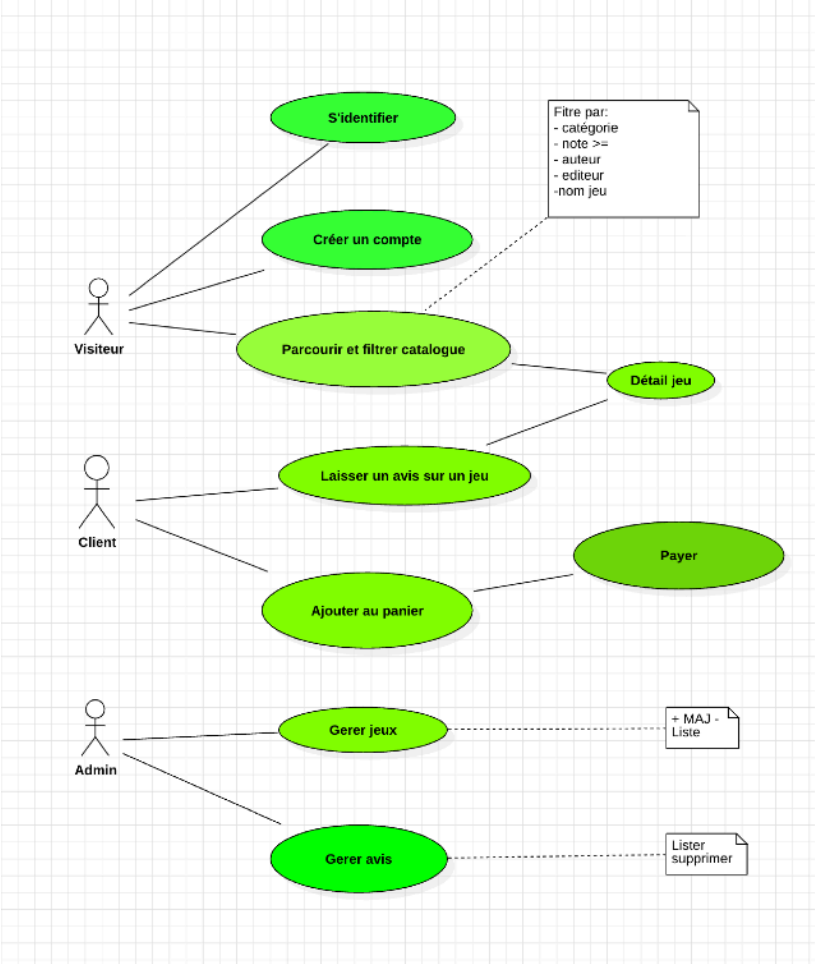
GamesUp Documentation

Repository GitHub : https://github.com/Vega0104/GamesUpApp.git

1. Les fonctionnalités

La phase initiale du projet a consisté en une analyse approfondie du code existant afin de comprendre le contexte métier et d'identifier les fonctionnalités attendues. Cette analyse a permis d'établir un diagramme de cas d'utilisation recensant l'ensemble des interactions utilisateur-système et servant de référentiel pour prioriser les développements urgents dans le cadre de la refonte.



Le diagramme de cas d'utilisation permet de visualiser l'ensemble des fonctionnalités du système du point de vue des différents acteurs (utilisateurs, administrateurs). Il a servi de base pour identifier les priorités de développement et garantir que les besoins fonctionnels essentiels soient couverts lors de la refonte.

2. Refonte de l'API Spring

2.1 Objectifs de la refonte

La refonte de l'API Spring a poursuivi plusieurs objectifs d'amélioration de la qualité du code :

- Uniformisation linguistique : Migration complète du code en anglais pour respecter les standards internationaux de développement.

- Amélioration du nommage : Adoption de conventions de nommage claires et explicites pour les classes, méthodes et variables.

- Restructuration architecturale : Mise en place d'une architecture en couches respectant le pattern Entity - Repository - Service - Controller

2.2 Architecture en couches

L'architecture adoptée suit le principe de séparation des responsabilités :

- Entity : Représentation des entités métier et mapping avec la base de données (ancien model).
- Repository : Couche d'accès aux données et requêtes JPA.
- Service : Logique métier et règles de gestion.
- Controller : Exposition des endpoints REST et gestion des requêtes HTTP.

Cette structuration améliore la maintenabilité, la testabilité et facilite l'évolution future du système.

