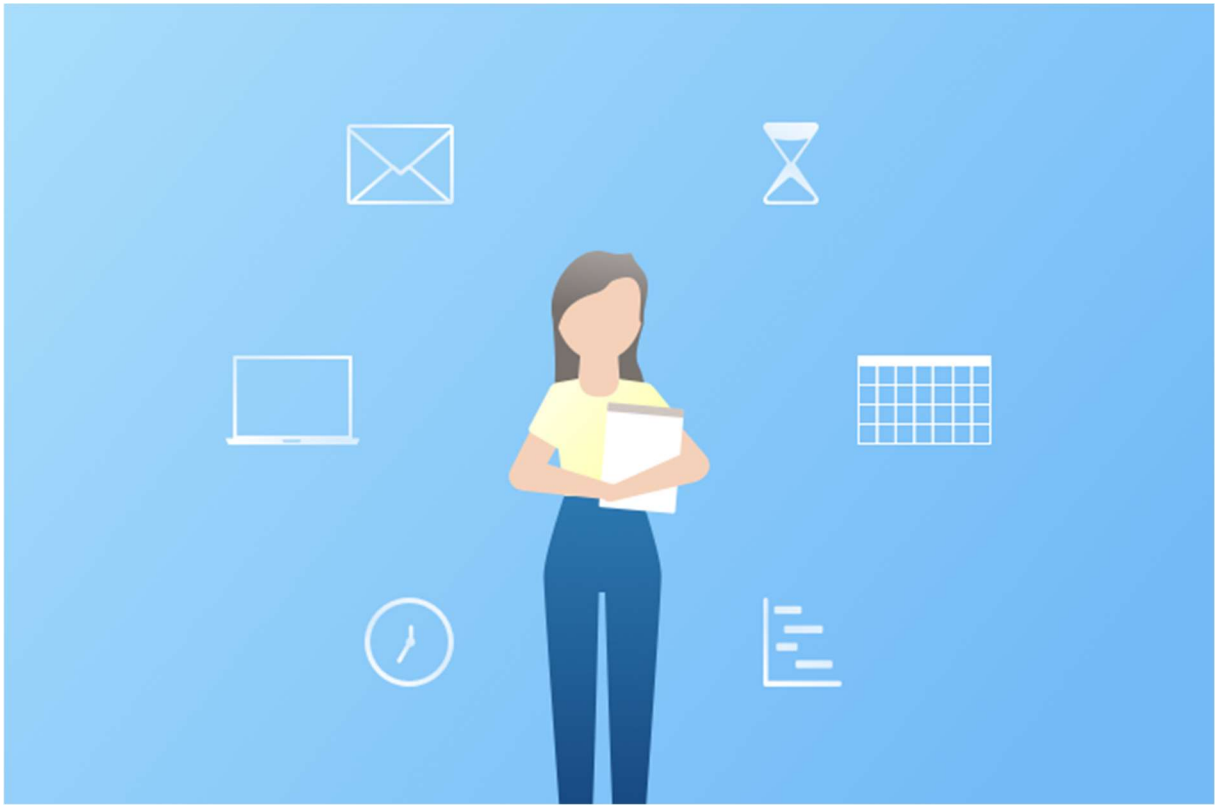
Board Games Manager (Angular)



Luca, Coduri

Luca.coduri@gmail.com

SI-CA2a

07.02.2020

Table des matières

Introduction

Cadre, description et motivation

Ce projet est réalisé dans le cadre du CPNV dans le but de déterminer mon niveau avec le Framework Angular ainsi qu'à me faire découvrir le déroulement du TPI.

Pour ce projet, Monsieur Viret ma proposer de créer un gestionnaire de jeux de société. Ce gestionnaire permettra de gérer sa collection de jeux. Il sera possible d'y ajouter des jeux à l'aide d'une banque de données d'une autre API gratuite. Mais aussi de supprimer ces jeux ou de les modifier avec un formulaire pour ajouter des informations ou changer ce qui est donné par l'API.

La dernière fonctionnalité est de pouvoir créer un vote afin de déterminer par exemple le prochain jeu qu'une bande de pote souhaiterait jouer.

Pour créer le vote, il sera possible de choisir des jeux de sa collection mais aussi depuis la banque de données de l'API gratuite.

Cette partie sera donc la face visible de l'iceberg, ce qu'on appelle plus communément le Frontend. Ce dernier sera codé en utilisant le Framework Angular.

Pour pouvoir gérer ses jeux il faut évidemment pouvoir s'enregistrer sur le site. C'est pourquoi une API sera codée à l'aide de Node.js. Cette API communiquera avec une base de données que l'on mettra en place avec Bastian.

Les fonctions de ce serveur Backend seront de gérer les demandes à propos des utilisateurs, des collections, des jeux ainsi que des votes.

Organisation

Organisation générale du projet :

Luca Coduri, luca.coduri@cpnv.ch, 079 835 60 66

Bastian Chollet, bastian.chollet@cpnv.ch, 079 784 31 20

Responsable de projet :

ANDOLFATTO Frédérique, Frederique.ANDOLFATTO@cpnv.ch, 024 / 55 + 76123

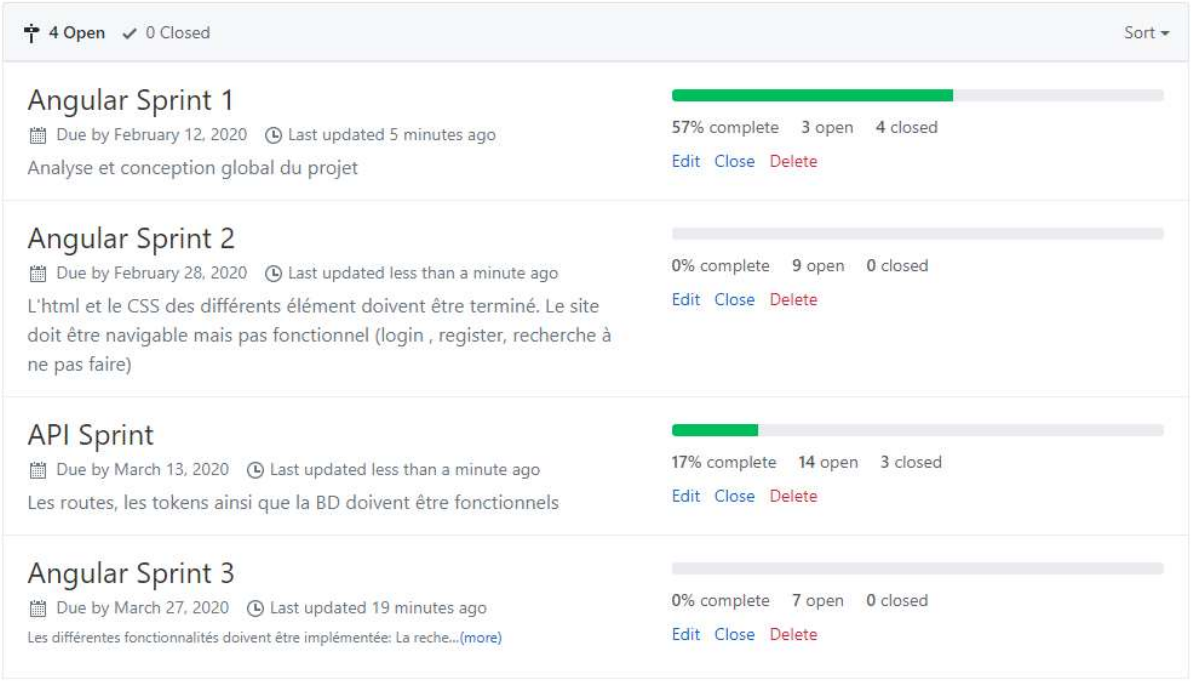
Viret Loic, loic.viret@cpnv.ch

	Luca Coduri	*Bastian Chollet*
Backend	*X*	*X*
Frontend	*X*	
Documentation	*X*	
Maintenance Planning	*X*	

Objectifs

1. Une base de données fonctionnelle et optimisée.
2. Un site intuitif avec une belle interface.
3. Un site fonctionnel.
4. Une API fonctionnelle.
5. Tous les bugs (connus) doivent être corrigés.
6. Au moins un test unitaire doit être mis en place.

