

Dossier de projet

Oh My Game



Présenté et soutenu par

Olivier Stierer

En vue de l'obtention du

Titre Professionnel

Développeur web et mobile



Centre de Formation O'Clock

Promotion BAO

2023-2024

Introduction	4
I. Compétences du référentiel couvertes par le projet	5
1. Développer la partie front-end d'une application en intégrant les recommandations de sécurité	5
1.1. Maquetter une application	5
1.2. Réaliser une interface utilisateur web statique et adaptable	5
1.3. Développer une interface utilisateur dynamique	6
2. Développer la partie back-end d'une application web ou mobile en intégrant les recommandations de sécurité	6
2.1. Créer une base de données	6
2.2. Développer les composants d'accès aux données	7
2.3. Développer la partie back-end d'une application	7
II. Résumé du projet.....	8
III. Cahier des charges	9
1. Présentation du projet.....	9
1.1. Problématique	9
1.2. Solutions proposées.....	9
1.3. Objectifs de l'applications.....	10
1.4. Public ciblé	10
2. Minimum Viable Product.....	10
2.1. Améliorations possibles et souhaitées	12
3. Prévisualisation du site	12
4. Spécifications utilisateurs	15
4.1. Rôles utilisateurs.....	15
4.2. Les users stories	15
4.3. L'arborescence	16
5. Modélisation de la base données	16
IV. Spécifications techniques du projet	18
1. Versionning	18
2. Technologies utilisées	19
2.1. Coté front.....	19
2.2. Coté back	20
3. Identification des besoins de sécurité	21
3.1. Gestion des inputs et attaques XSS :	21
3.2. Gestion des rôles et des droits :	22

3.3. L'authentification	22
3.4. La gestion des CORS.....	23
3.5. Gestion des mots de passe	23
V. Gestion du projet	24
1. Présentation de l'équipe.....	24
2. Partage des rôles	24
3. Méthodologie de travail	25
4. Outils de travail.....	25
VI. Réalisations personnelles	27
1. Scrapy.....	27
2. La messagerie	33
VII. Jeu d'essai d'une fonctionnalité	40
VIII. Veille effectuée sur les vulnérabilités de sécurité	43
IX. Situation de travail ayant nécessité une recherche	45
1. Le problème rencontré	45
2. La recherche effectuée	45
3. La source anglophone utilisée	45
Conclusion.....	46
Annexes.....	48

Introduction

Enseignant pendant de nombreuses années en école associative de musique, je me suis formé au CNAM de Nantes en **Java** par gout sans réelles intentions de changer de voie. Durant l'année 2022-2023, je suis une formation en **PHP** toujours au **CNAM**.

Cette dernière formation sera pour moi un point de bascule et me poussera à faire le choix d'une reconversion professionnelle dans le développement. Je fais alors le choix d'une formation Fullstack chez O'CLOCK pour compléter mes connaissances afin de passer le titre Développeur Web et Mobile.

Ce projet "Oh My Game" réalisé à la fin de mon cursus m'a permis de vivre une situation professionnelle durant laquelle j'ai pu être au plus proche de la réalité de la conception d'une application.

I. Compétences du référentiel couvertes par le projet

1. Développer la partie *front-end* d'une application en intégrant les recommandations de sécurité

1.1. Maquetter une application

Lors de la phase conception de l'application, nous avons réalisé un ensemble de *wireframes* afin de pouvoir en représenter schématiquement l'interface. Figurent ainsi l'emplacement des différents éléments ainsi que les différentes interactions qui seront disponibles pour l'utilisateur. Chaque membre de l'équipe a eu la charge de concevoir les *wireframes* d'une page. Nous avons réalisé dans le même temps les *user stories* qui déterminent les différentes fonctionnalités du site.

1.2. Réaliser une interface utilisateur web statique et adaptable

À partir des prototypes et d'une réflexion de groupe sur la charte graphique à implémenter, les différentes interfaces utilisateurs ont été réalisées, avec prise en compte des aspects de *responsive design*. Ainsi il est possible d'afficher les mêmes contenus quel que soit la taille du *device*. D'abord pensées pour le mobile et elles ont été ensuite déployées dans leur version desktop. Nous avons principalement utilisé les *breakpoints* de **Bootstrap**.

1.3. Développer une interface utilisateur dynamique

Afin de répondre aux besoins d'interactivité et de manipulations dynamiques des pages de l'application, nous avons utilisé **Javascript** ainsi que le client http **Axios**. Ainsi nous avons pu modifier le DOM à certains endroits par l'ajout ou la suppression d'éléments html. De la même façon il nous a été possible d'aller chercher des données et d'afficher ces dernières sans avoir besoin de recharger certaines pages. La messagerie ainsi que la carte affichant les utilisateurs de l'application en sont des exemples. En effet lors de l'inscription d'un utilisateur nous faisons appel à une API publique afin d'obtenir les coordonnées GPS de ce dernier à partir de son adresse afin de pouvoir le localiser sur la carte de l'application.

2. Développer la partie *back-end* d'une application web ou mobile en intégrant les recommandations de sécurité

2.1. Créer une base de données

Le cœur de l'application "Oh My Game" est sa base de données. Cette dernière a été conçue suivant les étapes suivantes :

- Le MCD, modèle conceptuel de données
- Le MLD, modèle logique de données
- Le MPD, modèle physique de données

S'ajoute à cela la création d'un dictionnaire de données.

Cette base de données a été déployé avec l'**ORM Doctrine** à partir de migrations faites selon les entités de l'application. Le choix du système de gestion de la base s'est porté sur **MariaDb**, et **PhpMyAdmin** ou **Adminer** ont été retenus pour l'administrer. Un jeu de

données a été importé grâce à un outil de *scrapping* que j'ai développé en **PHP** afin de peupler la base.

2.2 Développer les composants d'accès aux données

Afin d'interagir avec la base et d'accéder aux données nous nous sommes servis de **Doctrine**, l'**ORM** par défaut de **Symfony**. Cet **ORM**, *Object-relational Mapping*, va définir des correspondances entre les schémas de la base de données et les objets de l'application. Ainsi à la création d'une entité dans **Symfony**, à l'aide d'une commande du *maker* par exemple, un *repository* se crée, qui contient un ensemble de méthodes permettant d'interagir avec l'entité dont il est issu.

Ces méthodes vont permettre d'agir en lecture, insertion, modification, suppression (principe du *C.R.U.D*). Nous avons procédé à l'ajout de méthodes "custom" afin d'obtenir des résultats selon nos besoins (critères de tri, nombre de résultats, jointure avec d'autre tables, etc ...). Nous avons écrit la plupart de ces méthodes en **DQL** (*Doctrine Query Language*) qui est très similaire au **SQL** (*Structured Query Language*) à la différence qu'il se réfère aux objets et leurs méthodes plutôt qu'aux tables et colonnes de la base. L'accès aux données se fait parfois aussi via l'*EntityManager* de **Symfony** ; en utilisant uniquement ce composant de **Doctrine** on optimise l'interaction avec la base.

2.3. Développer la partie *back-end* d'une application

L'application a été conçue selon le *design pattern* **MVC** avec **Symfony**. Elle respecte la division en trois couches :

- Modèles, Vues, Contrôleurs

Dans la partie précédente nous avons décrit comment la partie Modèle avait été développée.

La partie contrôleur se divise en deux avec les contrôleurs ayant une route API et les autres. Tous les contrôleurs font appel à des méthodes de classes issues de l'entité ou bien du repository associé. Selon le type de contrôleur, la réponse sera soit un **JSON** ou l'appel à une vue **Twig**. L'ensemble des *templates Twig* constitue la partie Vue de l'application.

Afin de sécuriser convenablement "Oh My Game", nous avons mis en place un système d'authentification, configurer l'*access control list* pour la gestion des rôles, créer des *voter*, ajouter des contraintes de validations, gérer les **CORS** et configurer la session. La sécurité fait l'objet d'un chapitre ultérieur.

II. Résumé du projet

Les jeux de société ont fait un retour en force spectaculaire ces dernières années. Bien plus qu'une simple activité de loisirs, ils sont devenus un véritable phénomène sociétal. Cependant il reste difficile de rencontrer de nouveaux joueurs et le prix des jeux étant souvent élevé on souhaiterait pouvoir les essayer avant de les acheter.

C'est pourquoi j'ai proposé "Oh My Game", une application qui permettrait à des amateurs de jeux de société de s'échanger des jeux et de se rencontrer.

Tout d'abord, l'application permet d'accéder à une large base de données de jeux ainsi qu'aux détails de ces derniers que l'on soit visiteur ou utilisateur. Il est possible de se créer un espace personnel dans lequel on peut renseigner les jeux que l'on possède, et les rendre visibles aux autres utilisateurs de l'application.

On peut alors prêter ses jeux ou en emprunter, trouver des joueurs près de chez soi et communiquer avec eux via la messagerie. On peut aussi créer des après-midis ou des soirées jeux chez soi autour d'un thème ou d'un jeu en particulier.

"Oh My Game" est dotée d'un espace d'administration qui permet la gestion de la base de données, des rôles utilisateurs ainsi que la modération des commentaires.

Réalisée par une équipe de quatre développeurs sur une durée d'un mois, l'application s'appuie sur **Twig**, **Bootstrap**, **Axios**, **Webpack-encore** pour sa partie *front-end* et sur **Symphony 6.4**, **PHP 8.1**, **adresse.data.gouv**, **open street map** et **Mercure** pour sa partie *back-end*.

III. Cahier des charges

1. Présentation du projet

1.1. Problématique

On constate un fort regain pour les jeux de société depuis quelques années mais peu ou pas d'applications existent qui accompagnent ce phénomène.

En outre si des applications qui référencent des jeux existent, c'est surtout pour mettre en avant un réseau de distributeurs. Quid des applications dédiées aux joueurs qui permettraient de rencontrer de nouveaux joueurs et d'essayer de nouveaux jeux.

1.2. Solutions proposées

Le projet "Oh My Game" a pour objectif de développer une application web permettant aux joueurs de jeux de société d'entrer en relation avec d'autres joueurs à proximité de leur localisation afin de partager les jeux en possession de chacun.

1.3. Objectifs de l'application

- Permettre aux visiteurs et aux utilisateurs de consulter la base de données de l'application et d'accéder aux détails d'un jeu.
- Permettre à l'utilisateur de se constituer une "Bibliogame", liste de jeux en sa possession, d'après une Bibliothèque préétablie et gérée par l'administrateur du site.
- Permettre la diffusion de cette collection auprès des autres utilisateurs de l'application
- Permettre l'emprunt et le prêt de jeux
- Permettre aux utilisateurs de se contacter
- Permettre à l'utilisateur de trouver des joueurs près de chez lui grâce à une carte interactive.

1.4. Public ciblé

Oh My Game s'adresse à un public d'amateurs de jeux de sociétés de tous âges, souhaitant partager, échanger et se rencontrer.

Elle s'adresse également aux professionnels du monde du jeu de société souhaitant promouvoir leurs événements.

