RAPPORT API REST

PAR

Matthieu FANGET

Loïc ALLEMAND

Aymeric SURRE

Github : https://github.com/Matthieu73110/simo-projet-api-rest

IUT2 LP-SIMO

TABLE OF CONTENTS

INTRODUCTION

1.1 Présentation du projet

Notre API REST dédiée aux jeux vidéo offre un accès pratique à un vaste catalogue de jeux, fournissant des informations détaillées telles que les éditeurs, les développeurs, les noms et les plateformes associées à chaque jeu. Grâce à cette API, les utilisateurs peuvent explorer et interagir avec les données de manière efficace.

1.2 Participation de chacun

Personne	Tâche
Matthieu FANGET	Réflexion de l'API Création du diagramme UML Écriture de la spec Swagger Configuration du serveur
Loïc ALLEMAND	Réflexion de l'API Création du diagramme UML Écriture de la spec Swagger Gestion des données
Aymeric SURRE	Réflexion de l'API Création du diagramme UML Écriture de la spec Swagger Écriture du Rapport Création du front

1.3 Étapes de réalisation de l'API

Notre équipe a suivi un processus structuré pour concevoir et développer notre API. Les étapes clés que nous avons suivies sont les suivantes :

1. **Création de notre API** : Nous avons commencé par réfléchir à la nature de l'API que nous souhaitions créer. Cette étape nous a permis de définir les fonctionnalités et les objectifs de notre API.

2. **Conception du diagramme UML** : Pour planifier l'architecture de notre API, nous avons créé un diagramme UML pour représenter les classes, les relations et les interactions de notre système. Ce diagramme nous a donné une vue d'ensemble claire de notre API et nous a aidés à prendre des décisions d'architecture importantes.

3. **Écriture de la spécification Swagger** : Une fois notre diagramme UML finalisé, nous avons utilisé Swagger pour décrire notre API de manière standardisée. Nous avons rédigé la spécification Swagger en utilisant la syntaxe YAML, décrivant les points d'entrée, les méthodes HTTP, les paramètres, les réponses et les modèles de données de notre API. Cela nous a permis de documenter précisément les détails de l'API et de faciliter la génération automatique du code du serveur.

4. **Génération du code du serveur** : À l'aide d'outils de génération de code compatibles avec Swagger, nous avons généré le code source du serveur. Le code généré comprenait les routes, les contrôleurs et les modèles nécessaires pour implémenter les fonctionnalités décrites dans la spécification.

5. **Création des données et des classes** : Nous avons créé les différentes structures de données et les classes nécessaires pour implémenter les fonctionnalités de notre API.

6. **Tests et validation de l'API** : Une fois le développement de l'API terminé, nous avons effectué des tests approfondis pour vérifier sa fonctionnalité et sa conformité aux spécifications. Nous avons exécuté des scénarios de test, vérifié les réponses, détecté d'éventuelles erreurs et optimisé le code source du serveur pour correspondre au mieux aux spécifications.

7. **Écriture du rapport** : Enfin, nous avons rédigé ce rapport qui documente notre approche ainsi que les différents documents que nous avons générés et utilisés.