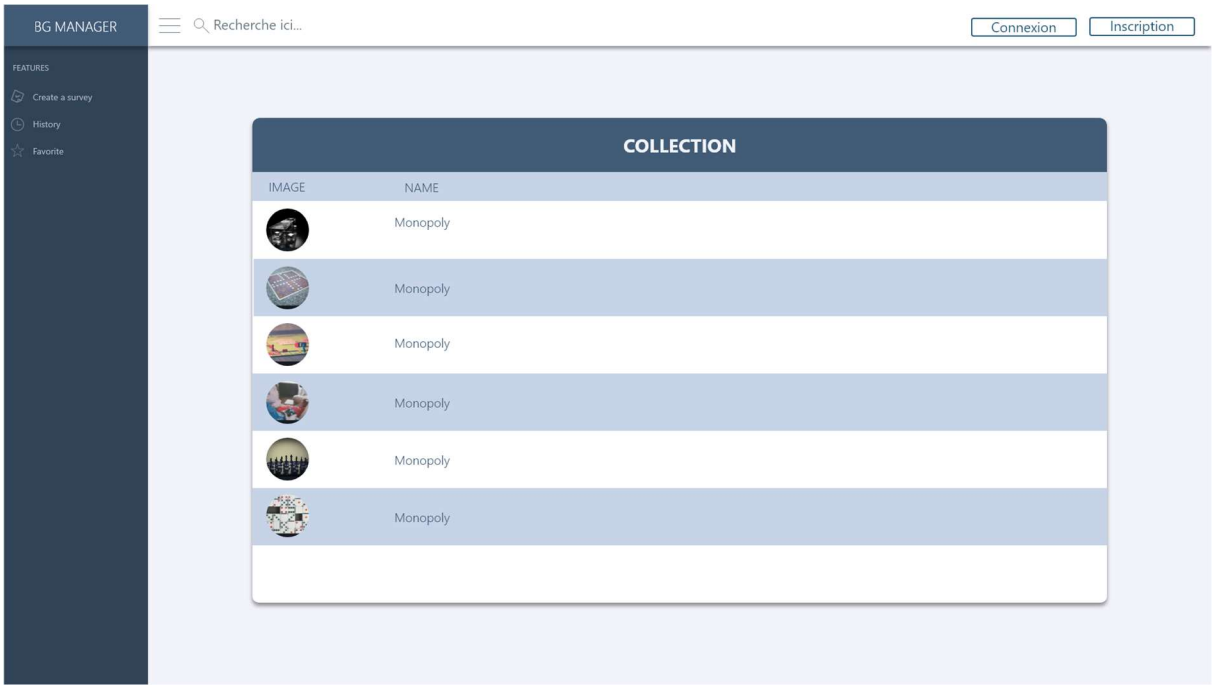
Planification initiale



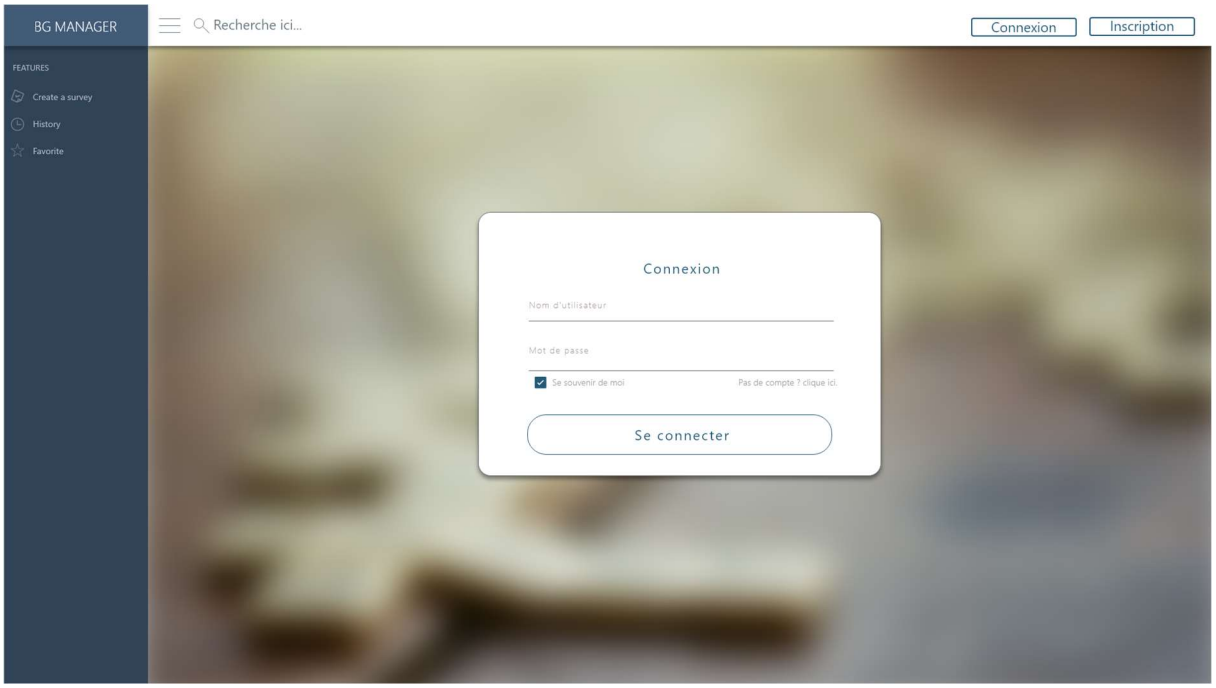
Analyse

Maquettes :

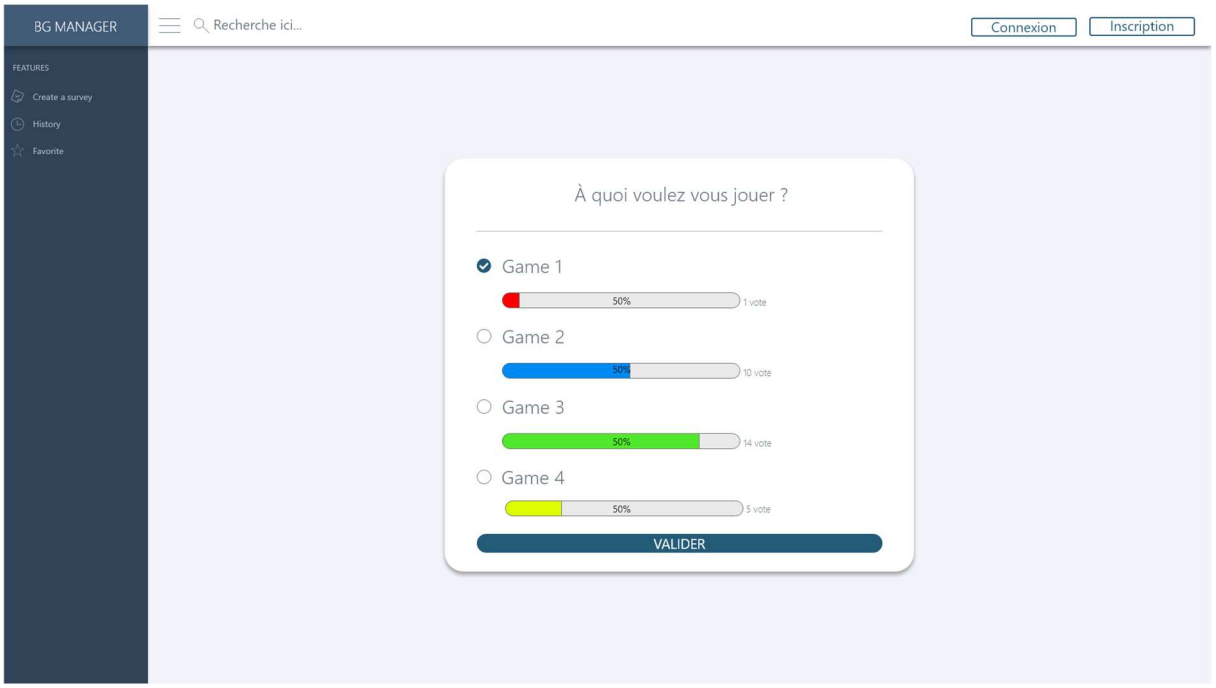
La collection d'un utilisateur :



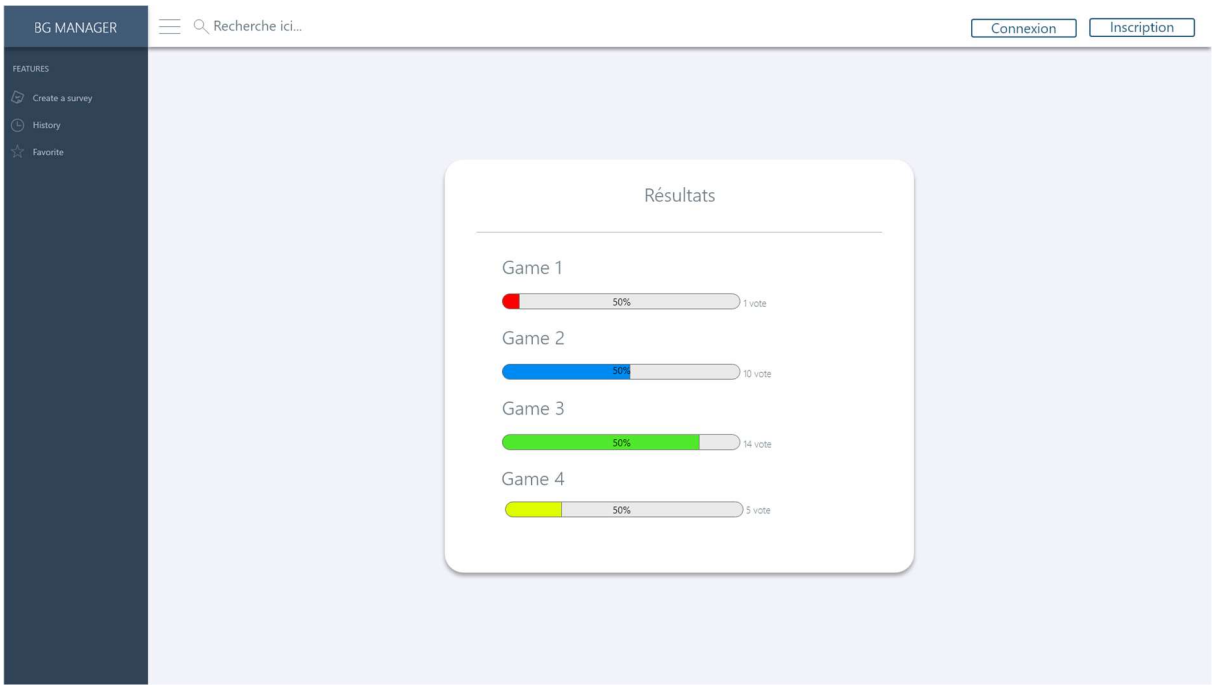
Page de login :