3. Sécurisation et tests de l’application

3.1 Sécurisation avec Spring Security

L'application intègre Spring Security pour assurer :

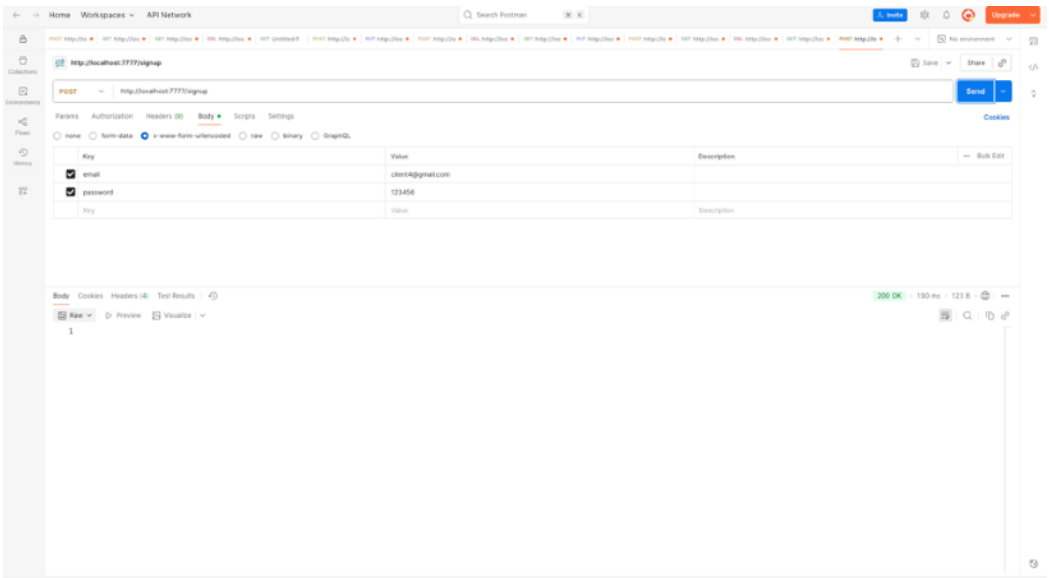
- Authentification : Gestion de l'identification des utilisateurs.
- Autorisation : Contrôle d'accès basé sur les rôles (RBAC - Role-Based Access Control).
- Protection des endpoints : Sécurisation des API selon les permissions utilisateur.

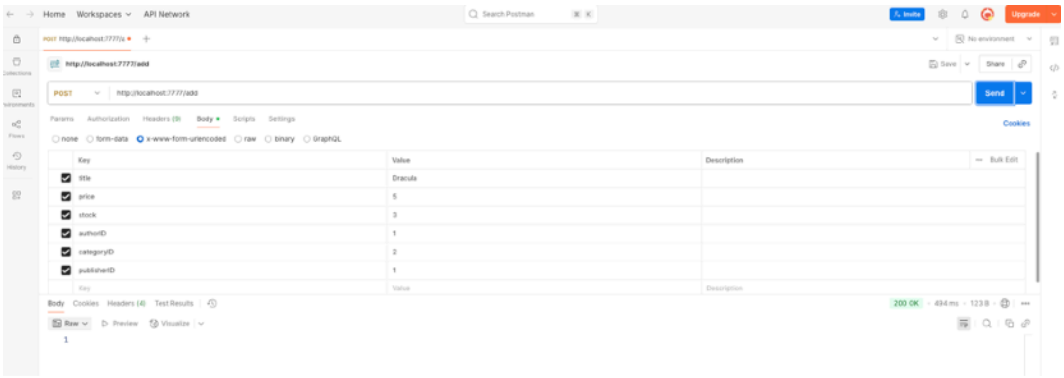
3.2 Stratégie de tests

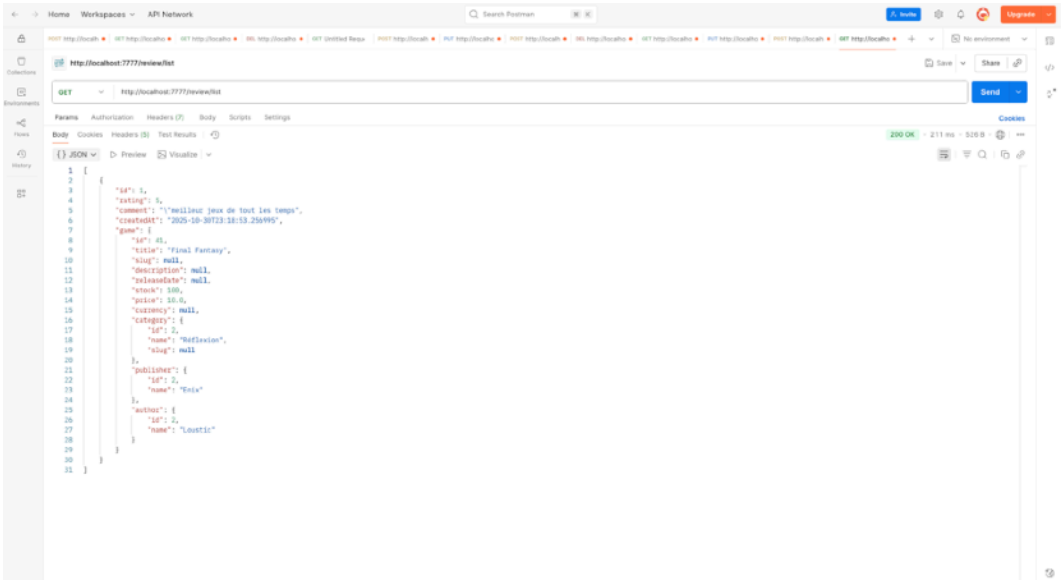
La qualité du code est garantie par une couverture de tests exhaustive :

- Tests unitaires : Toutes les fonctions de la couche Service sont couvertes par des tests unitaires, assurant la fiabilité de la logique métier.

