Projet Architecture orientée service

Master 2 Miage : Ingénierie Logicielle Pour le web

Présenté par :

- BASAK Roni
- SAADI Adel
- GRASSART Baptiste
- AYADI Naila
- MEZIANI Keltoum

2025-2026

Lien Github : https://github.com/Slivix/Projet_AOS

1. Vue globale du projet

- **Nom** : Puissance X
- **Objectif** : Jeu de type Puissance X jouable en ligne ou en local, avec une authentification et un compte pour chaque utilisateur, avec une inclusion d'une partie multi-joueurs.
- **Principes**: architecture orientée services (microservices ou services modulaires), une interface web comportant un Front-End et un Back-end, une base de données pour stocker les informations relatives aux jeux et aux joueurs et un modèle d'intelligence artificielle.

2. Acteurs et cas d'utilisation

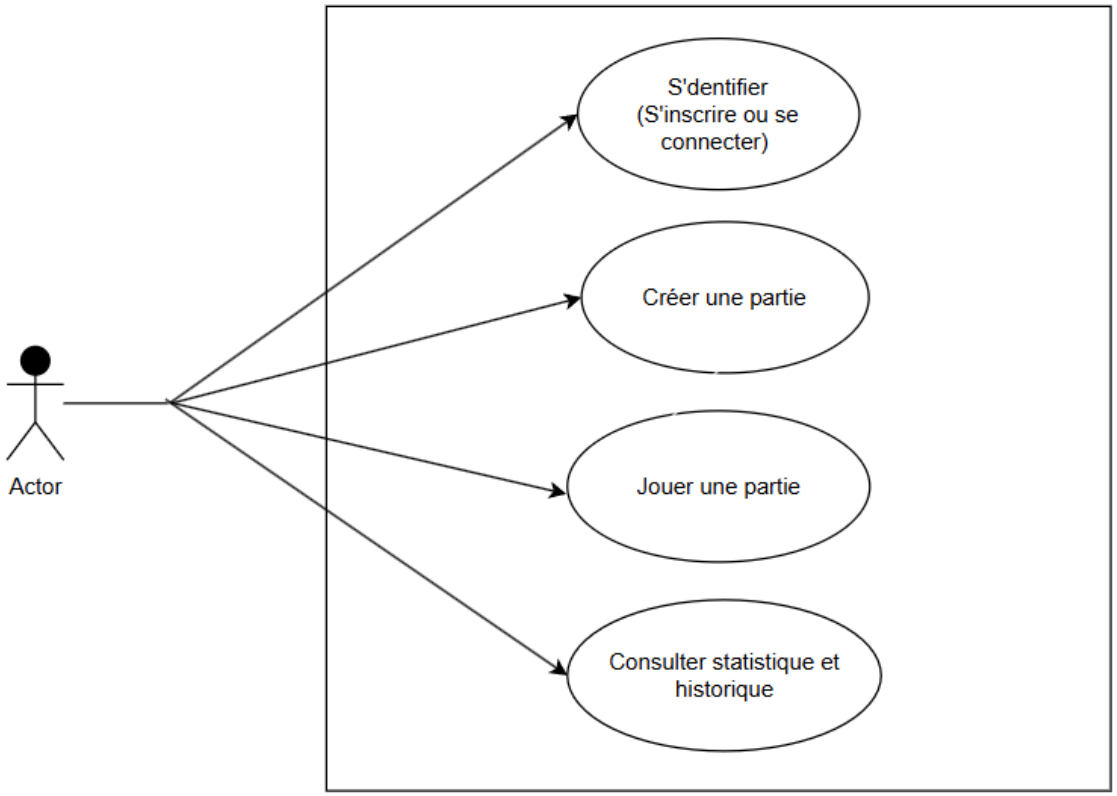
2.1. Acteurs

- Joueur authentifié
- Adversaire (joueur)
- Joueur IA

2.2. Cas d'utilisation

- Identification : S'inscrire (nouveau joueur) ou se connecter (ancien joueur).
- Créer une partie (choix de grille, conditions victoire).
- Rejoindre une partie (avec le code de partie).
- Jouer une partie (en temps réel).
- Jouer contre IA (sélection difficulté).