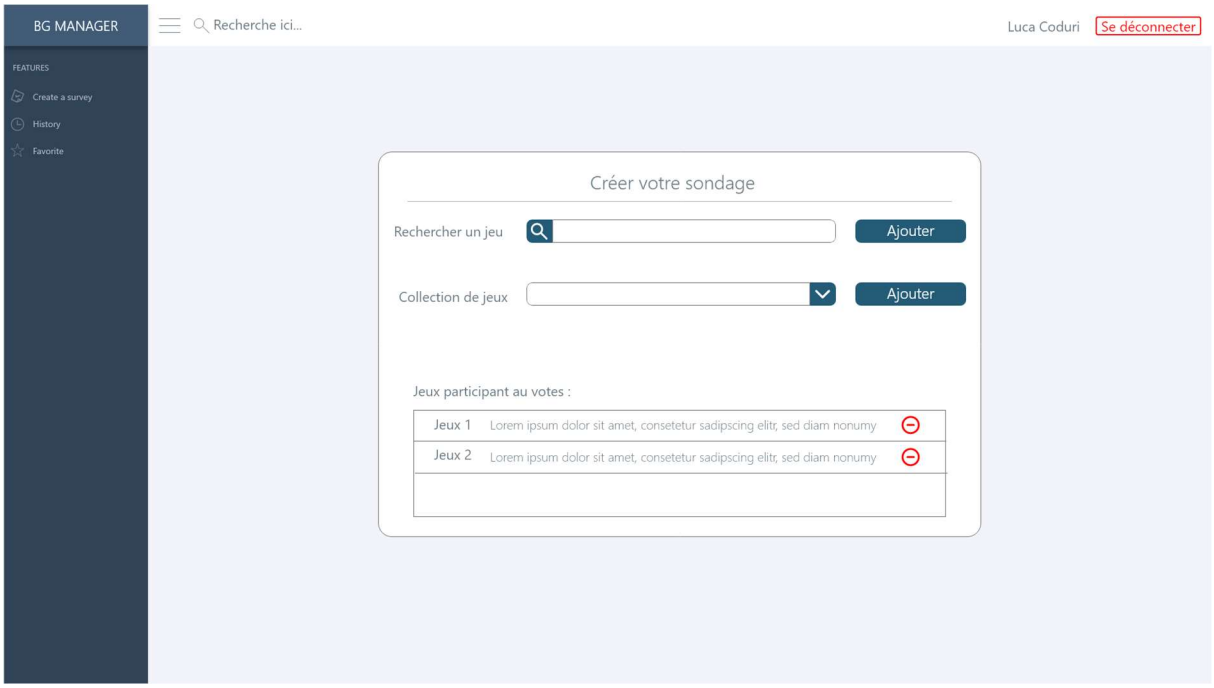
Page de vote :



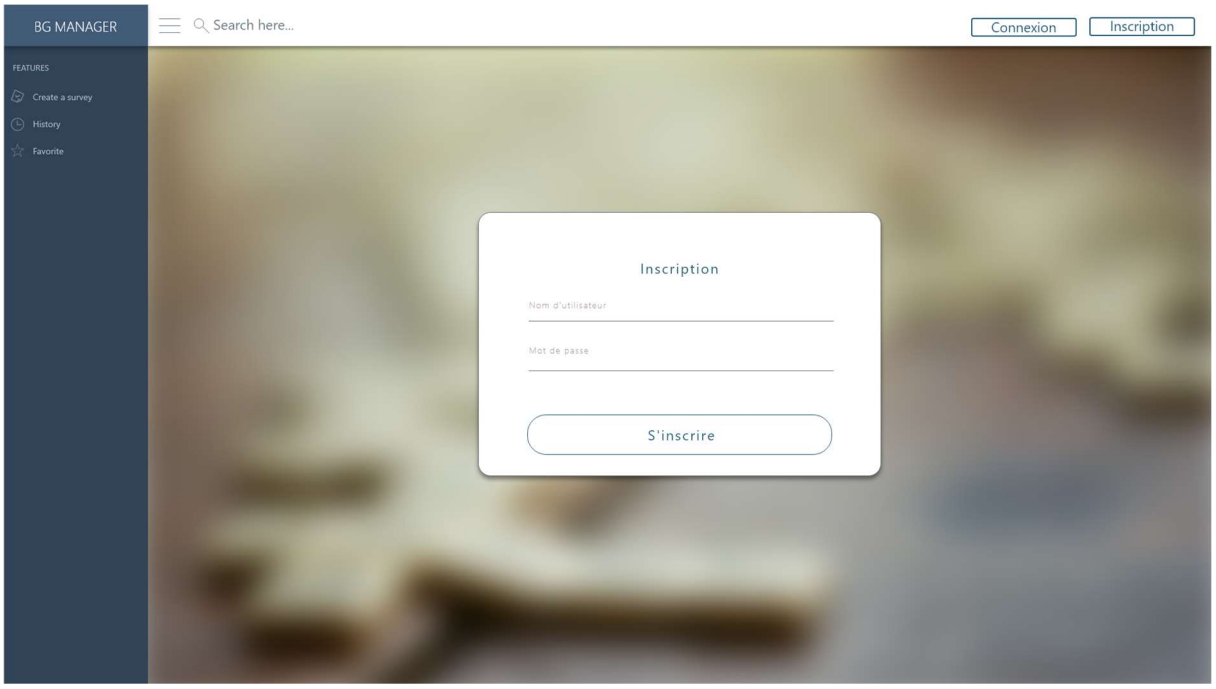
Résultats d'un sondage :



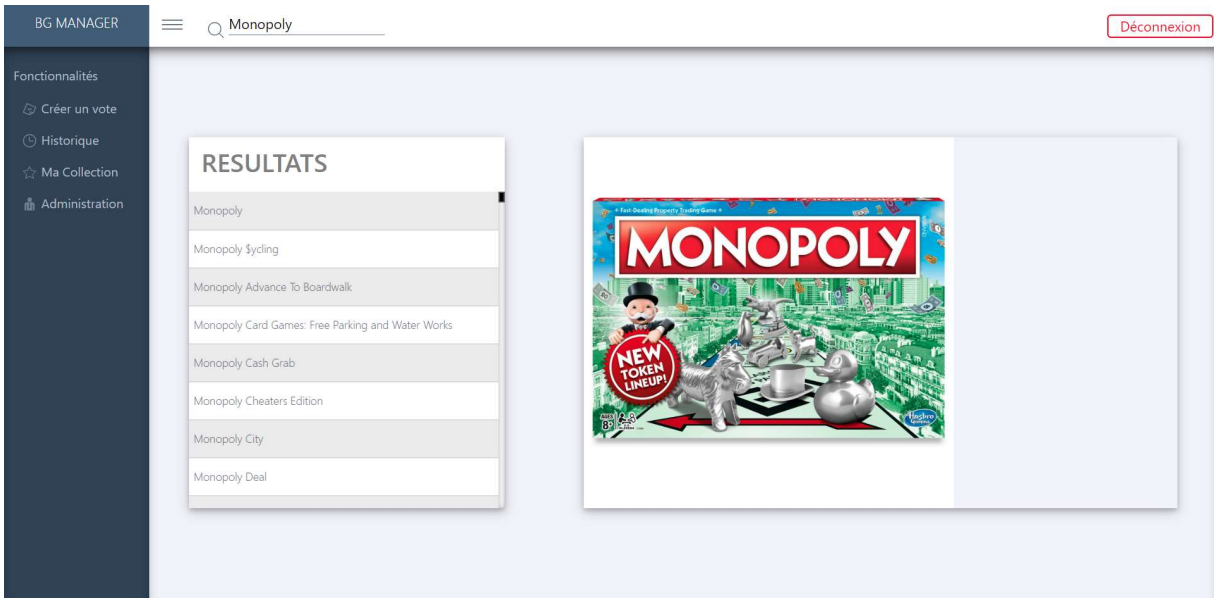
Création d'un sondage :