- Tests d'intégration : Les Controllers ont été testés de manière continue via Postman tout au long du développement. Voici des screen shots de tests réalisé avec Postman que l’on juge pertinent de montrer.







- Couverture globale : 68% de couverture de tests, garantissant une détection précoce des régressions. Coté couche service, le code est couvert à 88%, ce qui est la couche la plus importante.

gamesUP

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.gamesup.controller		14 %		n/a	16	21	32	38	16	21	0	5
com.gamesup.service		88 %		83 %	3	25	12	90	2	22	0	4
com.gamesup.security		9 %		n/a	2	3	7	8	2	3	0	1
com.gamesup.entity		71 %		n/a	2	4	5	14	2	4	2	4
com.gamesup		37 %		n/a	1	2	2	3	1	2	0	1
com.gamesup.config		100 %		n/a	0	8	0	10	0	8	0	1
Total	217 of 690	68 %	1 of 6	83 %	24	63	58	163	23	60	2	16

com.gamesup.service

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PurchaseServiceImpl		84 %		75 %	1	7	6	32	0	5	0	1
UserServiceImpl		80 %		100 %	1	7	4	21	1	6	0	1
GameServiceImpl		95 %		n/a	1	7	2	25	1	7	0	1
ReviewServiceImpl		100 %		n/a	0	4	0	12	0	4	0	1
Total	44 of 393	88 %	1 of 6	83 %	3	25	12	90	2	22	0	4

Le rapport de couverture des tests démontre le taux de code testé par classe et par package. Un objectif d'environ 90% a été fixé pour garantir la qualité et la fiabilité du système, tout en évitant une sur-ingénierie des tests.

4. Mise en place d'un système de recommandation

4.1 Architecture du système

Un système de recommandation basé sur le machine learning a été développé pour enrichir l'expérience utilisateur. Ce système repose sur une API Python distincte utilisant l'algorithme K-Nearest Neighbors (KNN).

Le diagramme de composants illustre l'interaction entre l'API Spring et l'API Python. Il montre comment les deux systèmes communiquent via des appels REST pour générer des recommandations de jeux personnalisées, tout en maintenant une séparation claire des responsabilités.

4.2 Fonctionnement de l'algorithme KNN

L'algorithme KNN identifie les jeux similaires en se basant sur leurs caractéristiques (catégories, mécaniques, complexité) et génère des recommandations pondérées selon les préférences utilisateur.

5. Respect des principes SOLID

5.1 Présentation des principes SOLID

Les principes SOLID constituent un ensemble de bonnes pratiques de conception orientée objet visant à produire un code maintenable et évolutif :

- S - Single Responsibility Principle : Chaque classe a une responsabilité unique
- O - Open/Closed Principle : Ouvert à l'extension, fermé à la modification
- L - Liskov Substitution Principle : Les sous-types doivent être substituables à leurs types de base
- I - Interface Segregation Principle : Préférer plusieurs interfaces spécifiques à une interface générale
- D - Dependency Inversion Principle : Dépendre des abstractions plutôt que des implémentations concrètes

5.2 Application dans le projet

Single Responsibility Principle : La séparation en couches Entity/Repository/Service/Controller garantit qu'une classe ne gère qu'une seule préoccupation. Par exemple, les Services contiennent uniquement la logique métier, tandis que les Repositories ne gèrent que l'accès aux données.

Open/Closed Principle : L'utilisation d'interfaces pour les Services et Repositories permet d'ajouter de nouvelles implémentations sans modifier le code existant.

Dependency Inversion Principle : Les Controllers dépendent des interfaces de Services (abstractions) plutôt que des implémentations concrètes, facilitant les tests et l'évolution du code.

Le diagramme de classes met en évidence l'application des principes SOLID à travers la structure du code. On y observe notamment l'utilisation d'interfaces, l'héritage, et les relations entre les différentes entités du système, démontrant une architecture bien pensée et respectueuse des bonnes pratiques.

6. Machine Learning : réflexion et bonnes pratiques

6.1 Choix de l'algorithme KNN

L'algorithme K-Nearest Neighbors a été retenu pour ce projet en raison de sa simplicité conceptuelle et de son efficacité dans le contexte de recommandations item-based. Contrairement aux algorithmes plus complexes nécessitant un volume important de données d'entraînement, le KNN offre de bons résultats même avec un dataset modeste.

