premier lancement d'Expo sur votre machine virtuelle Android, l'application Expo devra d'abord être installée ; cela se fera automatiquement. Une fois l'application Expo lancée le projet chargera puis vous pourrez accéder à l'application mobile.

4 – Documentation Détaillée

4.1 - Serveur

La partie serveur de Board Club a été réalisée en utilisant :

- Node.js pour exécuter le JavaScript
- Express.js en tant que framework nous permettant de développer notre serveur.
- MongoDB pour gérer notre base de données.
- Mongoose pour définir des objets avec un schéma basés sur un document MongoDB.

Trois schémas (qu'on peut retrouver dans le dossier */models*) ont été créés pour les trois collections de notre base de données.

User :

Lié à la collection *users*, ce schéma contient :

- login : de type String et obligatoirement unique.
- email : de type String et obligatoirement unique.
- password : de type String, hashé avec bcrypt.
- inscriptionDate : de type Date et généré automatiquement avec la date du jour à la création du compte de l'utilisateur.
- avatar : de type String et vide par défaut, pourra par la suite contenir l'url de l'avatar de l'utilisateur.
- city : de type String et vide par défaut.
- zipcode : de type String et vide par défaut.
- followers : un array vide par défaut.
- following : un array vide par défaut.
- topGames : un array vide par défaut.
- blockedUsers : un array vide par défaut.
- isAdmin : de type Boolean et false par défaut.
- isBanned : de type Boolean et false par défaut.

Game :

Lié à la collection *games*, ce schéma contient :

- userId: de type String, contient l'id du créateur de la partie.
- title : de type String.
- description: de type String.
- gameDate : de type Date.
- boardGameId : de type String, contient l'id dans la base de données de BoardGameGeek du jeu de société choisi.
- boardGameName: de type String, contient le nom du jeu de société choisi.
- city : de type String.
- zipcode : de type String.

- playersLevel : de type String.
- playersNumber : de type Number, sera sur 1 après la création de la partie.
- playersMax : de type Number, le minimum étant fixé à 2.
- players : un array vide, contiendra l'id de tous les participants à la partie.
- creationDate : de type Date et généré automatiquement avec la date du jour à la création de la partie.

Comment :

Lié à la collection *comments*, ce schéma contient :

- user : de type Object, contient l'id (user.id) de l'utilisateur ayant créé le commentaire et son nom (user.login).
- text : de type String.
- createdAt : de type Date et généré automatiquement avec la date du jour à la création du commentaire.
- gameId : de type String, contient l'id de la partie liée au commentaire.

La partie serveur s'exécute sur le port 5000 (port défini dans le fichier *index.js*) et possède cinq fichiers (dans le dossier */routes*) contenant toutes les routes nécessaires à répondre aux différentes requêtes.

Afin de contrôler qu'il n'y a aucune erreur lors de la création ou des modifications d'un compte ou d'une partie mais aussi de la connexion d'un utilisateur, quatre fichiers de validation ont été créés (dans le dossier */validation*), renvoyant au client des erreurs spécifiques si nécessaire.

4.2 - Client / Site

Afin de développer la partie website de notre projet, voici une présentation de l'architecture de celle-ci.

Le fichier */src/App.js* est le cœur du site et indique le chemin vers tous les composants à appeler. Si vous souhaitez rajouter une nouvelle page au site c'est ici qu'il faudra déclarer quel chemin appelle quel composant.

Le dossier */src/components* (voir le chapitre 1.4 pour l'arborescence complète du projet) contient l'ensemble des composants à afficher ; triés par sections du site (*Admin*, *Profile*, *CreateGame*, etc).

Deux sections sont générales et utilisées à de nombreux endroits sur le site : */src/components/Loading/Loading.js* (fournissant l'élément s'affichant lors du chargement d'un autre élément) et */src/components/Games/Game.js.* (fournissant le rendu d'une partie parmi une liste de parties).

C'est dans les fichiers situés dans le dossier */src/components* que se situe la majorité du code du site.

Le projet utilise également Redux afin de centraliser les données et les actions nécessaires au bon fonctionnement du site.

Pour cela on utilise in *store* global déclaré dans */src/store.js* et importé dans */src/App.js*. Les différentes actions appelant le serveur sont stockées dans */src/actions*, tandis que les reducers permettant la mise à jour des données du store (state globales) se trouvent dans */src/reducers*.

4.3 - Client / Application mobile

L'application mobile utilise la structure de base fournie par l'installeur Expo, à laquelle trois nouveaux dossiers ont été ajoutés :

- Le dossier *global* contient les valeurs qui doivent pouvoir être accessibles partout dans l'application.
- Le dossier *pages* contient les pages navigables dans l'applications. Ces pages ne contiennent aucune logique.
- Le dossier *Components* contient les composants avec la logique de l'application.

Le fichier *App.js* à la racine de l'application est le cœur de l'application, c'est à partir de ce fichier-ci que les pages sont appelées. Si vous ajoutez de nouvelles pages à l'application c'est dans ce fichier *App.js* qu'il faudra les définir à l'aide de *react-navigation*.