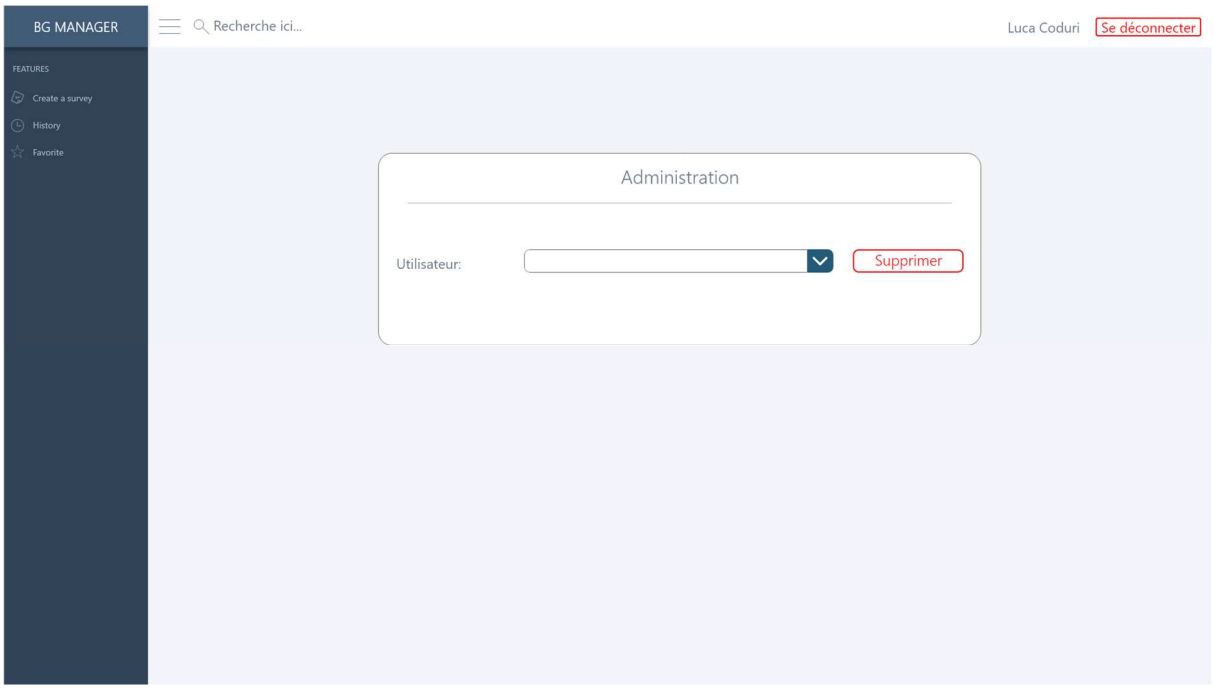
Inscription, cette page permet de s'inscrire en fournissant tout simplement un nom et un mot de passe :



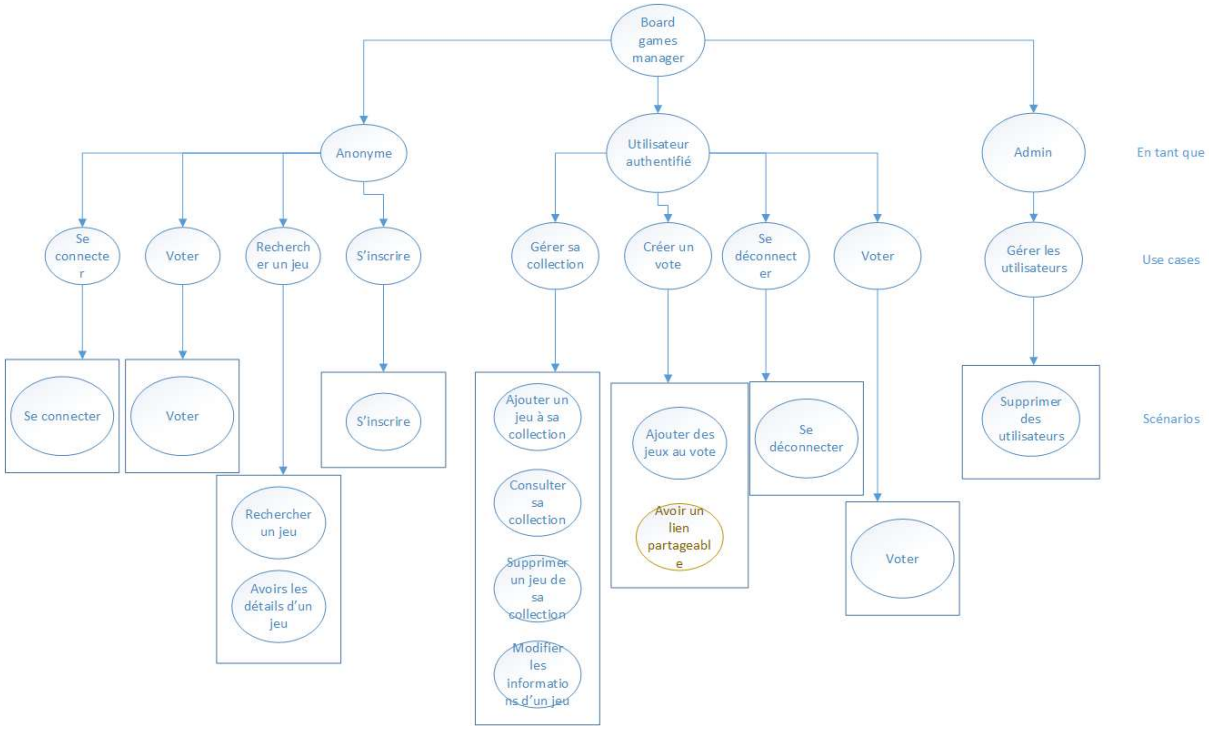
Résultat d'une recherche:



Page administrateur, cette page permet de supprimer un utilisateur enregistrer sur le site :



Use cases et scénarios



En tant qu'utilisateur enregistré :

Note : un utilisateur possède tous les avantages d'un utilisateur anonyme.

Créer un sondage

Action	Condition	Réaction
Créer un sondage personnalisé		
L'utilisateur clique sur « créer un sondage » dans le menu latéral. Puis rempli les différents champs et valide son sondage.		Le sondage sera mis en place ainsi qu'un lien de partage.

Gérer sa collection

Action	Condition	Réaction
Afficher sa collection de jeu de société		
L'utilisateur clique sur « ma collection » dans le menu latéral.		La liste de jeu apparaît avec les informations les plus importantes.
Avoir plus de détails à propos d'un jeu		
L'utilisateur clique sur le jeu de son choix.		Une page avec les détails du jeu apparait.
Ajouter un jeu à sa collection		
L'utilisateur clique sur « Ajouter un jeu » se trouvant sur la page d'affichage de sa collection.		Un formulaire lui permet de saisir les données manuellement ou alors de pré remplir certaines données avec la barre de recherche. Puis un bouton valider lui permet de valider l'ajout.
Modifier les informations à propos d'un jeu dans la collection		
L'utilisateur clique sur le bouton éditer à côté d'un des jeux dans sa collection.		Un formulaire lui permettant de modifier les différentes informations s'affiche.
Supprimer un jeu de sa collection		
L'utilisateur clique sur la croix à côté d'un jeu dans la liste	Une fenêtre de confirmation apparait.	Si l'utilisateur confirme le jeu est supprimé sinon rien ne se passe.

Se déconnecter

Action	Condition	Réaction
Se déconnecter de sa session		
L'utilisateur clique sur le bouton « déconnecter » situé dans la barre dans la partie supérieure.		L'utilisateur est déconnecté et peut maintenant naviguer sur le site en tant qu'anonyme.

En tant qu'administrateur :

Note : un administrateur possède tous les avantages d'un utilisateur enregistré.

Gérer les utilisateurs

Action	Condition	Réaction
Supprimer un utilisateur		
L'administrateur clique sur « administration » dans le menu latéral. L'administrateur choisit un utilisateur puis clique sur supprimer	Une fenêtre de confirmation apparaît.	Si l'administrateur confirme l'utilisateur est supprimé sinon dans le cas contraire rien ne se passe.

En tant qu'utilisateur anonyme :

Rechercher un jeu

Action	Condition	Réaction
Rechercher un jeu dans la base de données		
L'utilisateur entre le nom d'un jeu qu'il souhaite rechercher et confirme avec « entrer ».		Une page avec une liste de jeu correspondant à la recherche apparait.
Avoir les détails d'un jeu rechercher		
L'utilisateur choisi le jeu qu'il souhaite dans la liste apparue après une recherche.		Une page contenant les détails du jeu apparait.

Voter

Action	Condition	Réaction
Voter pour un jeu dans la liste de choix		
L'utilisateur clique sur le lien du vote. Puis choisi une des réponses possibles. Pour valider son vote il clique sur valider.	Si l'utilisateur a déjà voté pour ce sondage, l'état actuel du vote sera affiché mais il ne pourra pas revoter.	Le vote est validé et l'utilisateur est redirigé vers sur l'état actuel du sondage.