2. Minimum Viable Product

Voici la liste des fonctionnalités que nous avons définies comme étant nécessaires au MVP :

- Accès à la liste de tous les jeux, avec la possibilité de les trier selon Titre, Age, Année de sortie, éditeurs, Types et Thèmes. Pour l'utilisateur connecté, possibilité d'ajouter des jeux à sa collection.
- Accès aux détails d'un jeu : titre, auteur, une description longue, les types, les thèmes, les auteurs, l'éditeur, les illustrateurs, l'âge, la date de sortie, le nombre de joueurs, une carte qui affiche les joueurs qui possèdent le jeu et qui sont proches de l'utilisateur lorsque l'on est connecté ou tous les joueurs qui possèdent le jeu en France.
- Formulaire d'inscription
- Formulaire de connexion
- Accès à une page de profil pour l'utilisateur connecté avec la possibilité de :
 - Modifier, supprimer son profil
 - Voir les jeux qu'il emprunte, les jeux qu'il prête
- Accès à une page "Bibliogame" qui regroupe tous les jeux que l'utilisateur possède et qu'il a renseigné. Il peut retirer des jeux de sa collection, accéder à la page de détail d'un jeu
- Accès à une page carte qui affiche tous les joueurs proches de l'utilisateur. Possibilité d'accéder au profil public des autres joueurs en cliquant sur leur marqueur sur la carte
- Accès au profil public des autres joueurs sur lequel sont affichés les jeux de leur collection avec la possibilité de les contacter (lien vers la messagerie)
- Page d'erreur 404
- Espace d'administration du site avec la possibilité d'ajouter, supprimer, mettre à jour un jeu.

- Formulaire de contact afin que visiteurs ou utilisateurs puissent soumettre des questions ou signaler des problèmes à l’administration du site.

2.1. Améliorations possibles et souhaitées

- Possibilité de créer un après-midi ou une soirée jeu chez soi. De pouvoir gérer les demandes de participations. Pouvoir modifier ou supprimer un événement à soi.
- Pouvoir participer à un après-midi ou une soirée jeux chez un autre utilisateur ou chez un professionnel (bar à jeux, magasin, etc ...)
- Possibilité pour un utilisateur de changer de mot de passe à partir de sa page de profil
- Fonctionnalité de récupération de mot de passe quand on a perdu son mot de passe
- Afficher les événements sur la carte
- Uploader une image de profil
- Améliorer la gestion des prêts et des emprunts en ajoutant des notifications par mail ainsi que la possibilité de faire des demandes de réservations
- Mettre en place une messagerie en temps réel
- Rendre un jeu disponible uniquement en après-midi ou soirée jeu
- Commenter et noter un jeu

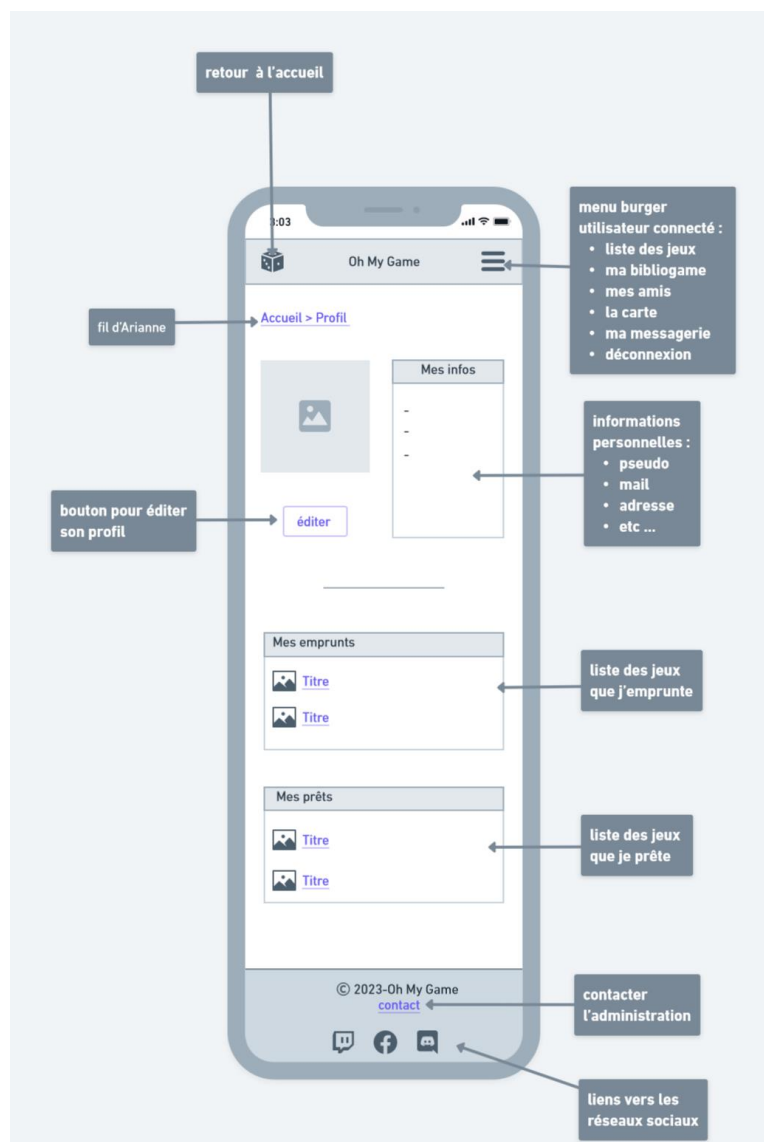
3. Prévisualisation du site

Afin de prévisualiser le site et l’accessibilité aux différentes fonctionnalités définies dans le MVP, nous avons élaboré des *wireframes*. Elles ont été créées de façon responsive, avec une version Desktop et une version Mobile pour chaque vue. Ces *wireframes* nous ont

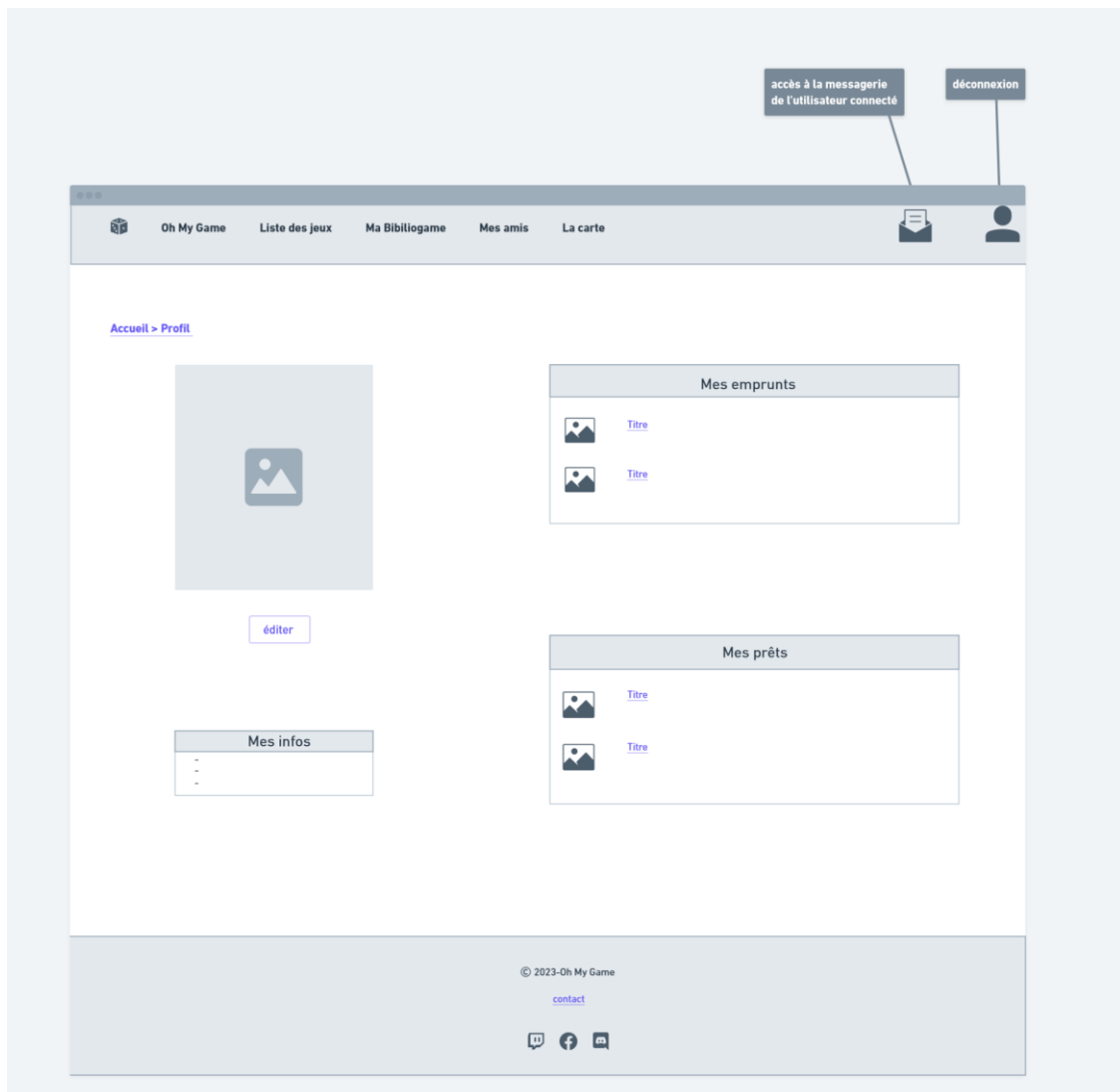
permis d'avoir une idée globale de comment structurer notre code HTML et les éléments de **Bootstrap** dans nos vues **Twig**.

La conception des *wireframes* a été répartie entre les différents membres de l'équipe. Voici les *wireframes* de la page profil, réalisés par mes soins :

Version mobile :



Version desktop :



4. Spécifications utilisateurs

4.1. Rôles utilisateurs

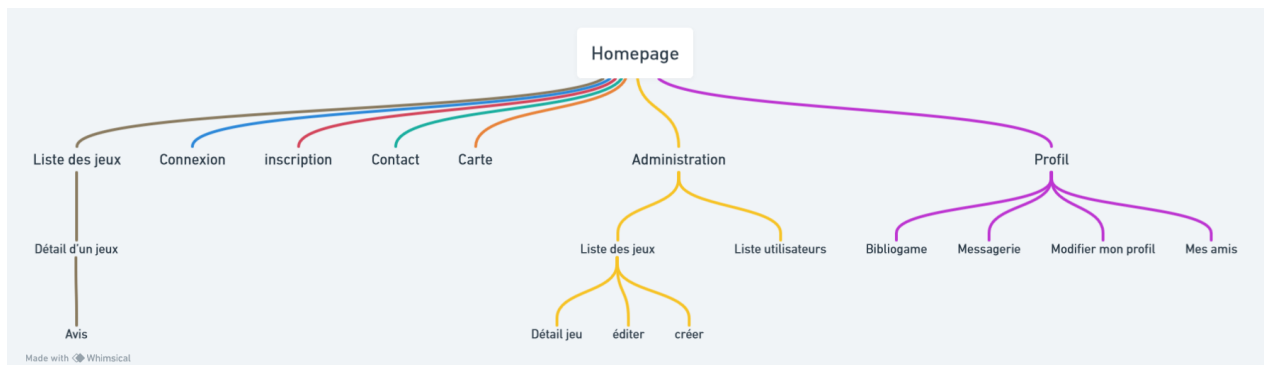
Il y a trois types d'utilisateurs différents pour notre application :

1. Les utilisateurs non connectés, visiteur, qui ont accès à la page d'accueil, la liste des tous les jeux et leur détail. Ils peuvent voir l'ensemble des joueurs et des événements "Oh My Game" (selon la version) en France sans pour autant pouvoir accéder au détail.
2. Les utilisateurs connectés ont accès à l'ensemble des fonctionnalités hors administrations.
3. L'administrateur a accès au *back-office* de l'application et peut gérer les jeux et les utilisateurs ainsi que les commentaires (selon version)