6.2 Bonnes pratiques appliquées

Séparation des préoccupations : L'algorithme ML est isolé dans une API Python dédiée, permettant une évolution indépendante du système de recommandation sans impacter l'application principale.

Normalisation des données : Les caractéristiques des jeux sont normalisées (StandardScaler) avant le calcul de similarité, garantissant que chaque dimension contribue équitablement au résultat.

Métriques pertinentes : L'utilisation de la distance cosine est adaptée aux données catégorielles et permet de mesurer efficacement la similarité entre jeux.

Architecture modulaire : Le système est conçu pour faciliter l'évolution vers des algorithmes plus sophistiqués (collaborative filtering, deep learning) sans restructuration majeure.

6.3 Points de vigilance et axes d'amélioration

Limitations du POC actuel :

- Données synthétiques : Le système utilise actuellement des données de démonstration
- Absence de feedback loop : Le modèle ne s'améliore pas automatiquement avec les interactions utilisateur
- Scalabilité : Le KNN peut devenir lent avec un très grand nombre de jeux (>10 000)

Améliorations futures envisagées :

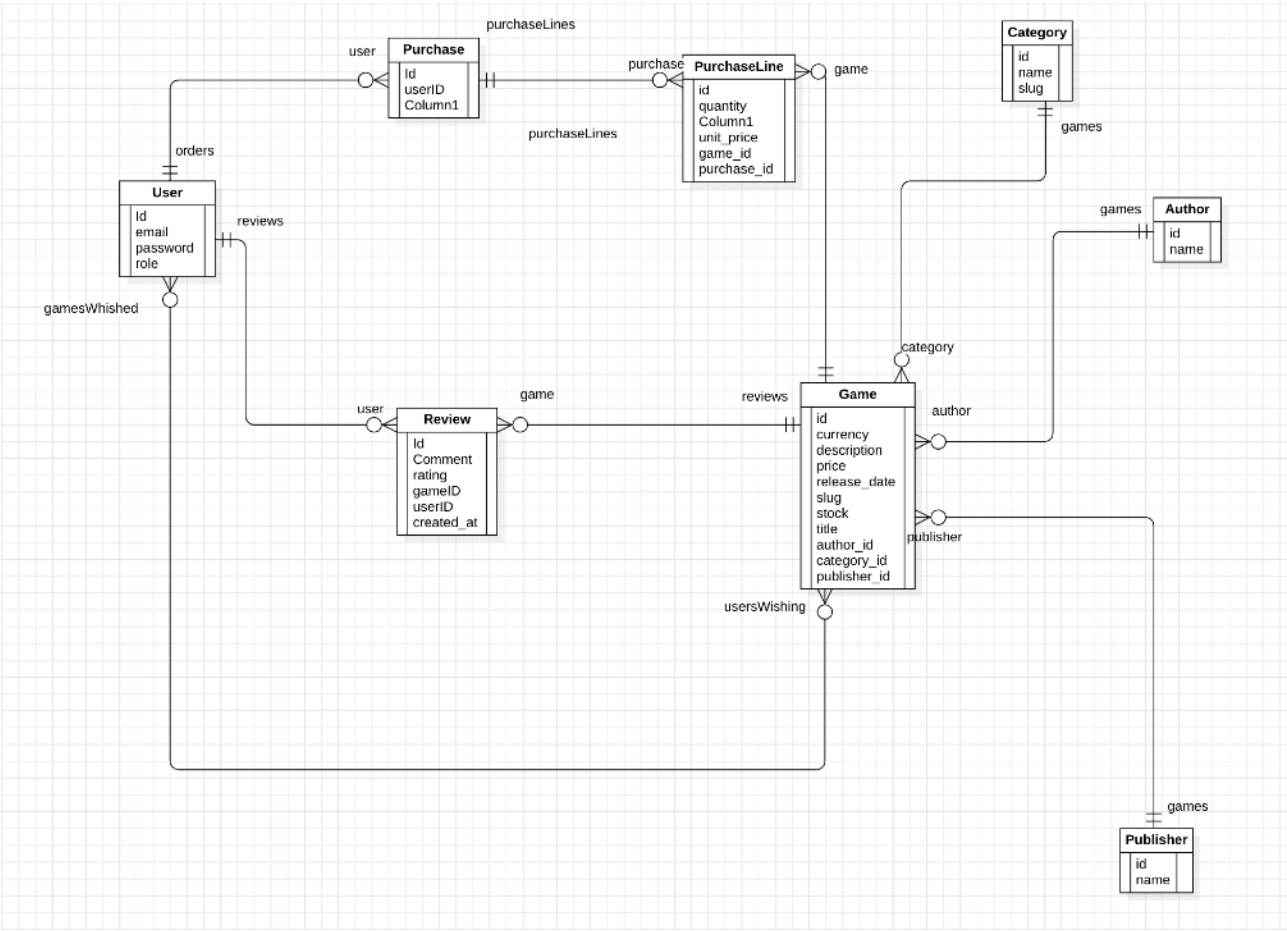
- Intégration de données utilisateur réelles pour des recommandations personnalisées
- Mise en place d'un système de collecte de feedback pour améliorer le modèle
- Exploration d'algorithmes plus avancés (Matrix Factorization, Neural Collaborative Filtering)
- Implémentation d'un système de cache pour optimiser les performances