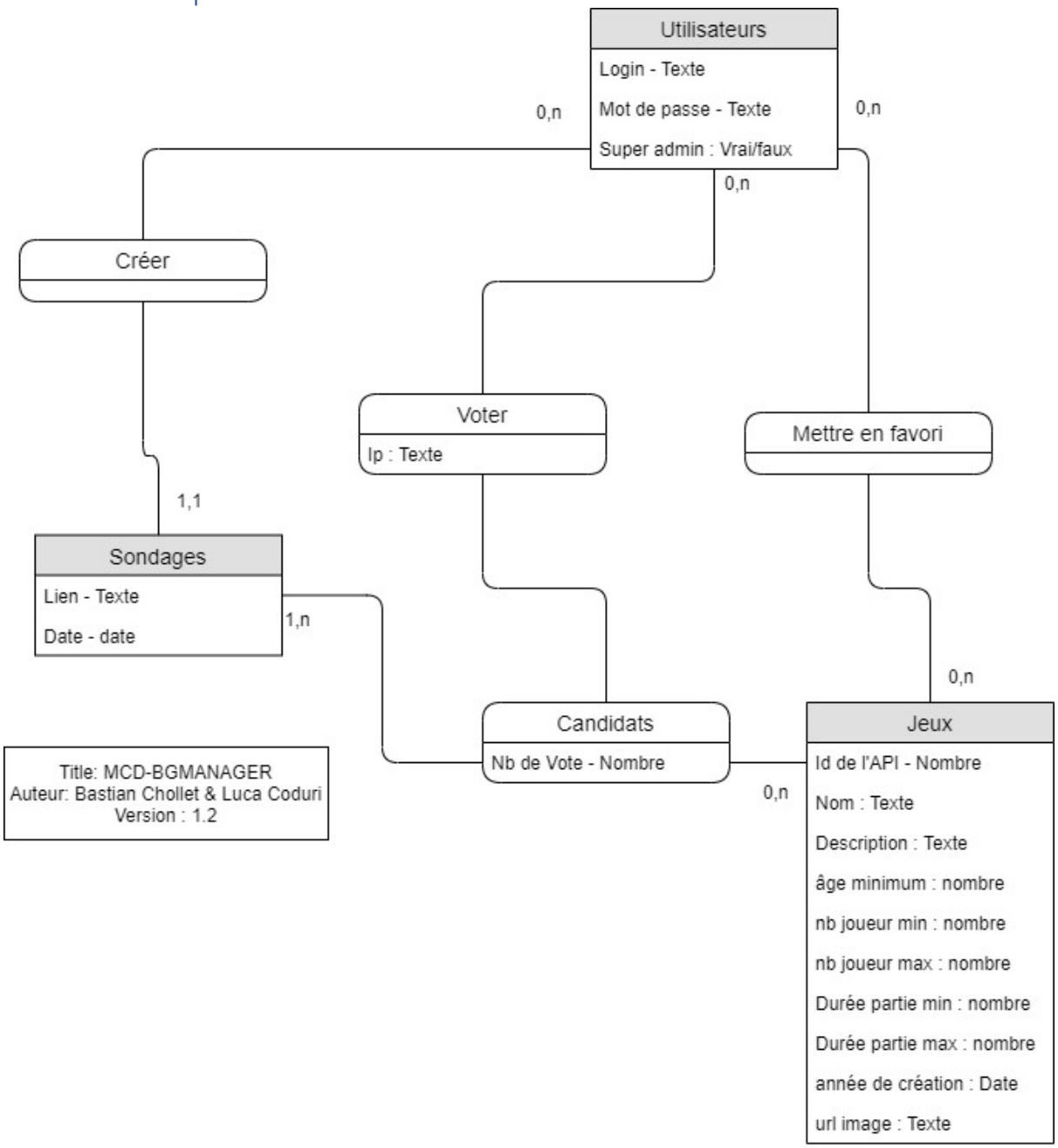
S'inscrire

Action	Condition	Réaction
Ouvrir la page d'inscription		
L'utilisateur clique sur le bouton « S'inscrire » situé dans la barre dans la partie supérieure.		Un formulaire d'inscription apparait.
Valider l'inscription		
L'utilisateur rempli le formulaire et confirme l'inscription avec le bouton valider.	Le système vérifie le nom d'utilisateur	Si le nom est disponible l'utilisateur peut à présent se connecter sinon le formulaire indique les champs à modifier.

Se connecter

Action	Condition	Réaction
Afficher la page de connexion		
L'utilisateur clique sur le bouton « Se connecter » situé dans la barre dans la partie supérieure.		La page de de connexion apparait.
Valider la connexion		
L'utilisateur entre ses information et valide la connexion en cliquant sur « se connecter »		L'utilisateur est maintenant connecté et redirigé sur sa collection.

Modèle Conceptuel de Données



Stratégie de test

Les tests vont se dérouler de différentes manières en fonction du code qui est testé.

Backend :

Pour comprendre les résultats retourner par l'API de BGG et ainsi pouvoir débugger par la suite, j'utilise Postman. Je pourrai ainsi tester le cas où tout fonctionne comme prévu, quand une route est appelée avec de mauvaises données ou encore quand ça ne fonctionne pas tout simplement.

Cependant pour tester le backend j'ai appris à utiliser le module Jasmine afin de pouvoir créer des tests automatiques pour express. Ce module est bien plus pratique que Postman car ça me permet d'avoir une réponse claire et rapide sur la réussite des tests et ainsi vérifier si au fur et à mesure que j'ajoute des fonctionnalités que les autres fonctionnent toujours.

Jasmine me permet donc de travailler en TDD pour l'API, car je peux commencer par écrire les tests et coder la fonctionnalité après.

Frontend :

Angular-cli contient déjà tout ce qu'il faut pour faire les tests unitaires et d'intégrations.

Le hasard a fait qu'il s'agisse aussi de Jasmine ce qui est pratique. Cependant je ne ferai pas de tests automatiques pour Angular car cela reste tout de même légèrement plus compliqué qu'avec express. Ayant déjà beaucoup à comprendre, je me suis permis de laisser ces tests de côté.

Lors du développement Je ferai donc mes tests à l'aide de console.log() à chaque fois que quelque chose ne fonctionne pas comme prévu.

Quand je pensais avoir finis je vérifiais que tout fonctionne en naviguant sur le site et en essayant les différentes fonctionnalités.

Budget

Aucune dépense ne sera nécessaire pour ce projet.

Implémentation

Choix techniques

Pour ce projet, il a été choisi d'utiliser le Framework Angular afin de déterminer mon niveau et réaliser un cahier des charges pour le TPI qui soit réalisable. Mais cela me permet aussi de m'entrainer et découvrir des fonctionnalités que je ne connais pas.

Afin de faciliter la conception du MCD, Draw.io a été utilisé. Ce dernier est pratique car il contient des modèles qui rendent la création de mcd faciles.

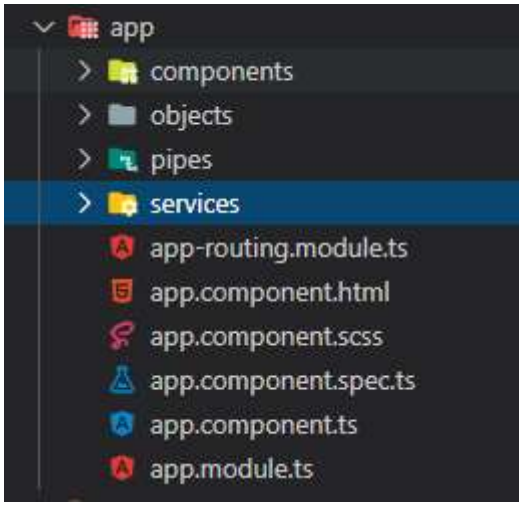
Pour le MLD, nous avons choisi avec Bastian d'utiliser MySQL Workbench car il rend lui aussi cette tâche particulièrement facile et possède l'avantage de pouvoir exporter le modèle au format d'un script SQL. Il nous suffit donc d'exécuter ce dernier sur le serveur MySQL.

Afin de me faciliter la tâche lors de la conception de l'interface graphique, j'ai fait des recherches sur des outils de maquette. Cette recherche ma donc mené sur Adobe Xd qui permet de créer des prototypes rapidement et facilement. L'interface fut agréablement intuitive et j'ai donc pu prendre mes repères assez facilement.

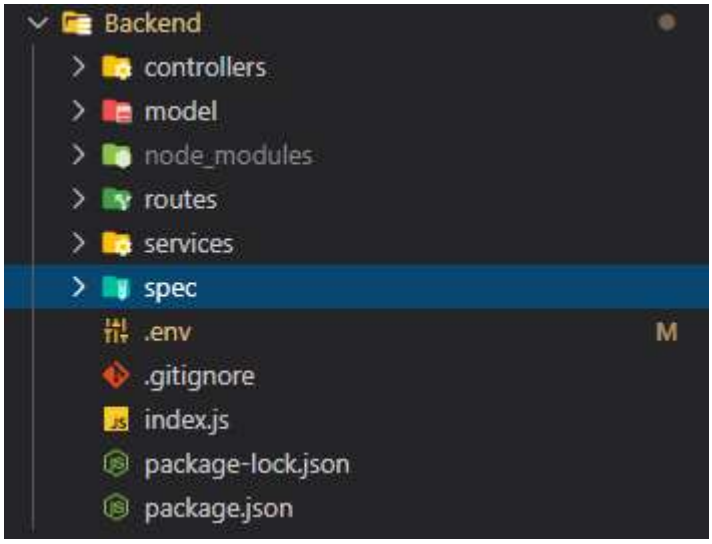
J'ai décidé de ne pas me concentrer sur l'aspect du site en fonction des différentes tailles d'écran car, j'ai trouvé que ce n'était pas vraiment le but premier de ce projet. C'est pourquoi le site sera affiché comme je l'ai pensé seulement sur des écrans ayant comme résolution 1920X1080.