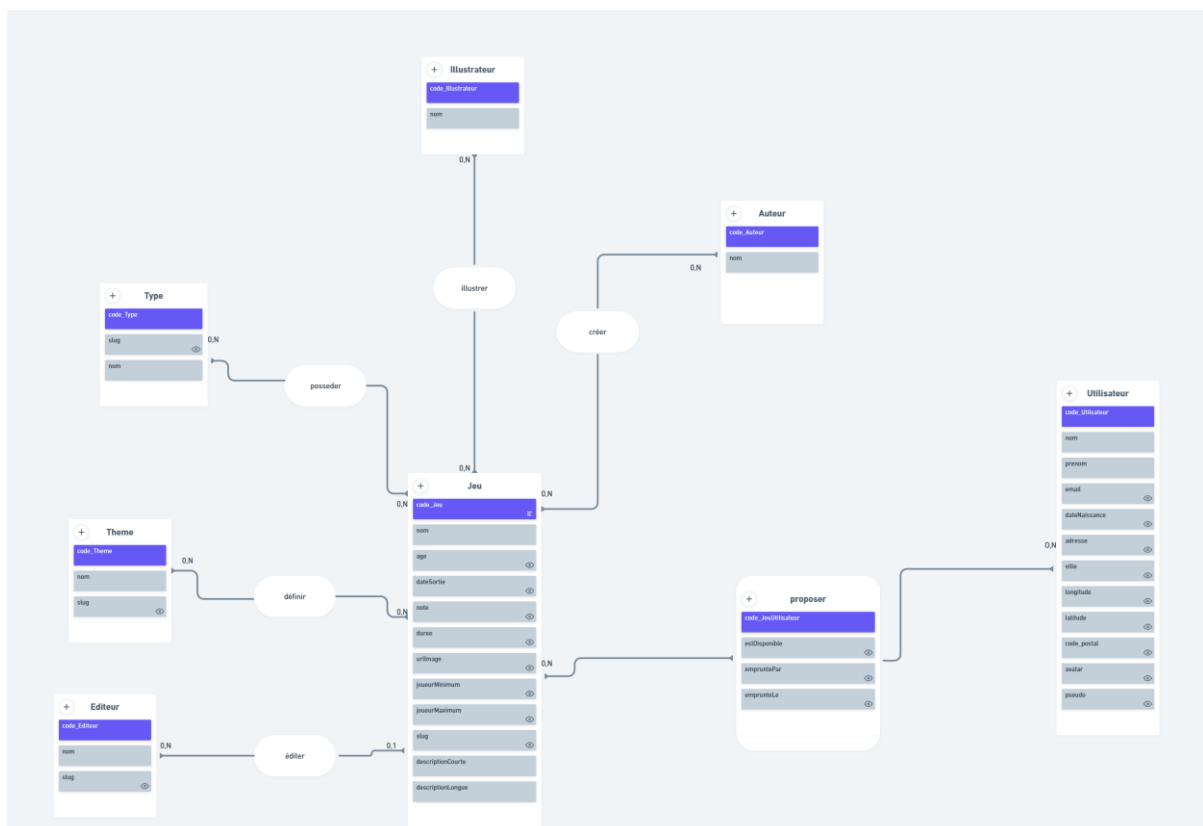
4.2. Les users stories

EN TANT QUE	JE VEUX	AFIN DE
Visiteur	pouvoir consulter la liste de tous les jeux	trouver des jeux que je n'ai pas
Visiteur	pouvoir me créer un compte	d'accéder à toutes les fonctionnalités de l'app
Visiteur	pouvoir accéder au détail d'un jeu	découvrir ses caractéristiques
Visiteur	pouvoir aux avis d'un jeu	me faire un avis sur le jeu
Visiteur	pouvoir accéder à un jeu au hasard	découvrir de nouveaux jeux
Visiteur	rechercher des jeux par critères	afin de trouver des jeux qui correspondent le mieux à mes préférences
Visiteur	consulter la carte des utilisateurs	d'apprécier la taille de la communauté des joueurs à proximité
Visiteur	contacter l'administrateur du site	résoudre un problème, poser une question
Utilisateur	pouvoir créer ma bibliogame	diffuser la liste des jeux que je possède
Utilisateur	ajouter un jeu à ma bibliogame	faire évoluer ma collection
Utilisateur	retirer un jeu de ma bibliogame	faire évoluer ma collection
Utilisateur	pouvoir accéder aux profils des autres utilisateurs	pouvoir leur emprunter des jeux
Utilisateur	pouvoir contacter un joueur	pouvoir parler avec lui des conditions du prêt, échanger sur divers sujets
Utilisateur	éditer mon profil	changer d'adresse, avatar
Utilisateur	supprimer mon profil	supprimer mes données personnelles de l'application
Modérateur	consulter les avis d'un jeu	vérifier le contenu d'un avis
Modérateur	supprimer un avis	pouvoir modérer un avis
Admin	modifier le rôle d'un utilisateur	de le faire passer de user à modérateur à admin et vice versa
Admin	bloquer un utilisateur	lui interdire l'accès à l'application
Admin	créer un jeu	ajouter un jeu à la base de données
Admin	éditer les infos d'un jeu	corriger des données
Admin	supprimer un jeu	supprimer de la base de données

4.3. L'arborescence



5. Modélisation de la base données



Nous avons réalisé le modèle conceptuel de données pour notre base, y figurent 7 entités ainsi qu'une table relationnelle porteuse de données. On constate que toutes les relations ont une cardinalité minimum de 0. En effet même si à terme on souhaite avoir tous les champs renseignés pour un jeu (l'auteur, l'illustrateur, etc ...) on ouvre la possibilité à l'administrateur de ne créer que l'objet jeu en premier lieu. Ainsi il pourra s'occuper des relations plus tard ; il gagne ainsi en souplesse et confort pour l'administration de la base. La base a été déployée sur le serveur via les commandes de Doctrine :

The image shows two terminal window screenshots side-by-side. Each window has a title bar with three colored dots (red, yellow, green). The left window contains the command `php bin/console doctrine:database:create`. The right window contains the command `php bin/console doctrine:migrations:migrate`.

```
php bin/console doctrine:database:create
```

```
php bin/console doctrine:migrations:migrate
```

Il est possible de consulter un extrait du script de création de la base de données en annexe. Il est important de constater que certaines contraintes posées sur des clés comportent la valeur DELETE ON CASCADE. Cela permettra d'éviter les violations d'intégrité référentielle lors de la suppression d'une occurrence d'entité. On constate aussi la création d'index qui vont permettre l'accélération des requêtes SQL.

La création de la base s'est accompagnée de la création d'un dictionnaire de données ainsi que de la création d'un tableau regroupant l'ensemble des routes de l'application. Ces fichiers étant assez volumineux, ils figurent en annexe.

IV. Spécifications techniques du projet

1. Versionning

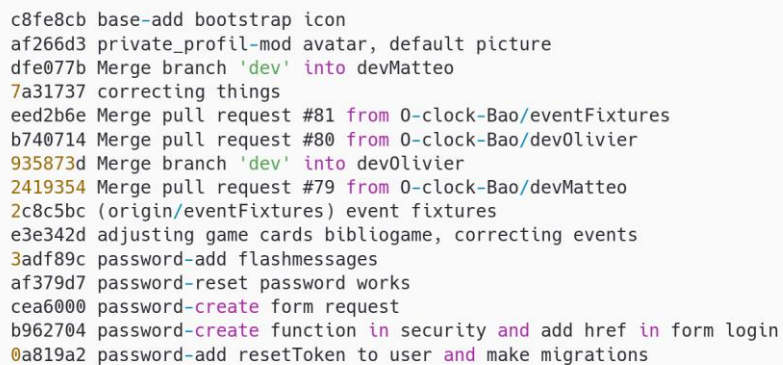
Afin de rendre la collaboration plus efficiente entre tous les membres de l'équipe, nous avons utilisé **GIT** et avons appliqué une méthodologie commune.

Nous avons un seul *repository*, l'application n'étant pas scindée en deux comme on peut le rencontrer souvent quand l'application est pensée avec une partie *back* API développée par une équipe et une partie *front* développée par une autre équipe. Sur ce *repository*, hébergé sur Github, nous avons créé une branche MAIN, sur laquelle seraient *merge* les versions prêtes à être déployées sur le serveur. Nous ne codions jamais dessus.

Une branche DEV a été créée à partir de MAIN, et chaque membre de l'équipe a pu tirer une branche perso à partir de DEV, suivant la convention de nommage suivante : "dev+NomDuDéveloppeur". À partir d'elle nous tirions des branches pour chaque nouvelles *features*, puis nous mergions sur notre branche dev personnelle. Nous pouvions ensuite *push* et créer une *pull request* que notre git master validait. Nous opérions le plus souvent possible une *code review* en équipe avant chaque PR.

Nous nous sommes donnés comme consigne de faire des *commits* de façon régulière afin de gagner en clarté. Pour la même raison la rédaction des *commits* obéissaient tous la même convention d'écriture, décidée en équipe. Chaque *commits* devaient commencer par le nom de la fonctionnalité associée, un tiret, puis le détail de ce qui a été fait. Toutes les branches et les *commits* étaient en anglais afin d'être au plus proche des conditions d'entreprise.

Afin de mieux se figurer les relations entre branches et les conflits pouvant émerger, le nombre de branches et de *commits* devenant de plus en plus important, nous avons parfois utilisé une version graphique de git avec **GitKraken**.



```
c8fe8cb base-add bootstrap icon
af266d3 private_profil-mod avatar, default picture
dfe077b Merge branch 'dev' into devMatteo
7a31737 correcting things
eed2b6e Merge pull request #81 from 0-clock-Bao/eventFixtures
b740714 Merge pull request #80 from 0-clock-Bao/devOlivier
935873d Merge branch 'dev' into devOlivier
2419354 Merge pull request #79 from 0-clock-Bao/devMatteo
2c8c5bc (origin/eventFixtures) event fixtures
e3e342d adjusting game cards bibliogame, correcting events
3adf89c password-add flashmessages
af379d7 password-reset password works
cea6000 password-create form request
b962704 password-create function in security and add href in form login
0a819a2 password-add resetToken to user and make migrations
```

2. Technologies utilisées

2.1. Coté front

“Oh My Game” est une application conçue selon le *design pattern* MVC. Pour gérer l’affichage des vues et donc le coté *front* de l’application nous avons utilisé le moteur de *templates* open source par défaut de **Symfony**, **Twig**. Il est très simple à installer et à prendre en main. Il compile les *templates* en un code **PHP** optimisé, réduisant ainsi la surcharge associée à l’interprétation du code **PHP** brut incorporé dans les *templates*.

Nous avons utilisé **Bootstrap**, une librairie HTML/CSS/JS, qui nous a facilité le placement des différents éléments des pages grâce à l’utilisation du système de grille, avec colonne et ligne, qu’il propose. Cette librairie nous a aussi permis de rendre les pages responsives, **Bootstrap** incluant un certain nombre de *breakpoints*. En outre nous avons pu profiter des tous les composants inclus tel que les boutons, les formulaires, le inputs, etc ...

Afin de pouvoir personnaliser **Bootstrap**, nous nous sommes servis de **WEBPACK**, et avons installé **NPM**, le gestionnaire de paquets, ainsi que **Node.js**.

L'intégration de carte s'est faite avec les objets *map* de **Leaflet**, la bibliothèque JavaScript open source de cartographie.

Nous avons utilisé le client http **AXIOS** afin de dynamiser un certain nombre de pages de l'application. Il a permis de réaliser un ensemble de requêtes GET ou POST en ajax vers des API publiques ou notre API privée afin d'accéder aux données. Il nous a permis aussi l'affichage de ces dernières, par manipulation du DOM en supprimant ou en ajoutant des éléments, sans avoir à recharger les pages

Pour une interactivité renforcée nous avons mis en place une messagerie en temps réel avec **Mercure**. Ecrit en GO, basé sur le protocole http et la norme *server sent event*, c'est un protocole permettant de transmettre en temps réel des mises à jour de données vers les navigateurs web (ou autres clients HTTP) de manière fiable, rapide et économe en énergie.