7. Conclusion

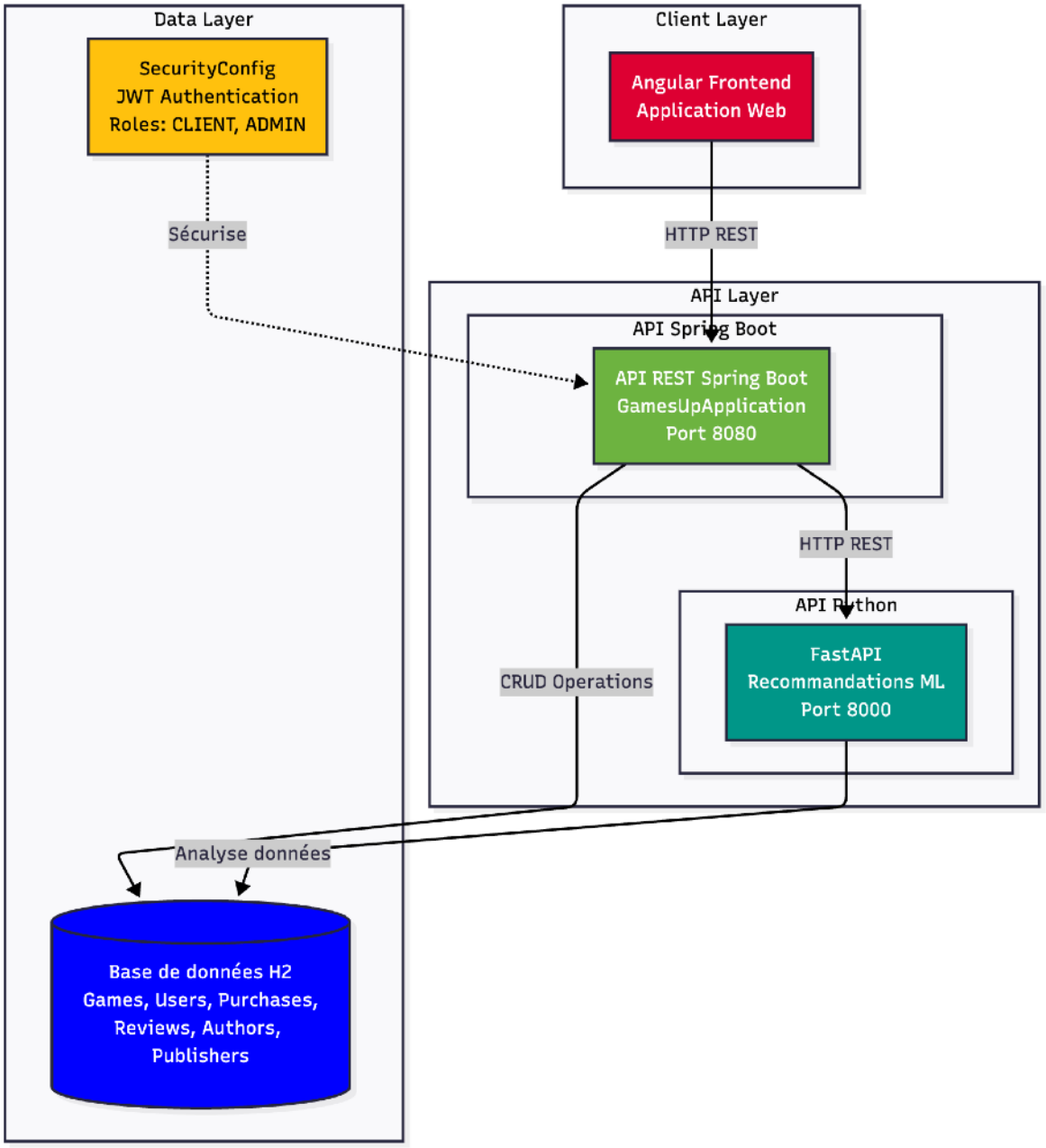
Ce projet démontre une démarche complète de refonte d'application, alliant bonnes pratiques de développement, sécurisation, qualité du code et innovation technologique. La mise en place d'une architecture solide, documentée par des diagrammes clairs, garantit la maintenabilité et l'évolutivité du système. L'intégration d'un système de recommandation par machine learning ouvre des perspectives d'amélioration continue de l'expérience utilisateur.