Pour le matériel, Bastian et moi avons pensé être une bonne idée d'installer l'API que nous produirons sur un Raspberry PI qui tourne chez moi. Cette API sera donc disponible peu importe l'endroit où nous sommes. Cela peut être pratique car nous n'aurons pas le besoin de lancer une instance du serveur à chaque fois que nous voulons coder sur nos PC respectifs.

Concernant la structure de dossier, le CLI d'Angular le gère plus ou moins pour moi. C'est-à-dire que lors de la création du projet le CLI créer les différents fichiers et dossiers pour rester organisé. Cependant il met les composants, services, pipes et intercepteur dans le même dossier. C'est pourquoi j'ai choisi de créer un dossier spécifique pour chacun de ces types de fichier.



Malheureusement il n'y a pas de CLI pour la création de notre (Bastian et moi) API. C'est pourquoi nous avons fait quelque recherche et décidé d'adopter cette structure de dossier :

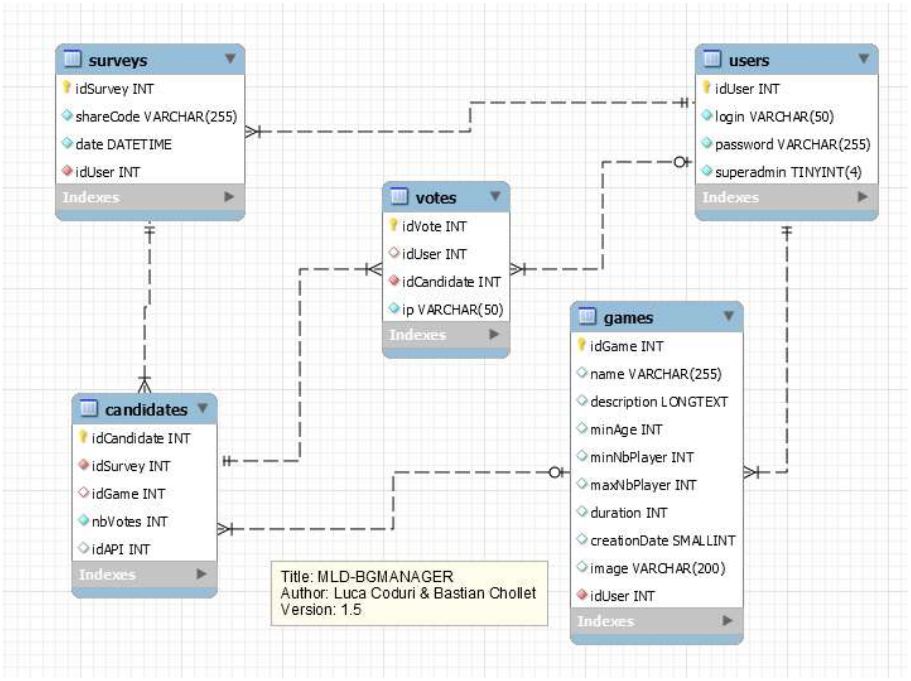


Pour ne pas réinventer la roue, Nous avons utilisé plusieurs bibliothèques, voici les plus importantes :

- Express afin d'avoir une base de serveur web.
- JSONWebToken afin de différencier les utilisateurs.
- Jasmine pour les tests automatiques.
- Mysql2 afin de contacter la base de données.

Pour connaitre les bibliothèques que je n'ai pas citées, veuillez regarder dans le fichier package.json à la racine du dossier Backend.

Modèle Logique de données



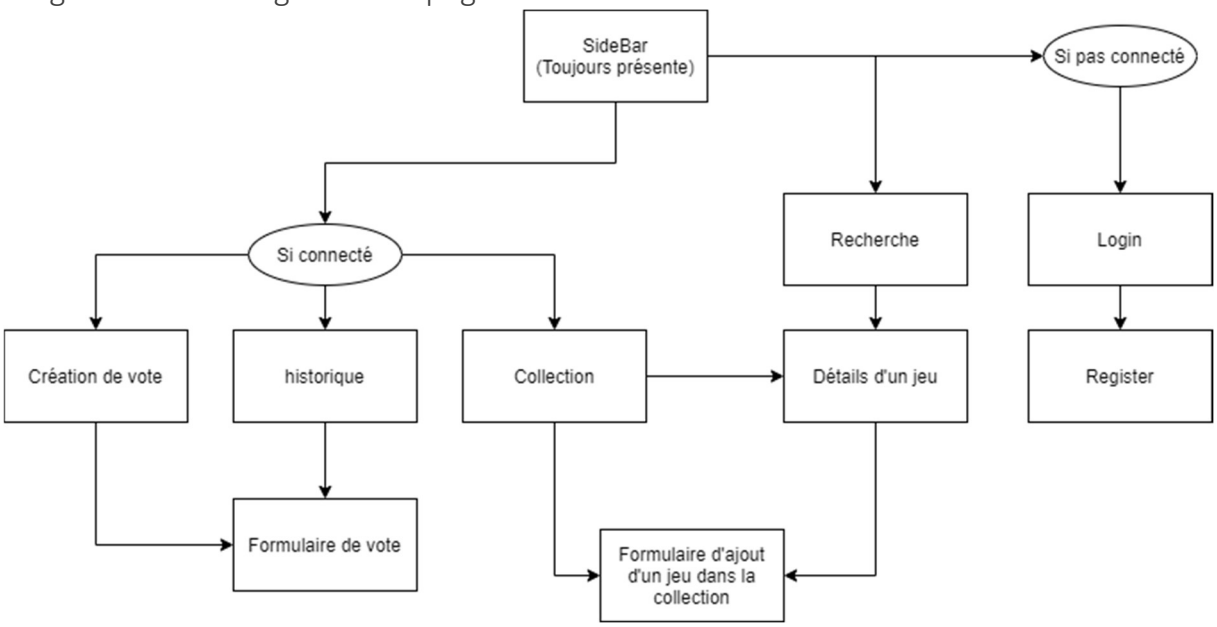
Conventions de nommage

Pour les conventions de nommage, j'appliquerai ce qui est détaillé sur ce site :
http://snowdream.github.io/javascript-style-guide/javascript-style-guide/fr/nantions.html

Tout y est très bien expliqué et j'ai trouvé que c'était logique comparée à ce que je fais d'habitude.

Points techniques spécifiques

Diagramme de navigation des pages :



Note :

L'historique n'a pas été codé.

Contrairement à un site web traditionnel qui tourne entièrement sur un serveur, une application Angular tourne en partie chez le client. C'est-à-dire qu'il n'est pas conseillé d'y intégrer des mots de passe de basse de données ou autres données sensible au risque de se faire pirater. C'est pourquoi le projet est divisé en deux applications. Nous avons le frontend qui correspond donc à Angular et le backend qui correspond à Node.js. Ils communiquent entre eux à l'aide du protocole http. Afin que le serveur Node.js puisse différencier les utilisateurs nous avons mis en place un système de token. Voici à quoi ressemble un token :

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Comme on peut le voir il est composé de trois clés. Chacune de ses clés à une utilité différente. La partie en rouge contient une information permettant de déterminer le cryptage. La clé verte contient les données comme le nom d'utilisateur ou son id mais il pourrait y avoir tous ce qu'on voudrait. Les deux premières clés sont facilement lisibles de tous puisqu'ils sont cryptés en base 64. Cependant la troisième clé est un cryptage des deux autres à l'aide d'un mot de passe. Cela signifie qu'il est impossible de modifier les données du token car le la troisième clé ne correspondra plus avec les données modifiées. On peut donc grâce au token être sûr de l'intégrité des données.

Tests

Tests effectués

Back-end

Pour le backend, j'ai écrit quelques tests automatiques avec Jasmine :

- Connexion à la base de données
- Route POST /users retourne 201
- Route POST /users créer un utilisateur dans la BD
- Route POST /login retourne 201
- Route POST /login retourne 401
- Route GET /BGG/games/:name retourne un tableau de jeu
- Route GET /BGG/games/:name retourne aucun résultat
- Route GET /BGG/games/:idGame/details retourne les détails
- Route GET /BGG/games/:idGame/details retourne aucun résultat
- Route POST /users/:idUser/surveys/:idSurvey/vote retourne 200
- Route POST /users/:idUser/games retourne 201
- Route POST /users/:idUser/games retourne 400
- Route Delete /users/:idUser retourne 200
- Route Delete /users/:idUser retourne 400

Je peux donc vérifier si ces routes fonctionnent comme prévu en exécutant la commande « npm test »

Pour les autres routes, les tests ont été fait avec postman.