2.2. Côté back

La partie *back-end* a été réalisée à l'aide du framework **PHP Symfony** et **Composer** pour l'importation de composants, comme le **MakerBundle** qui nous a permis de générer rapidement des contrôleurs, des classes, des formulaires, etc, en une ligne de commande.

Des fixtures ont été créées afin de peupler facilement la base avec **Faker**

L'**ORM Doctrine** a été utilisé pour la gestion de la base de données, et un ensemble de C.R.U.D a été développé afin d'interagir en lecture, ajout, mise à jour et suppression.

Un ensemble de contrôleurs a été développé afin d'appeler les méthodes de classes, d'opérer une logique et afficher les vues.

Des services ont été aussi créés. "Oh My Game" comprend un service qui permet l'initialisation des coordonnées GPS d'un utilisateur à partir de son adresse via l'api "adresse-data.gouv". Elle comprend aussi un service de mailing basé sur le **Mailer** de **Symfony**, afin de

pouvoir notifier les utilisateurs par mail. L'application possède aussi service pour uploader les images avec notamment des fonctions de validation de type et de redimensionnement.

Nous avons aussi développé des *listeners* qui vont permettre de mettre à jour automatiquement des attributs d'une entité lors de modifications de cette dernière. Ils sont liés à des événements **Doctrine** de type *PreUpdate* ou *PrePersist* par exemple.

J'ai aussi développé un service de *scrapping* écrit en PHP pour alimenter la base de données, il utilise le composant **BrowserKit**. Des commandes personnalisées ont été créées afin d'utiliser ce service de *scrapping*.

3. Identification des besoins de sécurité

La sécurisation des applications web est un sujet de plus en plus crucial. C'est pourquoi il est essentiel de se protéger face aux principales failles de sécurité et aux possibles attaques malveillantes qui pourraient porter atteinte à l'intégrité des données de l'application ou à la sécurité des données de l'utilisateur. Pour répondre à ces problématiques "Oh My Game" s'appuie sur un ensemble de solutions :

3.1. Gestion des inputs et attaques XSS :

Une des attaques possibles est l'injection de scripts malveillants via les formulaires. Nommés *cross-site scripting*, ou XSS, ces attaques peuvent permettre d'accéder à des données normalement inaccessibles, et de compromettre la base de données. Pour éviter tout risque d'injection, nous avons en place un ensemble de mesures.

Des contraintes de validations ont été placées dans les formulaires et dans les entités. Ainsi il n'est pas possible de soumettre un formulaire ou de rentrer en base une entité qui aurait un champ d'un mauvais type. Nous sommes en outre aidés dans cette entreprise de "nettoyage" des inputs par **Twig** qui échappe automatiquement les entrées

ainsi tout HTML ou caractère spécial est encodé, tout script HTML ou Javascript malveillant est rendu inoffensif.

Nous avons aussi privilégié l'utilisation de requêtes préparées. En séparant les paramètres des instructions SQL, le système de base de données va distinguer clairement les données et code SQL.

3.2. Gestion des rôles et des droits :

Une fois le *firewall* passé et l'utilisateur connecté il convient de déterminer quels sont ses droits et permissions d'accéder à certaines fonctionnalités de l'application. Un utilisateur ne doit pas avoir la possibilité d'accéder à la partie d'administration par exemple s'il ne possède pas les droits adéquats. C'est pourquoi nous avons configuré le fichier *security.yaml* en déterminant dans l'ALC (*access control list*) quelles urls sont accessible selon les rôles. Nous avons déterminé deux rôles : le `ROLE_USER` et le `ROLE_ADMIN`. Ainsi lors de l'appel d'une route, les expressions régulières renseignés dans l'ACL vont permettre d'identifier le type de route et le rôle associé.

Nous avons aussi mis en place plusieurs *voter* afin d'aller encore plus loin dans la gestion des droits. Ainsi il n'est pas possible de changer le statut d'un jeu emprunté si l'on n'est pas le propriétaire du jeu. De la même façon, un commentaire fait sur un jeu ne peut être modifié que par son auteur.

3.3. L'authentification

Afin de sécuriser l'accès à l'application nous avons mis en place une authentification custom. Lors de la tentative de connexion, la première étape est de passer le *firewall*. Ici ce dernier va chercher l'utilisateur en base de façon sécurisée grâce au service *user_provider* à partir de l'email donné. Ceci fait le service *UsersAuthenticator* est appelé et va déployer un ensemble de vérifications, parmi lesquelles la validation d'un *token* CSRF. Ce dernier est la réponse adéquate afin de parer les attaques de type *Cross-site Request Forgery* qui vont

tenter d'usurper l'identité de l'utilisateur. Ce jeton est créé à la soumission du formulaire de login, grâce à la fonction **Twig** `csrf_token()` et sera récupéré via l'objet *Request* par la fonction *Authenticate()* du service quiinstanciera un objet *Passeport* dont le *token* sera passé en paramètre. Si cet objet *Passeport* est valide alors la connexion est validée. En outre nous avons configuré le fichier *framework.yaml* afin de s'assurer que la session et l'ensemble des cookies seraient bien détruits à la fermeture du navigateur quand bien même l'utilisateur ne se serait pas déconnecté.

3.4. La gestion des CORS

Les CORS (*Cross-origin Ressource Sharing*) est un mécanisme qui définit comment un navigateur et un serveur peuvent interagir ensemble. La politique de sécurité de même origine ne permet pas à une page web d'utiliser des ressources provenant d'un domaine différent. Les autorisations CORS permettent le partage de ressources sécurisées entre des navigateurs et serveurs web d'origines différentes. Ainsi il est possible de consommer certaines API. Afin d'autoriser l'accès à notre application à partir d'origines différentes du site courant, nous avons utilisé le composant **NelmioCorsBundle** que l'on peut configurer dans le fichier *nelmio_cors.yaml*. Des variables d'environnement définies dans le point *.env* de l'application sous la forme d'expressions régulières y sont appelées. Elles sont les urls autorisées. Ce bundle fonctionne avec l'ajout d'en tête (headers) CORS dans les requêtes/réponses.

3.5. Gestion des mots de passe

Lors de l'inscription d'un nouvel utilisateur, celui-ci renseignera un mot de passe et à la soumission du formulaire le *UserRegisterService* est appelé. Ce dernier va notamment crypter le mot de passe en faisant appel à la méthode *haspasswords()* de l'objet *UserPasswordHasherInterface*. Dans le fichier *security.yaml*, l'option *password_hasher* est 'auto'. Ainsi l'algorithme le plus sécurisé est choisi automatiquement pour encoder les mots de passe avant d'être enregistrés en base.

v. Gestion du projet

1. Présentation de l'équipe

Le projet a été réalisé par une équipe de quatre personnes, dont la composition est le fruit d'un tirage au sort. Le hasard a voulu que nous soyons tous de la même spécialité orientée *back-end*, ce qui aura comme conséquence de décider pour nous du mode de conception de l'application ainsi que de l'organisation du travail et la répartition des tâches.

2. Partage des rôles

Afin d'optimiser l'organisation et de favoriser la collaboration nous, nous avons suivi la méthode Scrum et nous nous sommes attribué différents rôles :

Product Owner : moi-même. En effet étant à l'origine du projet, j'étais celui qui à priori avait le plus réfléchi aux besoins fonctionnels du projet. Si la *roadmap* a été décidée en équipe et si l'ajout de nouvelles fonctionnalités faisait aussi l'objet d'un débat entre nous, j'étais celui qui décidait pour l'équipe en cas conflit.

SCRUM Master : c'est Mattéo qui s'est porté volontaire pour prendre en charge la gestion du projet et gérer la communication au sein de l'équipe.

GIT Master : c'est Matty qui s'est portée volontaire. Ayant une expérience de plusieurs années en entreprise dans le développement, elle était la plus à même d'assumer ce rôle crucial.

Lead dev : c'est Nicolas qui s'est porté volontaire.

Il est important de noter que si le choix des rôles s'est fait à priori, l'organisation n'est pas restée la même durant la réalisation. En effet mon expérience dans la gestion de projet

et le management d'une équipe m'a amené assez vite à prendre le rôle de SCRUM Master. De la même façon l'expertise technique de Mattéo l'a amené à devenir notre lead dev.

3. Méthodologie de travail

Notre période de travail de quatre semaines a été divisée en autant de sprints.

Le sprint 0 était pour nous la phase de conception de l'application. Le cahier de charges, les *wireframes*, l'arborescence du site, le modèle conceptuel de données, les *users stories* et les routes ont été réalisés. C'était aussi l'occasion pour nous de penser le design et la charte graphique. Nous en avons profité pour mettre à jour et normaliser l'environnement de travail de tout le monde. Ainsi chaque développeur a travaillé avec **LAMP**, **Symfony 6.3** et **PHP 8.1**. Les tâches à effectuer sont décidées ensemble lors du *daily meeting* et les journées alternent entre moments passés en autonomie ou travail en groupe.

Les sprints 1 et 2 correspondaient à la phase de réalisation. Chaque journée débutant de la même façon, avec un *daily meeting*, pendant lequel nous commençons par récupérer la branche DEV du projet. S'en suivait un moment de résolution des conflits git. Nous procédions ensuite au bilan de la *code review* effectuée la veille par un des membres de l'équipe. Les corrections à apporter constituant ainsi la première partie des tâches à effectuer. Nous décidions ensuite des tâches pour chacun. Nous restions constamment à disposition des uns des autres afin de pouvoir aider en cas de besoin. En procédant ainsi nous avons pu adapter la charge de travail aux capacités de chacun et dimensionner le projet au fur et à mesure. C'est aussi à cette période que nous effectuons la mise en ligne de l'application.

Le dernier sprint nous a permis de travailler sur le design et de procéder à la résolution de bugs apparus lors de nos sessions en commun de tests fonctionnels. Nous ne développons plus aucune nouvelle fonctionnalité et nous mettons en ligne les dernières versions de l'application.

4. Outils de travail

Afin d'avancer de façon optimale, nous avons utilisé différents outils de travail, particulièrement bien adaptés au travail à distance.

- Tous les documents de gestion de projet, comme le cahier des charges, étaient stockés et mis à jour sur un **Drive** commun. Ainsi chaque membre du groupe avait accès aux dernières versions des *wireframes* ou du MCD et du dictionnaire de données par exemple.
- Nous avons mis en place un **Trello** qui listait toutes les tâches à faire, en cours ou finies, avec pour chacune d'entre elles, le membre du groupe chargé de sa réalisation, la version correspondante (MVP, V.2, etc ...), sa priorité.
- Nous nous retrouvions tous les matins sur **Discord** afin de réaliser le *daily meeting* et sur un canal **Slack** dédié à l'équipe. **Discord** nous permettait de streamer notre code afin de le soumettre aux autres membres de l'équipe, soit pour une résolution de bug ou une optimisation. De la même façon nous pouvions ainsi présenter aux autres le visuel de nos pages et leurs interactions.
- Comme mentionné dans le paragraphe sur le *versionning*, nous avons une méthodologie **Git** commune.