Annexes :

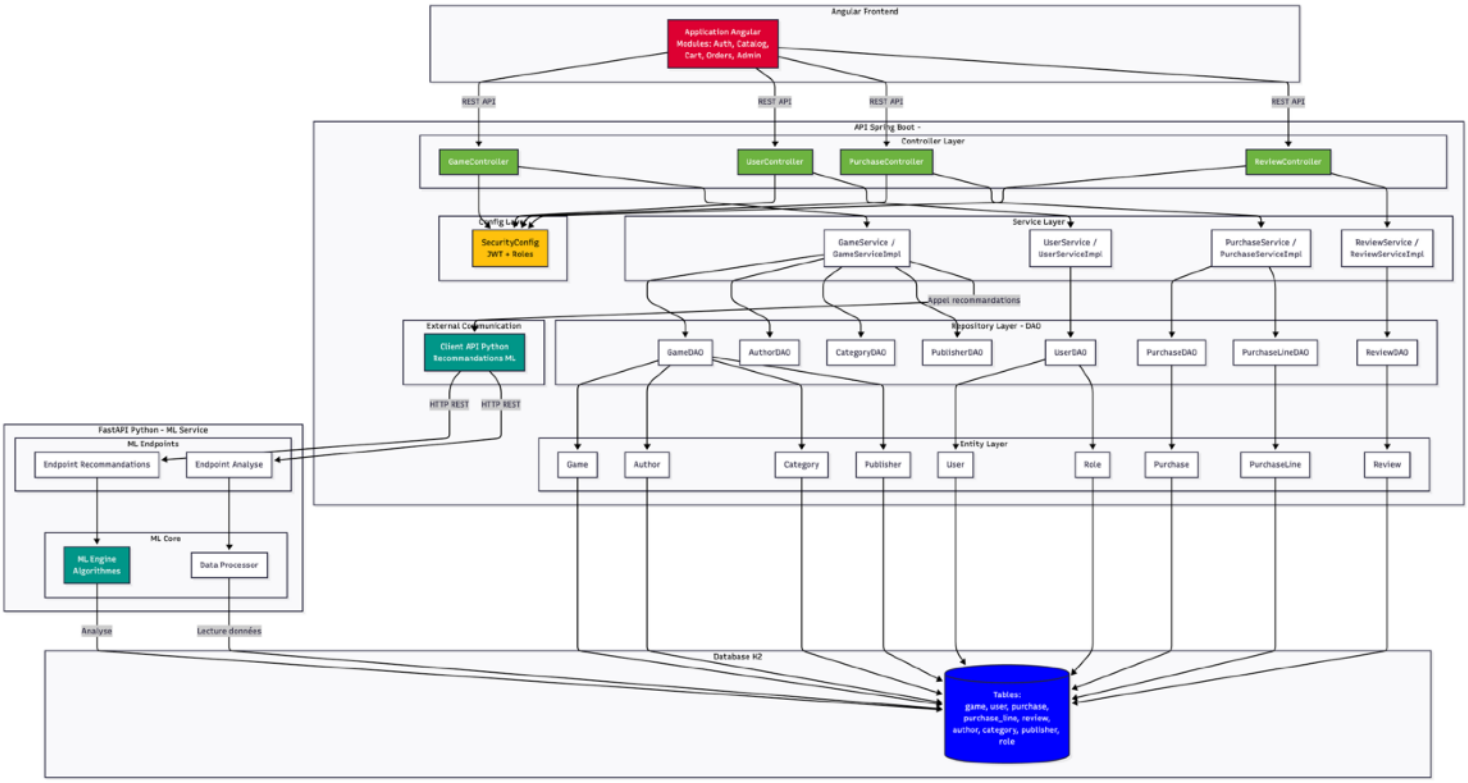
Diagrammes de classes :



Diagrammes d’architecture :