Front-end

Scénario	Statut	Quand ?	Qui ?	Comment	Note
Créer un sondage personnalisé	ok	20.03.2020	Luca Coduri	Chrome	
Avoir plus de détails à propos d'un jeu	*ok*	20.03.2020	Luca Coduri	Chrome	
Ajouter un jeu à sa collection	*ok*	20.03.2020	Luca Coduri	Chrome	
Modifier les informations à propos d'un jeu dans la collection	*ok*	20.03.2020	Luca Coduri	Chrome	
Supprimer un jeu de sa collection	*ok*	20.03.2020	Luca Coduri	Chrome	
Se déconnecter de sa session	*ok*	20.03.2020	Luca Coduri	Chrome	
Supprimer un utilisateur	*ok*	20.03.2020	Luca Coduri	Chrome	
Rechercher un jeu dans la base de données	*ok*	20.03.2020	Luca Coduri	Chrome	
Avoir les détails d'un jeu rechercher	*ok*	20.03.2020	Luca Coduri	Chrome	
Voter pour un jeu dans la liste de choix	*ok*	20.03.2020	Luca Coduri	Chrome	

Ouvrir la page d'inscription	*ok*	20.03.2020	Luca Coduri	Chrome	
Valider l'inscription	*ok*	20.03.2020	Luca Coduri	Chrome	
Afficher la page de connexion	*ok*	20.03.2020	Luca Coduri	Chrome	
Valider la connexion	*ok*	20.03.2020	Luca Coduri	Chrome	
Créer un sondage personnalisé	ok	02.04.2020	Bastian Chollet	Firefox	
Avoir plus de détails à propos d'un jeu	*ok*	02.04.2020	Bastian Chollet	Firefox	
Ajouter un jeu à sa collection	*ok*	02.04.2020	Bastian Chollet	Firefox	
Modifier les informations à propos d'un jeu dans la collection	*ok*	02.04.2020	Bastian Chollet	Firefox	
Supprimer un jeu de sa collection	*ok*	02.04.2020	Bastian Chollet	Firefox	
Se déconnecter de sa session	*ok*	02.04.2020	Bastian Chollet	Firefox	
Supprimer un utilisateur	*ok*	02.04.2020	Bastian Chollet	Firefox	
Rechercher un jeu dans la base de données	*ok*	02.04.2020	Bastian Chollet	Firefox	
Avoir les détails d'un jeu recherché	*Ok~*	02.04.2020	Bastian Chollet	Firefox	L'année affiché est fausse. Elle affiche tout le temps 1994
Voter pour un jeu dans la liste de choix	*ok*	02.04.2020	Bastian Chollet	Firefox	
Ouvrir la page d'inscription	*ok*	02.04.2020	Bastian Chollet	Firefox	
Valider l'inscription	*ok*	02.04.2020	Bastian Chollet	Firefox	
Afficher la page de connexion	*ok*	02.04.2020	Bastian Chollet	Firefox	
Valider la connexion	*ok*	02.04.2020	Bastian Chollet	Firefox	

Erreurs restantes

Il reste encore beaucoup d'erreur, notamment sur l'API. Avec Bastian Chollet nous aurions dû passer plus de temps sur l'API car c'est la base de nos projets.

Lorsqu'on ne fournit pas toutes les données dont une route à besoin pour fonctionner, la plupart du temps l'API finit par crasher. Ce qui est assez problématique car l'utilisateur de notre API ne fera pas tout forcément de la manière que nous avions pensée. Dans notre cas ça ne devrait pas trop poser de souci car nous connaissons ce qui déclenche ces problèmes et faisons-en sorte de les éviter.

Sur la partie Frontend il y a aussi des erreurs, mais elles n'empêchent pas l'utilisation de l'application. Ce sont surtout des erreurs dues à de mauvaises manipulations, par exemple entrer des caractères non autorisés dans un formulaire.

Important : l'application n'a été testé que sur Chrome et Firefox il est donc possible que des erreurs surviennent lors de son utilisation avec d'autre navigateur internet.

Conclusions

Le but premier de ce projet était de pouvoir évaluer mon niveau avec Angular afin de savoir si je pourrai utiliser cette technologie pour mon TPI. Je pense que cet objectif a été atteint. Mon projet a pu être mené à bien, même s'il reste quelques bugs. J'ai pu cependant grâce à ce projet apprendre de mes erreurs afin de ne plus les répéter par la suite. Je pourrais donc être plus rapide et plus efficace lors du TPI de fin d'année. Par exemple durant ce projet je n'ai pas trop aimé la manière dont j'ai géré les différents services avec Angular et je n'avais pas la possibilité de corriger sous peine de devoir recommencer une grosse partie du projet. C'est donc quelque chose que je changerai dans le futur.

Par rapport aux objectifs que j'ai cités au début de ce document, je suis plutôt satisfait. J'ai pu remplir quasiment la totalité de mes objectifs. Le seul objectif qui n'a pas été atteint est de corriger les bugs connus. Malheureusement je n'ai pas eu le temps de corriger tous les bugs. En revanche j'ai corrigé les plus importants d'après moi.

Nous avons avec Bastian peut-être pas assez planifier la production de l'API, ce qui fait que nous avions dû changer et repenser la manière dont les requêtes sont transmises à l'API. Par conséquent les tests automatiques qui avaient été écrits au préalable ne fonctionnaient plus, j'ai donc dû les réécrire de A à Z. J'ai effectué le même travail deux fois alors que nous aurions pu éviter cela si nous nous étions un peu mieux renseigné et organisé.

En dehors de ça je ne pense pas avoir eu trop de problèmes. Evidemment je ne sais pas tous faire mais en général une recherche sur google me permet de résoudre mes problèmes assez rapidement.

Ce projet peut être encore beaucoup amélioré, notamment en rajoutant des fonctionnalités comme un historique de votes, un système de recherche avancée et surtout en corrigeant les dernies bugs qui restent.

Annexes

API :

Voici une petite documentation expliquant comment utiliser les différentes routes de notre API.

Bastian et moi avons créé deux types de route : les publiques et les privées. Pour pouvoir accéder aux routes privées il faut fournir un token dans le header de la requête. Ce token peut-être récupérer à l'aide d'une autre route à condition que le nom d'utilisateur et le mot de passe soient correcte.

Routes publiques :

Post "/users":

Permet de créer un utilisateur dans la BD.

Données attendues dans le body :

```
{
	username: "nom",
	password: "mot de passe"
}
```

Réponse :

Confirmation de la création du compte.

Post "/login"

Permet de récupérer un token afin de pouvoir accéder aux routes privées.

```
{
	username: "nom",
	password: "mot de passe"
}
```

Get "/BGG/games/:name"

Permet de rechercher les jeux ayant un nom ressemblant au paramètre fourni dans le lien

Données attendues :

Indiquer le nom du jeu dans le lien à la place de *:name*

Réponse :

Un tableau JSON contenant les ID et les noms des jeux complet correspondant à la donnée fournie.

Get "/BGG/games/:idGame/details"

Permet d'avoir les détails d'un jeu

Données attendues :

Indiquer l'ID du jeu dont on veut avoir les détails dans le lien à la place *:idGame*

Réponse :

Un objet JSON avec tous les détails du jeu disponible dans la BD

Post "/users/:idUser/surveys/:idSurvey/candidates/:idCandidate/vote”
Permet de voter pour un candidat d’un sondage

Données attendues : il faut fournir l’id du sondage à la place de :idSurvey ainsi que l’id du candidat à la place de :idCandidate.

Réponse : Une confirmation que la requête à fonctionné.

Get "/users/surveys/:shareCode"
Permet de récupérer les infos d’un sondage

Données attendues :

Il faut fournir le code de partage à la place de :shareCode dans le lien.

Réponse :

Les informations à propos du sondage.

Get "/users/:idUser/surveys"
Permet de récupérer les sondages d’un utilisateur

Données attendues :

Il faut fournir l’id de l’utilisateur à la place de :idUser.

Réponse :

Les informations à propos du sondage.

Get "/users/surveys"
Permet de récupérer tous les sondages de la BD

Données attendues : aucune

Réponse :

Un tableau avec les informations de tous les sondages

Get "/users/surveys/:idSurvey/candidates"
Permet de récupérer les candidats d’un sondage

Données attendues : Indique l’id du sondage à la place de :idSurvey dans le lien

Réponse :

Un tableau contenant tous les candidats avec les infos leur concernant.

Get "/surveys/:idSurvey/hasVoted"
Permet de savoir si quelqu’un a déjà voté avec la même IP pour un sondage.

Données attendues :

L'ID du sondage à la place de :idSurvey dans le lien.

Réponse :

Une réponse Boolean en JSON.

Routes privées :

Get "/users/:idUser/games"
Permet de récupérer ses propres jeux

Données attendues :

L'ID de l'utilisateur à la place de :idUser dans le lien.

Réponse :

Tableau contenant l'ID et le nom de ses jeux

Get "/users/:idUser/games/:idGames"
Permet de récupérer les infos d'un jeu.

Données attendues :

L'ID de l'utilisateur ainsi que le l'ID du jeu dans le lien.

Réponse : Un objet JSON contenant toutes les infos du jeu.

Post "/users/:idUser/games"
Permet d'ajouter un jeu à sa collection.