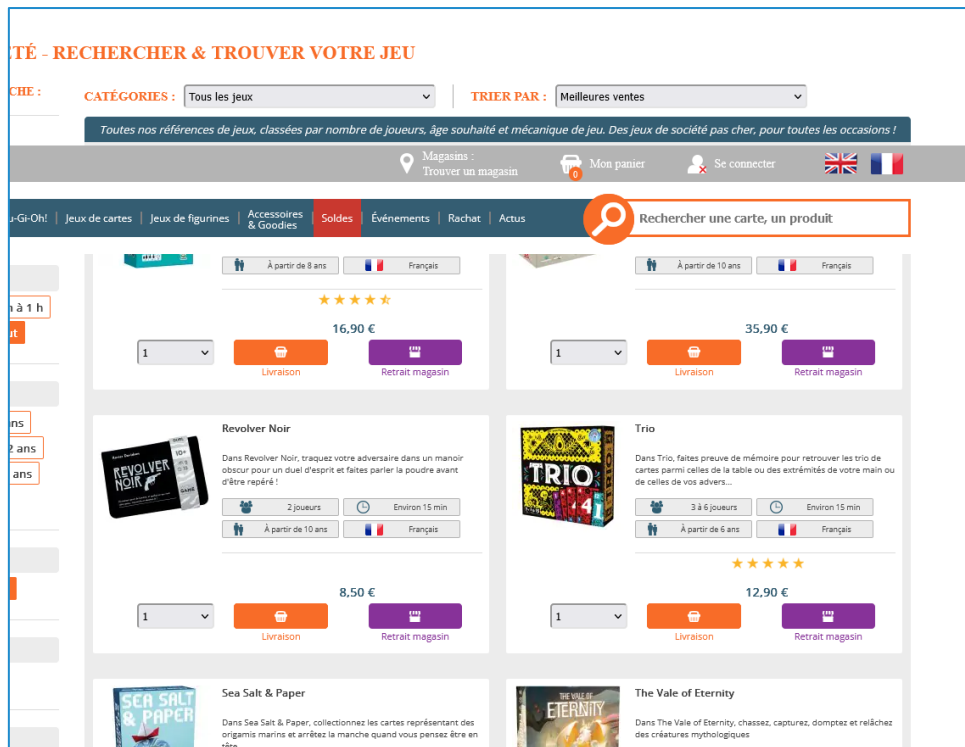
vi. Réalisations personnelles

1. Scrapy

“Oh My Game” est une application dont le fonctionnement repose en très grande partie sur sa base de données et plus spécifiquement sur l'ensemble des jeux qui y sont référencés. C'est pourquoi il était important pour moi que nous puissions peupler la base avec des données de qualités. Nous avons développé des *fixtures* afin de tester nos fonctionnalités au plus vite, mais avec l'objectif de rapidement ne plus dépendre d'elles.

C'est pourquoi j'ai développé un outil de *scrapping* afin de récupérer de vraies données à partir d'un site existant. Après un rapide inventaire des sites de vente de jeux, j'ai contacté le directeur de la chaîne de magasin **PLAY IN** afin d'obtenir leur autorisation d'exploiter leurs données. Quelques échanges de mails plus tard, ils acceptaient de nous laisser *scraper* leur site.

J'entamais donc le développement du service que je baptisais avec beaucoup d'originalité “Scrapy”. Je l'ai conçu de la manière suivante : une première fonction qui allait *crawler* un ensemble de pages qui affichent tous les jeux du magasin sous la forme de vignettes.



Leur code source me permet de identifier une url dans chaque vignette qui renvoie vers la page de détail d'un jeu.

Extrait du code source :

```
<div class="container_info_product container_info_product_with_rating">
<a href="/produit/264994-skyjo">
<div class="name_product" title="Skyjo">Skyjo</div>
```

Je vais donc me constituer dans un premier temps une liste d'url de page de détail, nommée *games_tmp*, grâce à la fonction nommée *getListeGames()*. Pour se faire j'utilise le composant **Browser-kit** qui me permet d'instancier un objet *httpBrowser* qui va, selon l'url passée en paramètre dans sa méthode *request()*, me renvoyer le code source de la page. Je converti ce dernier en tableau afin de boucler dessus.

```

public function getListGames(int $nbPagesToScrap)
{
    //mon tableau qui sera retourné

    for($gu=1;$gu<$nbPagesToScrap;$gu++){

        //on va chercher les noms des jeux sur nbPagesToScrap pages
        //on accède aux pages de ce site en particulier de la façon suivante
        //https://www.play-in.com/jeux_de_societe/recherche/?p=1, 2 etc ...

        $urlToScrap=$this->url.$gu; // l'url de la page à scrap + le numéro de page
        $this->browser->request('GET',$urlToScrap);//la ressource browser

        //on crée un objet Response en string

        $contentPageString=($this->browser->getResponse())->getContent();

        //on écrit dans un txt le string retourné par la ligne précédente
        file_put_contents($this->tmpFile,$contentPageString);
        //on récupère le contenu du txt sous la forme d'un tableau
        $arrayContent=file($this->tmpFile);
        //id en attendant la db
        $a=0;

        while($a<count($arrayContent)){
            ...
        }
    }
}

```

Je me sers alors d'un *crawler* que j'ai développé qui prendra deux variables comme arguments : la ligne de texte à analyser et l'élément que l'on cherche (le nom ou l'url).

```

public function crawler(string $row,string $needle)
{
    //formats est un tableau qui stocke tous les patterns qui
    //serviront dans les preg_match
    if(preg_match($this->formats[$needle]['line'],$row)){

        preg_match($this->formats[$needle]['itemInLine'],$row,$matches);
        if(isset($matches[1])){

            return $matches[1];
        }
    }
    else{

        false;
    }
}

```

Grâce à la fonction *preg_match()* de **PHP** et une expression régulière, un résultat m'est retourné s'il y a eu correspondance. J'instancie alors un objet *GameTmp* dont je set les

attributs *name* et *href* avec les données précédemment récupérées. Je *persist* et *flush* via la méthode *add()* du *repository* de la classe *GameTmp* afin d'inscrire l'objet en base.

```
$gameTmp = new GameTmp;

while(is_null($gameTmp->getName())){

    if($this->crawler($arrayContent[$a],"href")){

        $href=$this->crawler($arrayContent[$a],"href");
        $gameTmp->setHref($href);

    }

    if($this->crawler($arrayContent[$a],'name'))
    {

        $name=$this->crawler($arrayContent[$a], "name");
        $gameTmp->setName($name);

    }

    $a++;
    if($a >= count($arrayContent)){
        break;
    }
}

$gameTmp->setId($a);

if(!is_null($gameTmp->getName()) && !is_null($gameTmp->getHref())){

    $this->gameTmpRepo->add($gameTmp,true);
    $this->flag++;

}

}

return True;
```

Le processus est lancé via une commande que j'ai créée.

```
php bin/console Scrapy init --nbPages=(ici le nombre de pages que je souhaite scraper)
```

Le site compte environs 150 pages qui affichent en moyenne une vingtaine de vignettes. J'ai pu obtenir ainsi environs 3000 urls renvoyant vers autant de pages de détails, je pouvais donc potentiellement peupler ma base avec 3000 "jeux temporaires".

Une fois cette liste de *gameTmp* disponible je vais développer un deuxième *crawler* chargé de récupérer toutes les informations d'une page de détail d'un jeu. Le principe est le même, j'instancie un objet *HttpBrowser* dont la méthode *request()* va me renvoyer le code source de la page dont l'url est extraite de ma liste *game_tmp*. Grâce à un ensemble

d'expressions régulières utilisées avec la fonction `preg_match()` je vais pouvoir extraire toutes les données nécessaires en bouclant sur le code source.

Extrait du crawler, ici pour récupérer la description longue d'un jeu :

```
if(preg_match($arrayPreg['longDescription'],$line)){  
    $longDescriptionArray=[];  
    while(!preg_match('/<th\s\s\s\s="row">Langue</th>$/',$arrayContent[$a])){  
  
        $str=mb_convert_encoding(strip_tags($arrayContent[$a]),'utf-8');  
        $str=str_replace($decodeString,$replaceString,$str);  
        $longDescriptionArray[]=$str;  
  
        $a++;  
    }  
    if(!empty(implode(PHP_EOL,$longDescriptionArray))){  
        $arrayResponse['longDescription']=implode(PHP_EOL,$longDescriptionArray);  
    }  
    else{  
        $this->writeLog($name,"longDescription");  
        break;  
    }  
}
```

Je vais ainsi récupérer les informations suivantes : l'url de l'image du jeu, une description courte, une description longue, les nombre de joueur max et min, la durée moyenne d'une partie, les auteurs, l'éditeur, les types associés, les thèmes associés, les illustrateurs. Si une donnée est manquante je set l'attribut *invalide* de ma classe *Scrappy* à TRUE et j'inscris dans un fichier texte de log le nom du jeu et le champ manquant. Si *invalide* est à FALSE j'instancie alors un objet *Game* et je set ses champs avec les données récupérées. Je *persist* alors le jeu en base via la méthode *add()* issue du *repository* associé à la classe *Game*.

```

if(!$this->invalidate){

    //dd('jeu invalide check the log');

    //sinon on persist et on flush le game
    $game->setImageUrl($arrayResponse['image']);
    $game->setSlug($this->slugify($name));
    $game->setDuration((int)$arrayResponse['duration']);
    $game->setMinimumAge((int)$arrayResponse['age']);
    $game->setShortDescription($arrayResponse['shortDescription']);
    $game->setLongDescription($arrayResponse['longDescription']);
    $game->setPlayersMax((int)$arrayResponse['nbJoueursMax']);
    $game->setPlayersMin((int)$arrayResponse['nbJoueursMin']);
    $this->gameRepository->add($game,true);

```

Je m'occupe ensuite des relations en associant au jeu ses auteurs, ses illustrateurs, ses thèmes, ses types, son éditeur. Je m'assure d'abord que l'entité que je vais associer n'existe pas en base ; si oui alors je la crée et la *persist*, puis je *set* le champ correspondant de l'objet jeu.

```

//on ajoute les types à la base si ils n'existent pas en base
//on ajoute à la relation game_types
foreach($arrayResponse['types'] as $name){

    if(empty($this->typeRepo->findByName($name))){

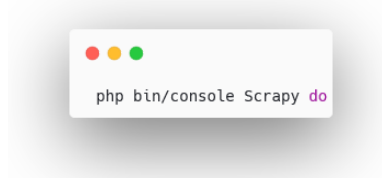
        $type=new Type;
        $type->setName($name);
        $type->setSlug($this->slugify($name));
        $this->typeRepo->add($type,true);
    }
    $type=$this->typeRepo->findByName($name);
    foreach($type as $item)
    {
        $game->addType($item,true);
    }
}

```

Pour la date de sortie du jeu j'ai utilisé le **Faker**, la date de sortie du jeu n'étant pas indiquée dans le code source de la page.

J'ai rencontré des problèmes liés à l'encodage du texte, j'ai donc dû utiliser la fonction *str_replace()* de PHP en association avec des tableaux de caractères afin d'inscrire un texte lisible en base.

J'ai créé une autre commande afin de lancer le processus.



Pour l’anecdote, quelques jours après ma demande auprès du directeur de la franchise, le site disposait d’un nouveau robot empêchant **Scrapy** de fonctionner, il n’était plus possible de récupérer le code source des pages. J’ai cherché les moyens de le contourner mais par manque de temps nous avons décidé de nous contenter du jeu de données acquis l’issu d’un test de Scrapy réalisé un peu avant. Nous avons alors environs 200 jeux sur les 3000 possibles.

2. La messagerie

Un objectif de “Oh My Game” est la mise en relation d’amateurs de jeux de société. Le fait de pouvoir communiquer facilement afin de régler les détails de l’emprunt d’un jeu ou tout simplement d’échanger avec ses amis était important pour nous. C’est pourquoi nous avons doté l’application d’une messagerie que j’ai développé.