4.4 - API externes

Le projet Board Club fait appel à trois API externes :

- L'API de BoardGameGeek, site de référence au niveau mondial sur le jeu de société.
- L'API Vicopo permettant de chercher facilement des villes françaises.
- L'API Distance24 pour calculer la distance entre les villes.

Nous allons maintenant rentrer plus en détail sur le fonctionnement et l'intégration de ces API à notre projet.

BoardGameGeek :

BoardGameGeek est le site de référence au niveau mondial concernant les jeux de société, recensant les informations de centaines de milliers de jeux.

L'utilisation de leur API nous permet d'obtenir des informations détaillées sur des jeux précis (durée de jeu, lien vers une image, nom du jeu, etc) mais aussi de pouvoir faire une recherche à partir d'une chaîne de caractères.

Utilisée dans les fichiers :

- */BoardClub/app/BoardClub/Components/GameDetail.js*
- */BoardClub/website/src/components/CreateGame/Card.js*
- */BoardClub/website/src/components/CreateGame/SearchGames.js*
- */BoardClub/website/src/components/Games/Game.js*
- */BoardClub/website/src/components/Gameview/Details.js*
- */BoardClub/website/src/components/Games/Game.js*
- */BoardClub/website/src/components/Profile/Edit/EditTopGames/FavoriteCard.js*
- */BoardClub/website/src/components/Profile/Edit/EditTopGames/SearchCard.js*
- */BoardClub/website/src/components/Profile/Edit/EditTopGames/SearchGames.js*
- */BoardClub/website/src/components/Profile/TopGames/FavoriteCard.js*

Requêtes utilisées :

- *https://boardgamegeek.com/xmlapi2/search?query=*<texte de la recherche>*&type=boardgame* (méthode GET)

  Cette requête permet d'effectuer une recherche à partir d'une chaîne de caractères et d'obtenir les id des jeux correspondants. L'id et le nom du jeu recherché seront ainsi stockés dans la base de données lors de la création d'une partie ou de l'ajout d'un jeu dans son top personnel afin d'obtenir plus tard des informations sur ce jeu.

- *https://boardgamegeek.com/xmlapi2/thing?id=*<id du jeu> (méthode GET)
  Cette requête permet de récupérer des informations détaillées sur un jeu de société à partir de son id.

L'API de boardgamegeek renvoyant une réponse au format xml, nous avons dû parser la réponse pour en tirer les informations nécessaires.

A noter également que l'API de Boardgamegeek ne supporte pas les requêtes cross-domain (CORS). Pour contourner ce problème, le projet utilise un proxy inverse en exécutant cors-anywhere et en ajoutant *http://localhost:8080/* avant chaque requête à l'API de Boardgamegeek.

Vicopo :

L'API fournie par Vicopo nous permet de retrouver facilement les noms et code postaux de villes françaises à partir d'une simple recherche.

Concrètement, cela permet à l'utilisateur de sélectionner sa ville parmi une proposition et d'ainsi uniformiser les villes rentrées dans la base de données ; rendant ainsi la recherche de parties et le calcul de distances possibles et efficaces.

Utilisée dans les fichiers :

- */BoardClub/website/src/components/Admin/Games/EditGame.js*
- */BoardClub/website/src/components/CreateGame/InfoGame.js*
- */BoardClub/website/src/components/Gameview/Edit/EditOwnGame.js*
- */BoardClub/website/src/components/Profile/Edit/EditLogin/EditLoginInformations.js*

Requête utilisée :

- *https://vicopo.selfbuild.fr/cherche/*<texte de la recherche> (méthode GET)

  Cette requête permet d'effectuer une recherche à partir de la chaîne de caractères saisie par le client dans un formulaire (qui peut aussi bien être un nom de ville ou un code postal) et récupérer en retour toutes les villes françaises correspondant à la recherche accompagnées de leur code postal respectif. Ces villes seront alors affichées sous le formulaire de manière dynamique à chaque changement de saisie ; l'utilisateur

n’ayant plus qu’à cliquer sur sa ville désirée pour effectuer son choix.

Tout comme pour Boardgamegeek, l’API de Vicopo ne supporte pas les requêtes cross-domain (CORS). Pour contourner ce problème, le projet utilise un proxy inverse en exécutant cors-anywhere et en ajoutant *http://localhost:8080/* avant chaque requête.

Distance24 :

Distance24 est une API demandant en paramètres un certain nombre de villes et renvoyant en retour de nombreuses informations sur celles-ci.

En mettant deux villes en requête, on peut ainsi obtenir la distance entre ses deux villes ce qui permet d’afficher sur le projet les parties les plus proches de la ville définie dans son profil.

Utilisée dans le fichier :

- */BoardClub/website/src/components/Homepage/ListNearestGames.js*

Requête utilisée :

- *https://fr.distance24.org/route.json?stops=*<première ville>*/*<seconde ville> (méthode GET)

  Cette requête permet d’effectuer une recherche à partir de deux villes et d’obtenir de nombreuses informations, dont celle qui nous intéresse : la distance entre celles-ci.

Tout comme pour Boardgamegeek et Vicopo, l'API de Distance24 ne supporte pas les requêtes cross-domain (CORS). Pour contourner ce problème, le projet utilise un proxy inverse en exécutant cors-anywhere et en ajoutant *http://localhost:8080/* avant chaque requête.