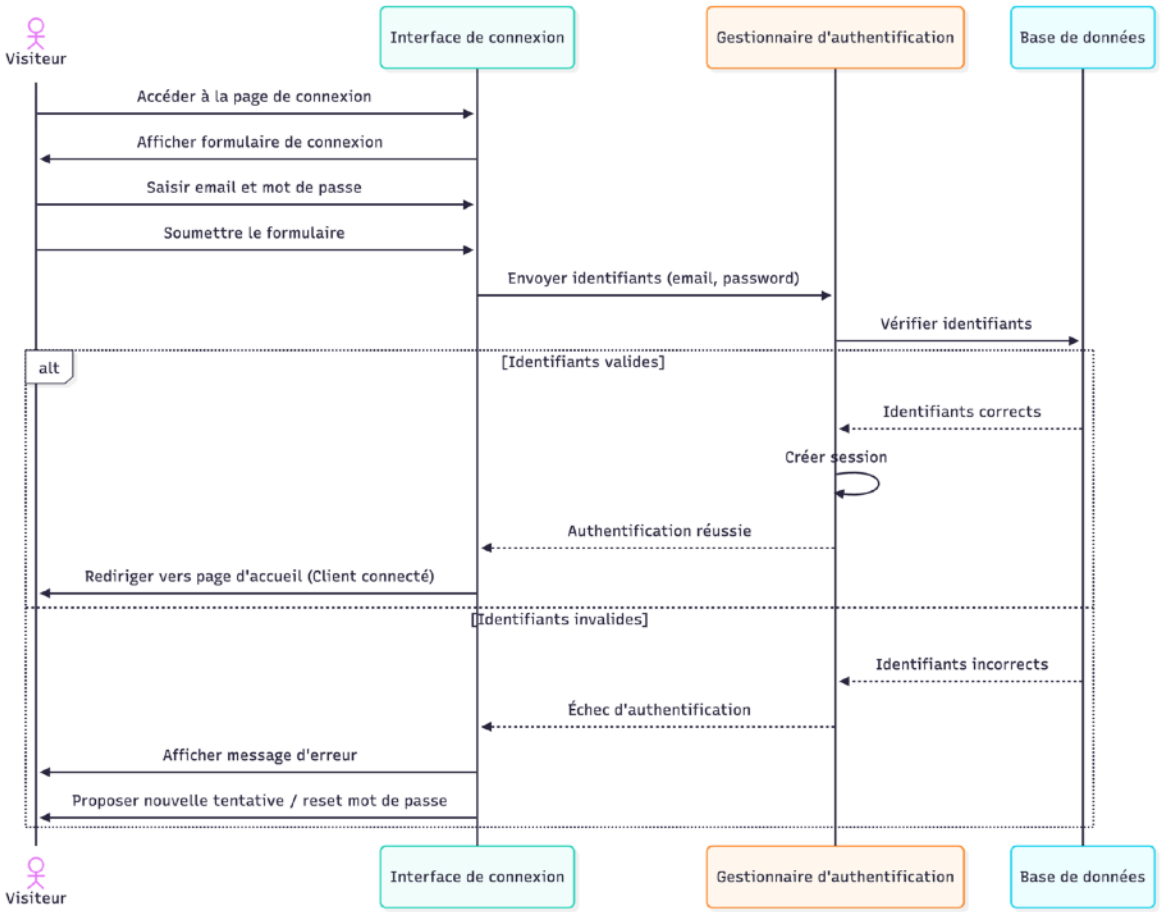


Diagrammes de composants :