Je l’ai conçu *from scratch*, ne m’appuyant sur aucun modèle en particulier. Mon idée était la suivante : je crée une entité *Chatting* (conversation) qui possède les attributs *userFrom*(expéditeur), *userTo*(destinataire) et *messages*

```

<?php

namespace App\Entity;

use App\Repository\ChattingRepository;
use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

#[ORM\Entity(repositoryClass: ChattingRepository::class)]
class Chatting
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\ManyToOne(inversedBy: 'chattingsFrom')]
    private ?User $userFrom = null;

    #[ORM\ManyToOne(inversedBy: 'chattingsTo')]
    private ?User $userTo = null;

    #[ORM\OneToMany(mappedBy: 'chatting', targetEntity: Message::class, orphanRemoval: true)]
    private Collection $messages;
}

```

Quand un utilisateur souhaite en contacter un autre on vérifie que la conversation n'existe pas déjà avec cet utilisateur qui initie la conversation soit en *userFrom* soit en *userTo*. Si elle n'existe pas on la crée.

```

#[Route('/profil/messagerie/{id}', name: 'app_profil_chatting_user')]
public function chattingValidate(User $user, ChattingRepository $chattingRepository): Response
{
    $userConnected = $this->getUser();
    $chattings = $user->getChattingsFrom();

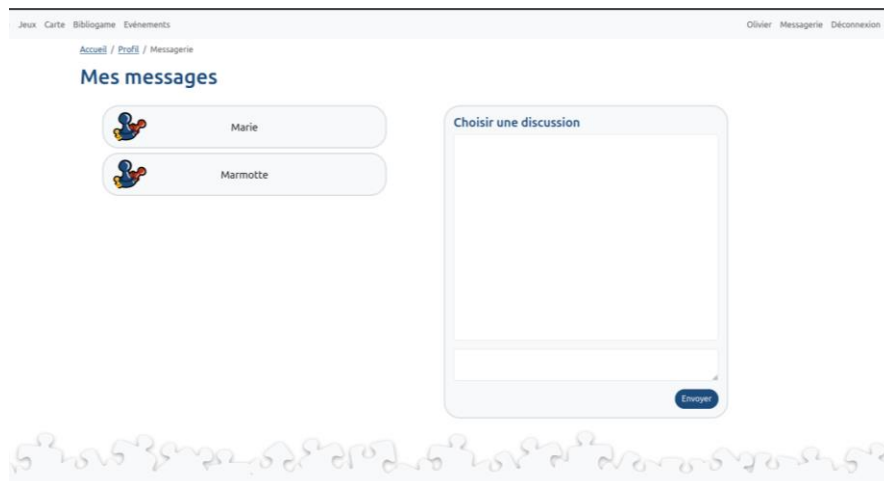
    //on veut pouvoir aussi afficher les chattings ou user est user TO
    $chattingsTo = $user->getChattingsTo();
    foreach ($chattingsTo as $chatting) {
        $chattings->add($chatting);
    }

    if (empty($chattingRepository->findBy(['userFrom' => $userConnected, 'userTo' => $user]))) {
        if (empty($chattingRepository->findBy(['userFrom' => $user, 'userTo' => $userConnected]))) {
            $newChatting = new Chatting();
            $newChatting->setUserFrom($user);
            $newChatting->setUserTo($userConnected);
            $chattingRepository->add($newChatting, true);
            $chattings->add($newChatting);
        }
    }

    return $this->redirectToRoute('app_profil_chatting', [
        'chattings' => $chattings
    ]);
}

```

Il ne reste ensuite qu'à afficher l'ensemble des messages qui lui sont associés. L'utilisateur peut ensuite envoyer un message qui sera enregistré et associé à la conversation, on réaffiche ensuite l'ensemble des messages, etc ... Pour moi une messagerie doit être instantané, je décide donc d'utiliser **Mercure**.



Si l'utilisateur possède déjà des conversations, on les affiche via une boucle **Twig**

```
{% for chatting in chattings %}
    {% if chatting.userFrom.id == app.user.id %}
        {% set userTo = chatting.userTo %}
    {% else %}
        {%set userTo = chatting.userFrom %}
    {% endif %}

    <div class="row bg-light border border-3 rounded-5 d-flex align-items-center p-1
mb-2">
        <div class="col-3">
            {% if userTo.avatar %}
                
            {% else %}
                
            {% endif %}
        </div>
        <div class="col-6 chatting text-center" value={{ chatting.id }} userTo="{{
userTo.alias }}" style="cursor:pointer">
            <span class="fs-5">{{userTo.alias}}</span>
            </div>
            {% if chatting.messages is empty %}
                <div class="col-3 text-center">
                    <a href="{{path('app_profil_chatting_delete',{id:chatting.id}) }}"
class="btn btn-danger rounded-circle" id='trash'>
                        <i class="fa-solid fa-trash-can" value={{ chatting.id }} ></i>
                    </a>
                </div>
            {% endif %}
        </div>
    </div>
{% endfor %}
```

Lorsqu'on clique sur une conversation pour la sélectionner on fait appel, via un *listener* lié la div correspondante à la conversation, à un ensemble de fonctions Javascript. Tout d'abord, on souscrit à un topic qui représente un canal, auquel on peut s'abonner, pour en recevoir les *Updates*.

```
GET http://localhost:3000/.well-known/mercure?topic=http://localhost/ping/3
GET http://127.0.0.1:8080/api/messages/3 [HTTP/1.1 200 OK 19ms]
```

```
var url=new URL('http://localhost:3000/.well-known/mercure');
url.searchParams.append('topic','http://localhost/ping/'+chattingId);
}
```

On crée ensuite la variable *eventSource* à partir de l'url du *topic*, créant ainsi l'interface qui va permettre de recevoir les événements envoyés par le serveur. On fait alors appel à sa propriété gestionnaire d'événement *onmessage()* qui fera appel à la méthode *showMessage()* lorsqu'un message sera envoyé par le serveur.

```
var chattingElements = document.querySelectorAll('.chatting');
var btnDeleteChatting = document.querySelectorAll('.btnDeleteChatting');
var submit=document.getElementById('submit');
var chattingId=null;
var chattingTo=document.getElementById('chattingTo');
var message=document.getElementById('message');

// foreach chatting in chatting list
chattingElements.forEach(function(chatting){

    chatting.addEventListener('click',function(event){
        chattingId=chatting.getAttribute("value")

        var url=new URL('http://localhost:3000/.well-known/mercure');
        url.searchParams.append('topic','http://localhost/ping/'+chattingId);

        const eventSource = new EventSource(url);

        eventSource.onmessage = e => showMessage(e.data,eventSource,chattingId);

        show(chatting.getAttribute("value"));
        submit.disabled = false;
        message.disabled = false;
        chattingTo.innerHTML = 'Ecrire à ' + chatting.getAttribute('userTo');

    })
})
```

On affiche ensuite tous les messages de la conversation via la méthode *show()*. Cette dernière va *fetch* tous les messages d'une conversation via une requête **AXIOS** en GET avec comme paramètre l'id de la conversation vers le contrôleur *listChatting*.

```
//refresh the chatting ; new message appears
function show(id){

  axios.get('/api/messagerie/'+id)
  .then(function(response){
    var chatFrame=document.getElementById('chatFrame');

    chatFrame.innerHTML="";
    var messages=response.data;
    chattingId=id;

    for (let i = 0; i < messages.length; i++) {
      el = document.createElement('p');
      el.innerHTML = messages[i].content;
      chatFrame.appendChild(el);
    }
  })
  .catch(function(error){
    console.log(error)
  })
}
```

Pour éviter tout problème de référence circulaire au moment de la sérialisation en JSON des objets *messages* dans la réponse j'ai créé un groupe.

```
#[Route('/api/messagerie/{id}', name: 'app_api_messagerie')]
public function listChatting(ChattingRepository $chattingRepository,$id): JsonResponse
{
    $chatting=$chattingRepository->find($id);
    $messages=$chatting->getMessages();
    return $this->json($messages,Response::HTTP_OK,[],['groups'=>"message"]);
}
```

L'espace d'affichage des messages est vidé avec sa propriété *innerHTML*, que je *set* vide, puis on *append* tous les messages récupérés par la requête via une boucle.

Il est possible ensuite d'envoyer un message en rentrant un texte dans le *textarea* ; le bouton "envoyer" est lié à deux écouteurs qui, selon que l'on clique ou que l'on presse la touche entrée, appellent la même fonction d'envoi du message. Une requête **AXIOS** est alors envoyé en POST cette fois-ci vers un contrôleur qui va se charger de récupérer le contenu du message et d'instancier un objet Message.

```
//sending message clicking on "envoyer"
document.getElementById('submit').addEventListener('click',function(){
  var userAlias = document.getElementById('userAlias');

  if(document.getElementById("message").value != ''){

    axios.post('/api/test',{
      message:document.getElementById("message").value,
      chatting:chattingId,
      alias:document.getElementById('userAlias').getAttribute('value'),
      userId:document.getElementById('userToId').getAttribute('value')
    }).then(function(response){
      document.getElementById('message').value="";
      show(chattingId);
    })
  }
})
```

```

#[Route('api/test',name:"app_test",methods:['POST'])]
public function ping(PublisherInterface $publisher,Request $request,ChattingRepository
$chattingRepository,EntityManagerInterface $entityManager,HubInterface $hub):JsonResponse
{
    //on récupère le message envoyé
    $messageContent = $request->getContent();
    $messageContent=json_decode($messageContent,true);
    $chatting=$chattingRepository->find($messageContent['chatting']);
    //

    $message = new Message;
    $message->setSentAt(new DateTimeImmutable());
    $message->setContent($messageContent['alias']." dit : ".$messageContent['message']);
    $chatting->addMessage($message);
    $entityManager->persist($message);
    $entityManager->flush();

    $messages=$chatting->getMessages();
    $array=[];

    foreach($messages as $message){
        $array[]=$message->getContent();
    }

    $update= new Update('http://localhost/ping/'.$messageContent['chatting'],json_encode($array));

    $hub->publish($update);

    return $this->json('http://localhost/ping/'.$messageContent['chatting']);
}
}

```

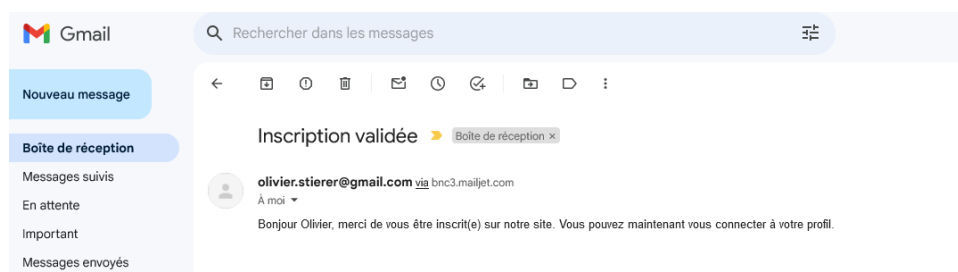
Les attributs *date* et *content* de ce dernier sont *set*. On ajoute ce message à la conversation puis on instancie un objet *Update* issue du composant **Mercure-Bundle** qui va permettre de publier tous les messages sur le *topic* lié à ma conversation. Ces derniers sont regroupés dans un tableau encodé en JSON qui est l'un des arguments du constructeur de l'objet *Update*. **Mercure** envoie alors ce tableau via le *hub* à l'ensemble des utilisateurs qui ont souscrit au *topic* correspondant. Grâce à la propriété *onmessage* de l'objet *eventsourcing* le client reçoit la notification poussée par le serveur dans laquelle se trouve le tableau des messages de la conversation.