DIAGRAMME

2.1 Diagramme de classes

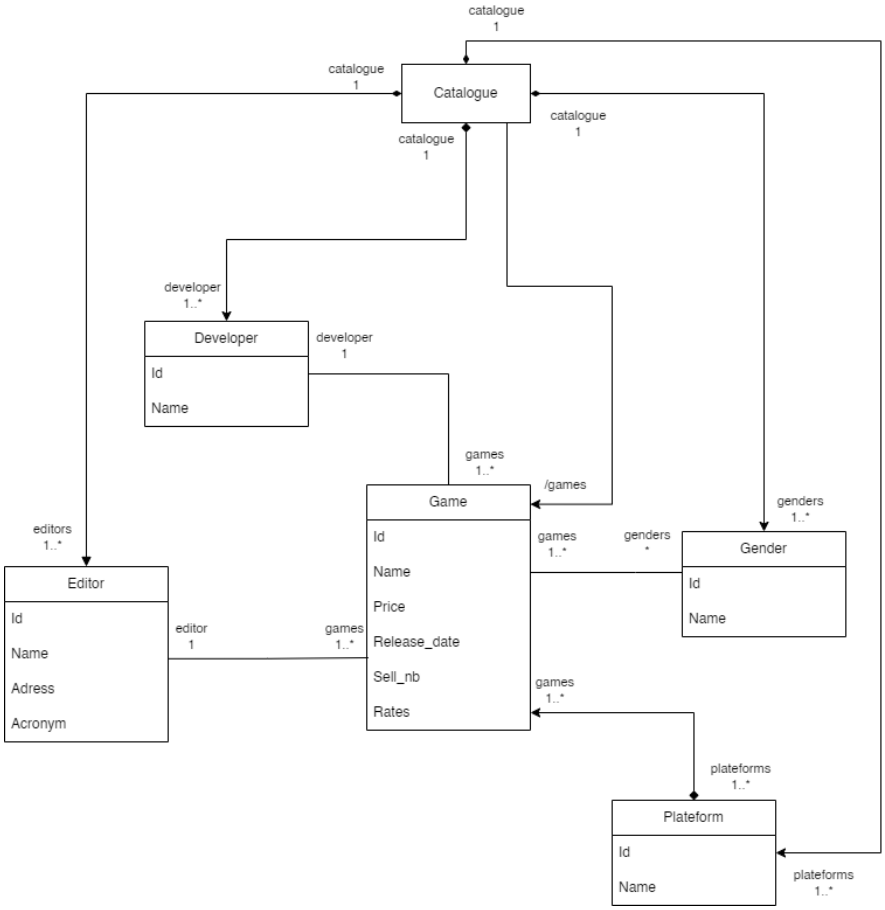

Figure 2.1: Diagramme de classe

CODE

3.1 Architecture de l'API

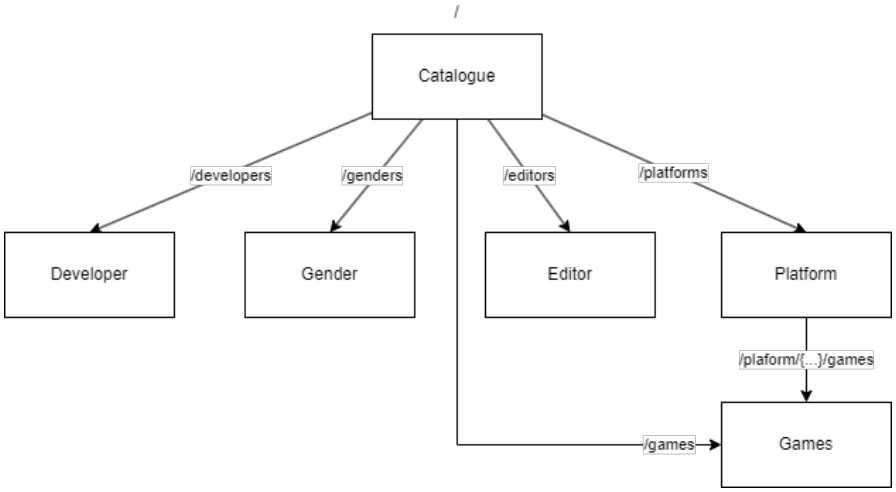

Figure 3.1: Arborescence du code

3.2 Architecture des fichiers du code

Figure 3.2: Architecture des fichiers du code

3.3 Explication de l'architecture des fichiers du code

L'implémentation de notre API suit une structure de fichiers organisée, ce qui facilite la gestion et la compréhension du code. Les principaux dossiers et fichiers de notre projet sont les suivants :