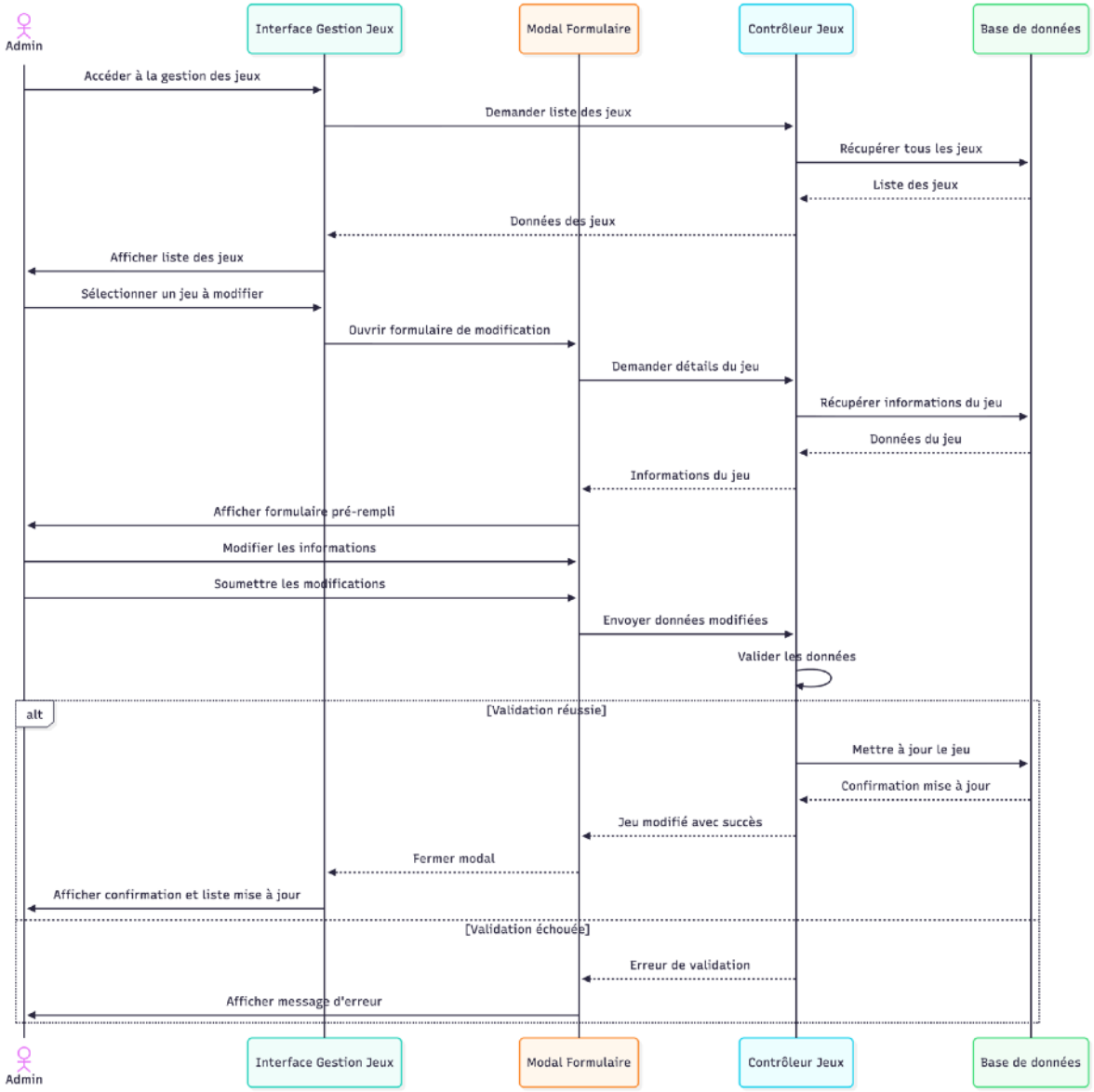


Diagrammes de séquences :

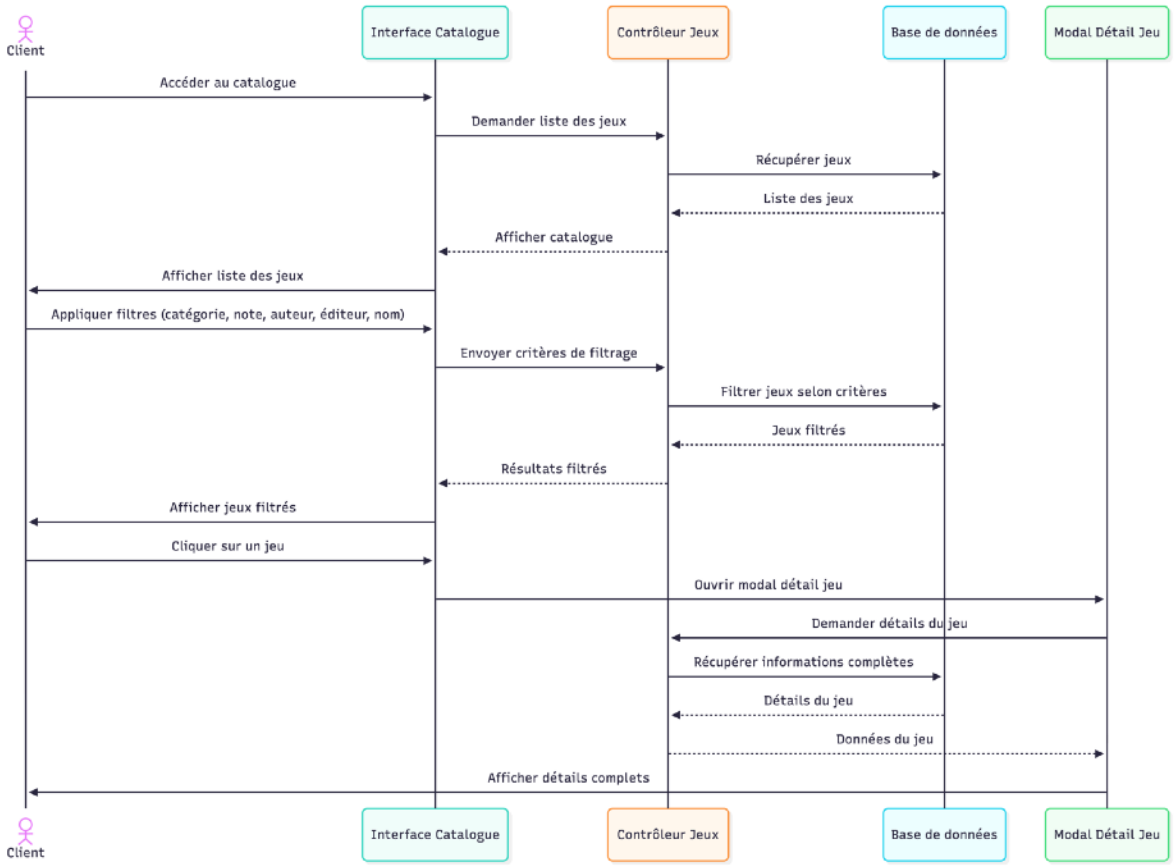
Authentification



Modifier un jeu :



Parcourir catalogue :



Acheter un jeu :