La fonction *showMessage()* est alors appelée, le tableau des messages est alors passé en argument. On affiche alors tous les messages de la même façon que la méthode *show()* en vidant le contenu de la div affichant la conversation, puis en bouclant sur le tableau on append chaque message.

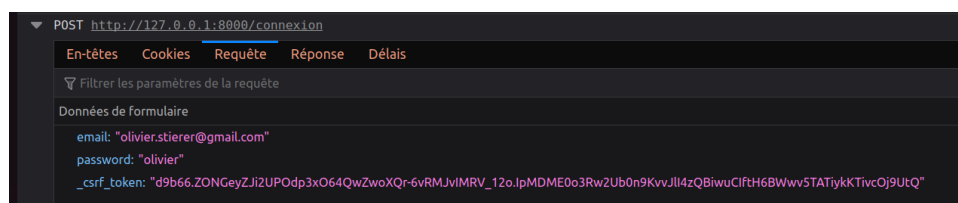
vii. Jeu d'essai d'une fonctionnalité

Afin de tester le bon fonctionnement de notre application, nous avons fait des essais d'inscription d'un nouvel utilisateur, et de connexion avec le compte nouvellement créé et d'affichage de l'utilisateur sur la carte.

En données entrées, nous avons renseigné toutes les informations nécessaires à la création d'un compte dans le formulaire d'inscription. Si l'inscription est un succès, un mail est envoyé à l'adresse renseignée.



Nous avons ensuite tenté l'authentification via le formulaire de connexion. L'appel au contrôleur qui permet l'authentification est bien appelé et les éléments nécessaires à l'authentification sont bien transmis.



Une fois connecté, nous avons bien accès à toutes fonctionnalités attendues pour un utilisateur.

Lors de l'inscription, un *listener*, qui appelle un service permettant de récupérer les coordonnées GPS à partir de l'adresse entrée dans le formulaire, doit *set* les attributs longitude et latitude de l'objet User nouvellement créé.


```

class UserListener
{
    public function __construct(
        private CoordinatesService $coordinatesService
    ) {
    }

    /**
     * Enregistre les coordonnées GPS de User, à PrePersist
     *
     * @param User $user
     * @param PrePersistEventArgs $event
     * @return void
     */
    public function prePersist(User $user, PrePersistEventArgs $event): void
    {
        $address = $user->getAddress() . ' ' . $user->getCity();
        $coordinates = $this->coordinatesService->getCoordinates($address);

        if ($coordinates) {
            $user->setLatitude($coordinates['latitude'])
                ->setLongitude($coordinates['longitude']);
        }
    }
}

```

Une fois sur la page de carte on constate l'appel de la requête Axios qui va *fetch* et *set* les utilisateurs sur la carte.

```

▶ XHR GET http://127.0.0.1:8000/api/users/

```

```

#[Route('/api/users', name: 'app_api_users', methods: ["GET"])]
public function users(UserRepository $userRepository): JsonResponse
{
    $users = $userRepository->findAll();
    return $this->json($users, Response::HTTP_OK, [], ["groups" => "users"]);
}

```

```

function mapAllUsers(map, user) {
  axios.get('/api/users/')
    .then(function (response) {
      // Vider le layerGroup avant d'ajouter de nouveaux utilisateurs
      usersLayer.clearLayers();

      response.data.forEach(element => {
        // marqueur pour les utilisateurs, sauf l'utilisateur actuel
        createMarkers(map, user, element, usersLayer);
      })
    })
    .catch(function (error) {
      // en cas d'échec de la requête
      console.log(error);
    })
}

```

On constate qu'un marqueur a bien été placé sur la carte à l'adresse renseignée lors de l'inscription.



Durant ce test, les comportements et données attendues ont été conformes aux attendus : inscription validée par mail, authentification réussie, le *listener* a bien été appelé, l'affichage de la localisation sur la carte réussi.

VIII. Veille effectuée sur les vulnérabilités de sécurité

À mesure que l'on progresse dans la conception d'application web plus on prend conscience de la multitude d'attaques possibles. **Symfony** et **Twig** gèrent déjà un ensemble de protections contre les actes malveillants et le temps qui nous était imparti pour la réalisation de notre application nous a souvent poussé à nous en remettre entièrement aux solutions proposées par notre framework sans plus de questions de notre part. C'est pourquoi j'ai fait la démarche d'approfondir mes connaissances dans les attaques possibles, les moyens existants et les bonnes pratiques qui permettent de s'en prémunir efficacement.

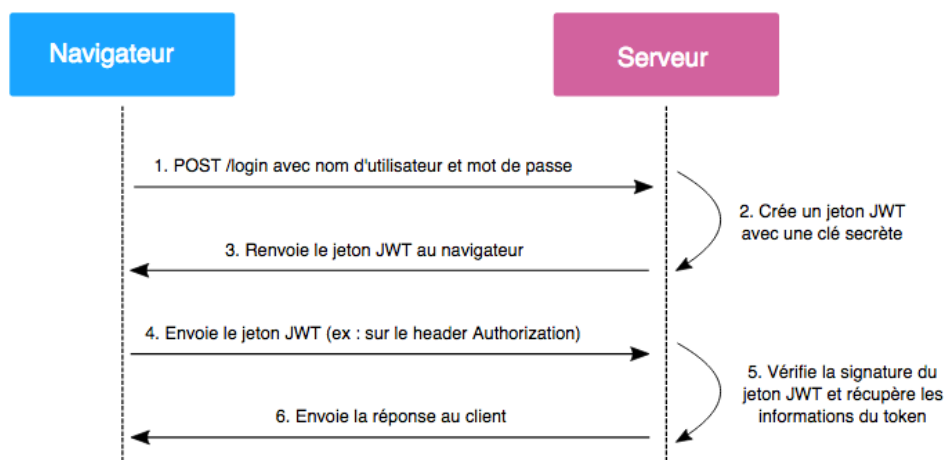
Tout d'abord j'ai entamé une recherche sur les vulnérabilités et leur exploitation. Certaines des attaques les plus courantes ont été déjà évoquées précédemment : les attaques par injections SQL ou de scripts malveillants, les usurpations d'identités, l'exploitation de la session, ou le détournement d'une url. D'autres m'étaient inconnues comme les attaques MITM (*Man In The Middle*) qui vont exploiter une brèche entre deux parties afin d'espionner les données envoyées. De la même façon j'ai appris l'existence de l'*IP Spoofing* qui va permettre à une personne mal intentionnée d'envoyer des paquets IP depuis une adresse IP source qui n'est pas la sienne, masquant ainsi son identité lors d'attaques par déni de service par exemple. Toutes ces actions ont souvent pour but le vol et la manipulation de données, la réalisation d'actions non voulues par l'utilisateur.

Parmi les articles lus (Vaadata, Oracle, Red Hat), un a retenu particulièrement mon attention. Il s'agit d'un [article](#) qui traite de la sécurisation des API et de la meilleure façon de les sécuriser avec l'utilisation de jetons, de chiffrement, de signatures, de quotas, de limitation des requêtes.

Or "Oh My Game" possède une partie API. Certains contrôleurs sont appelés via le client http **AXIOS** afin de renvoyer des données en JSON ou d'effectuer certaines actions et aucun d'entre eux ne semblent répondre aux critères de sécurisation cités précédemment. Je décide alors de mettre en place une authentification par **JWT** lors des requêtes vers l'API.

Pour cela je vais utiliser le **jwt-authentication-bundle** de **Lexik** qui va configurer l'ensemble des fichiers nécessaires afin de mettre en place une authentification par **JWT**.

Le **JWT** ou *JSON Web Token* sont des jetons générés par le serveur lors de l'identification d'un utilisateur qui va ensuite être transmis au client. Ils seront renvoyés avec chaque requête HTTP vers le serveur qui n'aura qu'à comparer la signature envoyée par le client avec la signature du jeton qu'il aura généré précédemment. Si les deux sont identiques alors le jeton est valide.

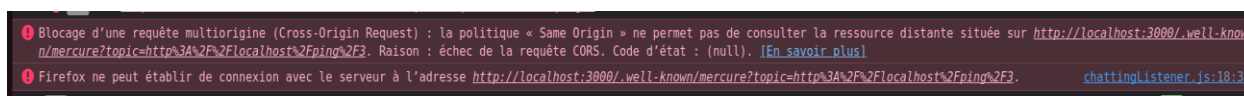


Sa structure est en trois parties et suit la norme RFC 7519 : un *header* qui contient l'algorithme pour la signature ainsi que le type du jeton encodé en base64 , ici JWT. Un *payload* qui contient pour "OhMyGame" des informations sur l'utilisateur encodées aussi en base 64 et enfin la signature qui est la concaténation des deux premières parties ainsi que d'une clé privée qui se trouve dans le dossier config/jwt du projet.

ix. Situation de travail ayant nécessité une recherche

1. Le problème rencontré

La création d'une messagerie de type chat nécessitait d'installer un système qui permette l'envoi de notifications depuis le serveur. C'est pourquoi j'ai choisi d'installer **Mercure**. Cependant son installation et sa configuration ont posé un certain nombre de problèmes. Parmi eux un problème de CORS. S'il était possible d'envoyer un message, celui-ci n'apparaissait pas instantanément chez le destinataire. J'obtenais en console le message d'erreur suivant



```
❌ Blocage d'une requête multiorigine (Cross-Origin Request) : la politique « Same Origin » ne permet pas de consulter la ressource distante située sur http://localhost:3000/.well-known/mercure?topic=http%3A%2F%2Flocalhost%2Fping%2F3. Raison : échec de la requête CORS. Code d'état : (null). [En savoir plus]
❌ Firefox ne peut établir de connexion avec le serveur à l'adresse http://localhost:3000/.well-known/mercure?topic=http%3A%2F%2Flocalhost%2Fping%2F3. chattingListener.js:18:32
```

2. La recherche effectuée

Afin d'effectuer ma recherche, j'ai tapé les termes 'allow cors-origin mercure' dans mon navigateur Firefox. Le deuxième résultat envoie vers une page de la documentation de **Mercure** sur laquelle figure un article avec comme entête "CORS Issue". J'y retrouve l'erreur vue dans la console de mon navigateur, ainsi qu'une autre version de ce message d'erreur mais pour Chrome. La solution proposée est de modifier le fichier *CaddyFile*, en *settant* la valeur de l'option *cors_origin*, soit en précisant une adresse http ou en utilisant le caractère "*" qui autorisera toutes les applications à se connecter au hub.

3. La source anglophone utilisée

CORS Issues	Problèmes de Cors
If the app connecting to the Mercure hub and the hub itself are not served from the same domain, you must whitelist the domain of the app using the CORS (Cross-Origin Resource Sharing) mechanism.	Si l'application qui se connecte au hub de Mercure et que le hub lui-même ne sont pas issus du même domaine, vous devez autoriser le domaine de l'application en utilisant le

<p>The usual symptoms of a CORS misconfiguration are errors about missing CORS HTTP headers in the console of the browser:</p>	<p>mécanisme de CORS(Cross-origin Resource Sharing)</p> <p>Le plus souvent le signe d'une mauvaise configuration des CORS sont les erreurs qui apparaissent dans la console du navigateur concernant l'absence de CORS dans les entêtes des requêtes http</p>
<p>To fix these errors, set the list of domains allowed to connect to the hub as value of the cors_origins in the Caddyfile.</p> <p>If you don't use an authorization mechanism (anonymous mode), you can set the value of cors_origins to * to allow all applications to connect to the hub (be sure to understand the security implications of what you are doing).</p>	<p>Pour régler le problème, configurer les cors_origin en renseignant la liste des domaines autorisés à se connecter au hub dans le fichier Caddyfile.</p> <p>Si vous n'utilisez pas le mécanisme d'autorisation (mode anonyme) vous pouvez configurer les cors_origin à * pour autoriser toutes les applications à se connecter au hub (soyez sûrs de bien prendre en compte ce que cela implique en terme de sécurité)</p>

Conclusion

“Oh My Game” et sa réalisation m’auront beaucoup apporté et je suis heureux d’avoir proposé cette idée qui a su motiver les membres de l’équipe et susciter l’intérêt de membres extérieurs tels que nos référents pédagogiques ou apprenants de la promotion. Je suis aussi fier du travail accomplis par mes partenaires et moi-même. Nous sommes allés bien plus loin que prévu et le projet compte nombre de belles réalisations.