- **Dossier `controllers`** : Ce dossier contient tous les contrôleurs de notre API. Les fichiers présents dans ce dossier sont :
  - `Developer.js` : Ce contrôleur gère les opérations liées aux développeurs.
  - `Editor.js` : Ce contrôleur gère les opérations liées aux éditeurs de jeux.
  - `Game.js` : Ce contrôleur gère les opérations liées aux jeux.
  - `Gender.js` : Ce contrôleur gère les opérations liées aux genres de jeux.
  - `Platform.js` : Ce contrôleur gère les opérations liées aux plateformes de jeux.
- **Dossier `data`** : Ce dossier contient les données de notre API au format JSON. Les fichiers présents dans ce dossier sont similaires à ceux du dossier `controllers`, mais au format JSON. Par exemple :
  - `Developer.json` : Ce fichier contient les données des développeurs.
  - `Editor.json` : Ce fichier contient les données des éditeurs de jeux.
  - `Game.json` : Ce fichier contient les données des jeux.
  - `Gender.json` : Ce fichier contient les données des genres de jeux.
  - `Platform.json` : Ce fichier contient les données des plateformes de jeux.
- **Dossier `services`** : Ce dossier contient les services utilisés par les contrôleurs. Les fichiers présents dans ce dossier sont similaires à ceux du dossier `controllers`, mais représentent des services. Par exemple :
  - `DeveloperService.js` : Ce fichier contient la logique métier et les opérations liées aux développeurs.

  - `EditorService.js` : Ce fichier contient la logique métier et les opérations liées aux éditeurs de jeux.
  - `GameService.js` : Ce fichier contient la logique métier et les opérations liées aux jeux.
  - `GenderService.js` : Ce fichier contient la logique métier et les opérations liées aux genres de jeux.
  - `PlatformService.js` : Ce fichier contient la logique métier et les opérations liées aux plateformes de jeux.

- **Fichiers en vrac** : En plus des dossiers spécifiques, nous avons également certains fichiers en vrac présents à la racine du projet, tels que `init.js`. Ces fichiers peuvent contenir des scripts d'initialisation ou d'autres fonctionnalités spécifiques à notre application.

Cette structure de fichiers claire et organisée facilite la navigation et la maintenance du code, permettant à l'équipe de développement de travailler efficacement sur différents aspects de l'API.

3.4 Interface de l'API

Le fichier index.html à la racine du projet est l'interface web de notre API. Elle offre aux utilisateurs un moyen plus convivial et interactif d'accéder et d'interagir avec notre API. Ce fichier constitue le point d'entrée pour visualiser, rechercher, filtrer et effectuer des opérations sur les données des jeux vidéo via une interface utilisateur intuitive.

CONCLUSION

En conclusion, notre projet de création d'une API REST dédiée aux jeux vidéo s'est déroulé avec succès dans l'ensemble. Nous avons tout d'abord construit un diagramme UML, puis écrit la spécification Swagger, et enfin configuré le serveur, et géré les données.

L'architecture de notre API, représentée par le diagramme UML, a été pensée pour assurer une organisation claire du code. De plus, notre structure de fichiers bien organisée facilite la gestion et la maintenance du projet.

Cependant, nous avons rencontré quelques difficultés lors de la création du fichier init.js, ce qui a nécessité de devoir écrire les données en dur. Cela a pris beaucoup de temps et a retardé notre progression. De plus, nous avons également eu des problèmes lors du développement de la partie front-end de l'application. Ces problèmes ont été nombreux et nous n'avons pas pu correctement terminer cette partie du projet.

Malgré ces obstacles, nous sommes satisfaits du travail accompli. Nous avons développé une API robuste et fonctionnelle, offrant un accès pratique à un vaste catalogue de jeux vidéo. Bien que le front-end n'ait pas été entièrement réalisé, nous avons pu nous concentrer sur la création de l'API elle-même, qui est l'élément principal du projet.