2.3. Diagramme de cas d'utilisation



3. Architecture orientée service

3.1. Architecture du projet

- Un frontend séparé pour l'interface frontend (interface web, affichage du jeu)
- Un service pour la logique de jeu, backend (API)
- docker-compose.yml

Cela permet d'avoir des composants indépendants qui communiquent entre eux (via HTTP/API).

3.2. Technologies utilisées

Partie	Technologies
FrontEnd	HTML / CSS / JavaScript
Backend	Python (Flask)
Authentification	Python
Déploiement	Docker & Docker-Compose

4. Fonctionnement global

à l'exécution :

- Le joueur s'authentifie (pseudo, email, mot de passe)
- Le frontend envoie les requêtes au backend
- Le backend gère :
    - La création/join d'une partie
    - Le parcours du jeu
    - La synchronisation des tours

5. Modélisation des données

La modélisation des données du projet repose sur une approche orientée objet, définie à partir des modèles Pydantic utilisés dans l'application. Ces modèles représentent les entités métier principales et servent de référence pour la structure logique des données.

L'entité **User** correspond aux utilisateurs de l'application. Elle est persistée afin de gérer l'authentification et le score des joueurs.

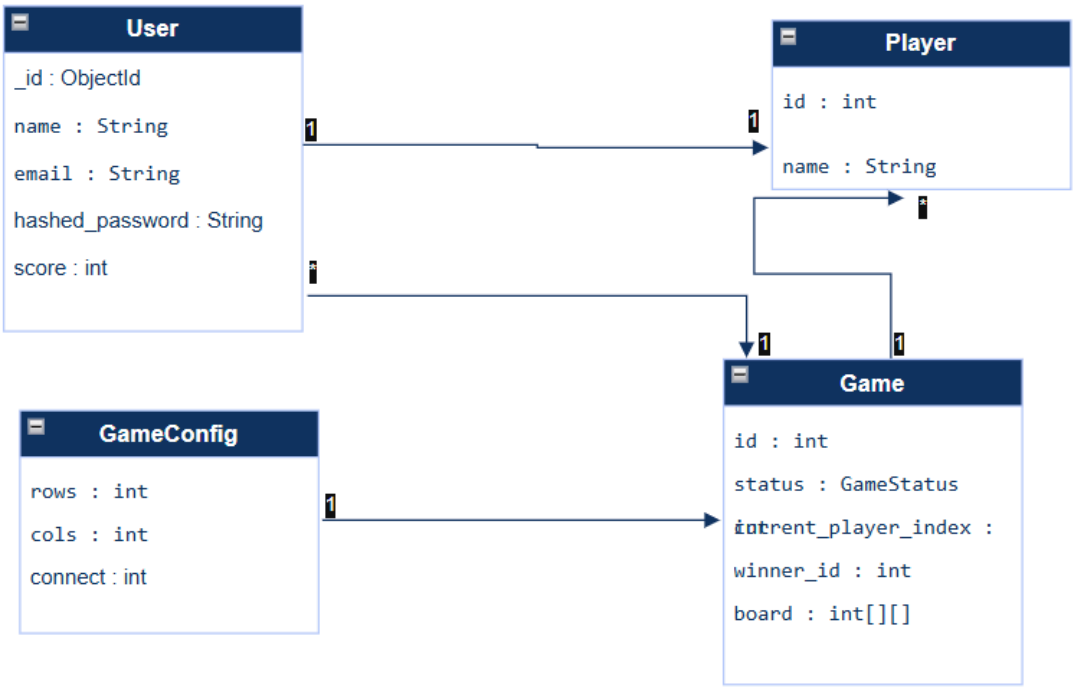
L'entité **Player** représente un joueur participant à une partie. Elle est liée à un utilisateur et permet d'associer un joueur à une partie spécifique.