Sur le plan technique Oh My Game m’a fait prendre conscience que le développement d’une application ne se résumait pas uniquement à coder mais allait bien au-delà. J’ai beaucoup appris des différentes phases hors réalisation, notamment de la phase de

conception durant laquelle j'ai pu participer à la création d'un cahier des charges et réaliser des maquettes. Avec Symfony je suis allé plus loin dans l'exploitation de ma connaissance de PHP et j'ai gagné en efficience. Le projet m'a fait gagner en confiance dans l'utilisation de Git. Je me suis aussi enrichi lors de séance de debug en m'immergeant dans un code qui n'était pas le mien.

Sur le plan humain le projet s'est montré particulièrement enrichissant. Le hasard a voulu que le niveau technique des membres de l'équipe ne soit pas homogène, trois membres sur quatre (dont moi) possédaient une expérience parfois significative dans le développement tandis que le quatrième débutait tout juste. Trouver des solutions pour pouvoir avancer tous ensemble n'a pas toujours été facile mais nous permis d'apprendre, parfois dans la douleur, à travailler en équipe.

Cette immersion dans des conditions de développement plus proches de la réalité que ce que j'avais pu connaître jusqu'alors m'a conforté dans mon choix de reconversion et a mis en lumière un certain nombre de mes points faibles, dont le *front-end*. Je vais donc profiter des ressources à mise à disposition par l'école pour m'initier à un framework *front* comme **React** et tenter de combler mes lacunes dans ce domaine. La suite sera aussi mon intégration en deuxième année au sein d'un B.U.T (Bachelor Universitaire Technologique) informatique en développement d'application à l'I.U.T de Nantes.

Pour "Oh My Game" l'histoire continue aussi : une des conditions de mon entrée à l'IUT en septembre est l'apprentissage du KOTLIN, je vais donc développer une véritable application mobile d'ici septembre à partir de ce projet.

Annexes

1. Extrait du script de migration :

```
$this->addSql('ALTER TABLE bibliogame ADD CONSTRAINT FK_2E765A0DE48FD905 FOREIGN KEY (game_id)
REFERENCES game (id)');
$this->addSql('ALTER TABLE bibliogame ADD CONSTRAINT FK_2E765A0D7597D3FE FOREIGN KEY (member_id)
REFERENCES user (id)');
$this->addSql('ALTER TABLE bibliogame ADD CONSTRAINT FK_2E765A0D39759382 FOREIGN KEY
(borrowed_by_id) REFERENCES user (id)');
$this->addSql('ALTER TABLE bibliogame ADD CONSTRAINT FK_2E765A0DF7F09C21 FOREIGN KEY
(request_by_id) REFERENCES user (id)');
$this->addSql('ALTER TABLE chatting ADD CONSTRAINT FK_C71695F520C3C701 FOREIGN KEY (user_from_id)
REFERENCES user (id)');
$this->addSql('ALTER TABLE chatting ADD CONSTRAINT FK_C71695F5D2F7B13D FOREIGN KEY (user_to_id)
REFERENCES user (id)');
$this->addSql('ALTER TABLE event ADD CONSTRAINT FK_3BAE0AA71F88D185 FOREIGN KEY (host_id)
REFERENCES user (id)');
$this->addSql('ALTER TABLE event_game ADD CONSTRAINT FK_3CE07D2771F7E88B FOREIGN KEY (event_id)
REFERENCES event (id) ON DELETE CASCADE');
$this->addSql('ALTER TABLE event_game ADD CONSTRAINT FK_3CE07D27E48FD905 FOREIGN KEY (game_id)
REFERENCES game (id) ON DELETE CASCADE');
$this->addSql('ALTER TABLE event_requests ADD CONSTRAINT FK_3D693F81A76ED395 FOREIGN KEY
(user_id) REFERENCES user (id)');
$this->addSql('ALTER TABLE event_requests ADD CONSTRAINT FK_3D693F8171F7E88B FOREIGN KEY
(event_id) REFERENCES event (id)');
$this->addSql('ALTER TABLE game ADD CONSTRAINT FK_232B318C6995AC4C FOREIGN KEY (editor_id)
```

```
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

/**
 * Auto-generated Migration: Please modify to your needs!
 */

final class Version20231205183724 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE author (id INT AUTO_INCREMENT NOT NULL, name VARCHAR(50) NOT NULL,
PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
        $this->addSql('CREATE TABLE bibliogame (id INT AUTO_INCREMENT NOT NULL, game_id INT NOT NULL,
member_id INT NOT NULL, borrowed_by_id INT DEFAULT NULL, request_by_id INT DEFAULT NULL, is_available
TINYINT(1) DEFAULT NULL, borrowed_at DATETIME DEFAULT NULL COMMENT \'(DC2Type:datetime_immutable)\',
INDEX IDX_2E765A0DE48FD905 (game_id), INDEX IDX_2E765A0D7597D3FE (member_id), INDEX IDX_2E765A0D39759382
(borrowed_by_id), INDEX IDX_2E765A0DF7F09C21 (request_by_id), PRIMARY KEY(id)) DEFAULT CHARACTER SET
utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
        $this->addSql('CREATE TABLE chatting (id INT AUTO_INCREMENT NOT NULL, user_from_id INT DEFAULT
NULL, user_to_id INT DEFAULT NULL, INDEX IDX_C71695F520C3C701 (user_from_id), INDEX IDX_C71695F5D2F7B13D
(user_to_id), PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE =
InnoDB');
```


2. La liste des routes

Url	Infos	name
public		
/	Page d'accueil	app_home
/jeux	base de données des jeux	app_games_list
/jeux/{slug-du-jeu}	Page de détail d'un jeu	app_games_details
/jeux/{slug-du-jeu}/avis	Page des avis d'un jeu/Ajouter un commentaire si loggé	app_games_reviews
/connexion	Page de connexion	app_login
/inscription	Page d'inscription	app_registration
/contact	Formulaire de contact	app_contact
admin		
/admin	Dashboard d'administration	app_admin
/admin/jeux	Liste de tous les jeux, pour l'admin	app_admin_games_list
/admin/{slug-du-jeu}	Détail d'un jeu admin, liste des reviews	app_admin_games_details
/admin/{slug-du-jeu}/editer	Editer un jeu	app_admin_games_edit
/admin/jeux/nouveau	Ajouter un jeu à la liste des jeux	app_admin_games_add
/admin/utilisateurs	Liste des utilisateurs	app_admin_users_list
/admin/utilisateurs/{id}/role	Changer le role d'un utilisateur	app_admin_users_role
user		
/profil	Page de profil	app_user
/profil/bibliogame	Bibliothèque de jeux personnelle	app_user_games
/profil/modifier	Modifier mon profil	app_user_edit
/profil/messagerie	liste des messages entre utilisateurs	app_user_messages
/deconnexion	page de déconnexion	app_logout

3. Extrait du dictionnaire de données

	entité	code	type	taille	description	contraintes
<u>Editor</u>	<u>Editeur</u>	<u>code_editeur</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>name</u>	nom	<u>varchar</u>	50	nom de l'auteur	NOT NULL, UNIQUE
	<u>slug</u>	slug	<u>varchar</u>	50	slug du nom de l'auteur	NOT NULL
<u>Theme</u>	<u>Theme</u>	<u>code_theme</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>name</u>	nom	<u>varchar</u>	50	animaux, zombies, mythologie, etc ...	NOT NULL, UNIQUE
	<u>slug</u>	slug	<u>varchar</u>	50	slug du type	NOT NULL
<u>Type</u>	<u>Type</u>	<u>code_type</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>name</u>	nom	<u>varchar</u>	30	jeu coopératif, ambiance, logique, quizz, etc ...	NOT NULL, UNIQUE
	<u>slug</u>	slug	<u>varchar</u>	30	slug du type	NOT NULL
<u>Illustrator</u>	<u>Illustrateur</u>	<u>code_illustrateur</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>name</u>	nom	<u>varchar</u>	50	nom du dessinateur	NOT NULL, UNIQUE
<u>Author</u>	<u>Auteur</u>	<u>code_auteur</u>	<u>int</u>		clé primaire	PRIMARY KEY
	<u>name</u>	nom	<u>varchar</u>	50	nom de l'auteur	NOT NULL, UNIQUE
<u>Game</u>	<u>Jeu</u>	<u>code_jeu</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>#editor</u>	<u>code_editeur</u>	<u>int</u>		clé étrangère	FOREIGN KEY
	<u>name</u>	nom	<u>varchar</u>	50	titre du jeu	NOT NULL, UNIQUE
	<u>minimumAge</u>	age	<u>int</u>		à partir de	
	<u>releasedAt</u>	<u>date_de_sortie</u>	<u>date</u>		date de sortie du jeu	
	<u>rating</u>	note	<u>float</u>		moyenne des notes utilisateurs	
	<u>duration</u>	durée	<u>int</u>		durée moyenne d'une partie en minutes	
	<u>imageUrl</u>	<u>url_image</u>	<u>varchar</u>	255	lien vers l'image du jeu	
	<u>playersMin</u>	joueur_min	<u>int</u>		nombre minimum de joueurs	DEFAULT 1
	<u>playersMax</u>	joueur_max	<u>int</u>		nombre maximum de joueurs	
	<u>slug</u>	slug	<u>varchar</u>	50	slug du jeu	NOT NULL
	<u>shortDescription</u>	description_courte	<u>text</u>		courte description du jeu en une phrase ou deux	NOT NULL
	<u>longDescription</u>	description_longue	<u>text</u>		description détaillé du jeu, avec exemple déroulement partie, etc ...	
<u>Bibliogame</u>	<u>JeuUtilisateur</u>	<u>code_jeuUtilisateur</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>#member</u>	<u>code_utilisateur</u>	<u>int</u>		clé étrangère	FOREIGN KEY
	<u>#game</u>	<u>code_jeu</u>	<u>int</u>		clé étrangère	FOREIGN KEY
	<u>#borrowedBy</u>	<u>code_utilisateur</u>	<u>int</u>		clé étrangère	FOREIGN KEY
	<u>#requestBy</u>	<u>code_utilisateur</u>	<u>int</u>		clé étrangère	FOREIGN KEY
	<u>isAvailable</u>	est_disponible	<u>bool</u>		est disponible à l'emprunt	
	<u>borrowedAt</u>	emprunté_le	<u>date</u>		date de l'emprunt	
<u>User</u>	<u>Utilisateur</u>	<u>code_utilisateur</u>	<u>int</u>		clé primaire	PRIMARY KEY, NOT NULL, AI
	<u>firstname</u>	nom	<u>varchar</u>	25	nom de l'utilisateur	NOT NULL
	<u>lastname</u>	prénom	<u>varchar</u>	25	prénom de l'utilisateur	NOT NULL
	<u>email</u>	email	<u>varchar</u>	40	adresse email de l'utilisateur	NOT NULL, UNIQUE
	<u>alias</u>	pseudo	<u>varchar</u>	50	pseudo de l'utilisateur	NOT NULL, UNIQUE