Données attendues :

```
{
      gameName: "nom",
      description: "description",
      minAge: "age min",
      minNbPlayer: "min de joueur",
      maxNbPlayer: "max de joueur",
      duration: "durée",
      creationDate: "date de création",
      image: "URL",
}
```

Réponse :

Une confirmation que le jeu à été ajouté à la collection.

Put "/users/:idUser/games/:idGame"
Permet de modifier les infos à propos d'un jeu qui se trouve dans sa propre collection.

Données attendues :

```
{
	gameName: "nom",
	description: "description",
	minAge: "age min",
	minNbPlayer: "min de joueur",
	maxNbPlayer: "max de joueur",
	duration: "durée",
	creationDate: "date de création",
	image: "URL",
}
```

Réponse :

Une confirmation que le jeu a été modifié.

Delete "/users/:idUser/games/:idGame"
Permet de supprimer un jeu de sa collection.

Données attendues :

L'ID de l'utilisateur ainsi que l'ID du jeu à supprimer.

Réponse :

Une confirmation que le jeu à été supprimé.

Post "/users/:idUser/surveys"
Permet de créer un sondage.

Données attendues :

```
{
	candidates: [
		{
		idAPI : "idAPI",
		idGame : "idGame"
		}
	]
}
```

L'ID de l'utilisateur est à définir dans le lien.

Réponse :

L'ID du sondage créé est retourné.

Delete "/users/:idUser/surveys/:idSurvey"
Permet de supprimer un sondage

Données attendues :

L'ID du sondage dans le lien.

Réponse :

Confirmation que le sondage a été supprimé.

Get "/users"
Permet de récupérer la liste des utilisateurs.

Données attendues :

Aucune.

Réponse :

Un tableau contenant tous les utilisateurs.

Delete "/users/:idUser"
Permet de supprimer un utilisateur. Pour pouvoir l'utiliser il faut être superadmin.

Données attendues :

L'ID de l'utilisateur dans le lien.

Réponse :

Une confirmation que l'utilisateur est supprimé est retourné.

Sources – Bibliographie

Comment livrer en production notre application Angular ? - Dev to be curious

https://boardgamegeek.com/wiki/page/BGG_XML_API2

The ultimate guide to flat design » We Love Brisbane – Website Design

History | Vecteurs et Photos gratuites

27+ Board Game Pictures | Download Free Images on Unsplash

mhevery/jasmine-node: Integration of Jasmine Spec framework with Node.js

strawpoll – Recherche Google

Comment livrer en production notre application Angular ? - Dev to be curious

Zombicide | Board Game | BoardGameGeek

Colorlib | Free Bootstrap Website Template

20 Functional Bootstrap Tables to Organize Data - Colorlib

The ultimate guide to flat design » We Love Brisbane – Website Design

mysqljs/mysql: A pure node.js JavaScript Client implementing the MySQL protocol.

Testing with the Angular HttpClient API - Netscape - Medium

History | Vecteurs et Photos gratuites

27+ Board Game Pictures | Download Free Images on Unsplash

Adminca bootstrap 4 & angular 5 admin template, Шаблон админки | Login

mhevery/jasmine-node: Integration of Jasmine Spec framework with Node.js

Unit Testing a Node.js Application with the Jasmine Testing Framework

HTTP Status Codes

PUT vs POST - Comparing HTTP Methods - KeyCDN Support

11 JavaScript Animation Libraries For 2019 - Bits and Pieces

progressbar - ngx-bootstrap · Bit

angular6 - child parent communication best practices in Angular - Stack Overflow

Angular - Component interaction

Angular @ViewChild: In-Depth Explanation (All Features Covered)

https://dyma.fr/flutter?gclid=CjwKCAiAvonyBRB7EiwAadauqWiKvmIXjy5AzrvASUyQmLqD8yPZzwjtitA9pH2wsCAbhYeafXm5SxoC3cEQAvD_BwE

WebAIM: Contrast Checker

Box Shadow CSS Generator | CSSmatic

20 Creative Search Bar Design Inspirations with HTML/CSS/ Bootstrap

7 Practical Tips for Cheating at Design - Refactoring UI - Medium

7 Rules for Creating Gorgeous UI (Updated for 2020)

Angular Authentication: Using Route Guards - Ryan Chenkie - Medium

Using RxJS switchMap With Angular 7 Reactive Forms to Cancel Pending Requests - Credera

Pure CSS Loader - Optimized Spinners for Web · Loading.io

https://www.restapitutorial.com/httpstatuscodes.html

Journal de travail du projet

Date	Temps	Evénement
05.02.2020	1h	Réflexion, recherche de fonctionnalité
06.02.2020	1h	Travail avec Bastian sur le MCD de l'API
06.02.2020	1h	Travail avec Bastian sur le MLD de l'API
07.02.2020	1h30	Discussion avec M. Viret sur la BD et les fonctionnalités du site
07.02.2020	20 min	Réécriture du cahier des charges
07.02.2020	1h	Tentative de l'installation d'Adobe Xd + création de la maquette Balsamiq
07.02.2020	1h	Création d'un prototype du site avec Adobe Xd
07.02.2020	1h	Documentation
07.02.2020	1h	Création du Github ainsi que quelques issues
11.02.2020	1h	Création du Github et insertion des issues, projets et milestones
11.02.2020	1h	Ecriture des use cases et des scénarios
11.02.2020	1h	Finition de la maquette
12.02.2020	1h30	Html + css de la sidebar
13.02.2020	1h30	Html + css du header
13.02.2020	30min	Html + css de la barre de recherche
13.02.2020	30 min	Html + css du component pour afficher une liste de jeux
13.02.2020	15min	Html + css de la page collection & résultats d'une recherche
14.02.2020	1h30	Html + css de la page de login
14.02.2020	10min	HTML + css de la page register
25.02.2020	1h30	Création de la structure et des fichiers de base de l'API
25.02.2020	10min	Création du routeur et des différents liens
25.02.2020	30min	Html + css de la page de création de vote
26.02.2020	1h30	Création des routes avec Bastian, mais elles ne fournissent aucune données
27.02.2020	2h	Création de fonctions pour les routes /login /register /search-game-API + explication à bastian de comment ça fonctionne
27.02.2020	1h30	Création des tests automatique pour les routes /login /register /search-game-API
27.02.2020	10min	Création des issues pour l'API

28.02.2020	1h	Création des tests pour les routes /vote /get-games-collection /get-games-info-collection Ajout de la posibilité de fournir un id lors de la création d'un utilisateur
02.03.2020	3h	Intégration de jwt + route get et post survey + check de permission
03.03.2020	1h	Check de l'ip avant un vote
03.03.2020	15min	Création de la page admin
04.03.2020	1h	Ajout des fonctions login et register dans le frontend
04.03.2020	30min	Correction des erreurs retournées par l'API
04.03.2020	30min	Ajout d'un errorHandler pour les requêtes à l'API
05.03.2020	1h30	Ajout de la fonction de recherche
05.03.2020	15min	Ajout d'un pipe pour filtrer la recherche
05.03.2020	20min	Le pipe ne fonctionnait pas dans ce cas, j'ai donc ajouté un map directement sur l'observable pour trier les données
05.03.2020	5min	Correction d'un bug avec la page admin
05.03.2020	15min	Ajout de la route pour avoir tous les utilisateurs dans l'API
05.03.2020	20min	Ajout de la fonction suppression d'un utilisateur
05.03.2020	10min	Modification de la barre de recherche par un select dans la page admin
06.03.2020	45min	Création de la page de la liste de jeux après une recherche
06.03.2020	45min	Ajout de la fonction de suppression d'utilisateur à la page administrateur
06.03.2020	1h30	Ajout de la fonction d'affichage des détails d'un jeu
07.03.2020	1h30	Css + correction d'un bug dans le component gameDetails
07.03.2020	30min	Ajout d'un loading spinner
07.03.2020	10min	Ajout du spinner lors du chargement des détails
07.03.2020	20min	La collection
10.03.2020	1h30	Création du formulaire d'ajout d'un jeu dans sa collection + auto-remplissage lors d'une recherche
12.03.2020	1h30	Ajout de la fonction d'édition d'un jeu
16.03.2020	30min	Correction d'un bug lors d'ajout d'un candidat
16.03.2020	1h	Angular fonction de création de vote + fonction de vote
17.03.2020	45min	Le fonctionnement de la barre de progression pour les votes
17.03.2020	30min	Empêcher l'utilisateur de voter s'il a déjà voté
19.03.2020	15min	Correction des tests de l'API

20.03.2020	15min	Rédaction des tests effectué
24.03.2020	10min	Correction de BD pour la suppression en cascade
24.03.2020	1h	Ajout de la suppression d'un jeu dans la collection
25.03.2020	2h	Documentions de l'API
25.03.2020	1h	Correction des images dans le dossier + Points techniques spécifiques
27.03.2020	1h	Rédaction de la conclusion + correction de quelques fautes
28.03.2020	1h	Ajout de commentaire dans l'API pour docjs
28.03.2020	1h	Ajout de commentaire dans Angular pour docjs
02.04.2020	1h	Manuel d'installation + correction doc