L'entité **Game** modélise une partie de jeu. Elle contient l'état courant de la partie (statut, joueur actif, gagnant) ainsi que le plateau de jeu, représenté sous forme de tableau bidimensionnel. Cette structure permet une gestion dynamique des parties.

L'entité **GameConfig** définit les paramètres d'une partie, notamment le nombre de lignes, de colonnes et le nombre de pions à aligner pour gagner. Chaque partie possède une configuration unique.

Les relations entre ces entités permettent une vision globale et cohérente du système. Cette modélisation unifiée intègre à la fois les besoins de persistance (utilisateurs) et la logique métier du jeu (parties et joueurs), tout en restant adaptée à l'architecture du projet.

- **Diagramme UML**



6. Integration de l'intelligence artificielle

Objectif

Ce module permet d'affronter une intelligence artificielle directement dans le navigateur, garantissant une réactivité maximale sans solliciter le serveur.

1. Architecture et Implémentation

- **Localisation :** Implémenté exclusivement côté client via **game_script.js.**
- **Indépendance Backend :** l'IA simule un second joueur local.
- **Réutilisation :** Utilise les mêmes structures de données (board, players) que le mode multijoueur local.

2. Logique de Décision (Heuristique)

L'IA suit une stratégie de décision en trois niveaux de priorité, exécutée après un court délai (setTimeout) pour améliorer l'expérience utilisateur (UX) :

1. **Offensif :** Si l'IA peut gagner au prochain coup, elle le joue immédiatement.
2. **Défensif :** Si le joueur menace de gagner au tour suivant, l'IA bloque sa progression.
3. **Aléatoire :** À défaut de menace ou d'opportunité immédiate, elle choisit une colonne valide au hasard.

3. Fonctions Principales

Fonction	Rôle
aiPickColumn(state)	Point d'entrée : choisit la colonne finale selon la stratégie.
aiWouldWin(board, col, pid, connect)	Simule un coup pour vérifier s'il mène à une victoire.
aiDrop(board, col, pid)	Calcule la position de chute du jeton (gestion de la gravité).
aiCheckWin(...)	Vérifie l'alignement des pions sur la grille.

4. Intégration et Interface (UI)

- **Configuration :** Dans **game.html**, le mode "IA" verrouille automatiquement le champ "Adversaire".
- **Feedback :** Affichage dynamique des statuts : *"À toi de jouer"* ou *"L'IA réfléchit..."*.
- **Données :** Exploite la matrice **state.board** (0 : vide, 1 : joueur, 2 : IA).

7. Detection de victoire

Le backend doit vérifier si un alignement (ligne/colonne/diagonale) est gagné.

1. On regarde à partir de la dernière case jouée
2. On compte dans **4 directions** :
    - horizontal
    - vertical
    - diagonale /
    - diagonale \
3. Si on trouve ≥ n pions alignés → **victoire**

Conclusion

Ce projet a permis de développer une application web complète autour du jeu *Puissance X*, en appliquant les notions de programmation orientée objet, de modélisation des données et d'architecture client–serveur. Le projet est fonctionnel et constitue une base solide pouvant être améliorée par la suite.

Difficultés rencontrées

- Gestion de l'état du jeu : difficulté à maintenir la cohérence des données tout au long de la partie, notamment lors des actions successives des joueurs et des changements de règles.
- Synchronisation des actions : nécessité de coordonner correctement les événements pour éviter les conflits ou les comportements inattendus.
- Gestion des dépendances : problèmes rencontrés lors de l'installation et de la compatibilité des bibliothèques entre les différents services.
- Communication entre services : mise en place des échanges réseau entre les conteneurs et gestion des ports.
- Débogage et tests : difficulté à identifier l'origine des erreurs dans un environnement conteneurisé.