Projet Fullstack

Nom du projet : Cristal

Nom du groupe : Cristaline

1. Description du projet

Cristal est une plateforme web permettant de référencer des jeux vidéo et de suivre son expérience de joueur, inspirée d'un concept similaire à MyLetterboxd mais appliqué aux jeux vidéo.

L'objectif du MVP est de proposer un hub simple où les utilisateurs peuvent :

- Consulter un catalogue de jeux
- Noter des jeux
- Ajouter des jeux en favoris
- Voir les favoris d'autres utilisateurs
- Importer des jeux depuis une source externe gratuite
- Démontrer une communication distribuée via Kafka

Le périmètre est volontairement limité afin de garantir la qualité, la stabilité et la reproductibilité du projet.

---

2. Architecture technique

Frontend

- Angular
- Architecture MVC

- Communication avec le backend via une REST API

Backend

- Java Spring Boot
- Architecture 3-tiers :
    - Controller
    - Service
    - Repository
- REST API
- Kafka pour la communication asynchrone via des topics

Base de données

- H2 (base embarquée pour le développement et les tests)

DevOps et Build

- GitHub pour la gestion du code source (main + branches de fonctionnalités)
- GitHub Actions pour l'intégration continue (build + tests + rapport de couverture)
- Docker pour la conteneurisation
- docker-compose pour exécuter la stack complète (backend + Kafka + frontend)

Organisation du repository

Structure retenue

Le projet Cristal est organisé en **monorepo** avec la structure suivante :

- `/cristal`
- `├── /api      → Backend Spring Boot`
- `├── /front    → Frontend Angular`

- ├── docker-compose.yml
- ├── README.md

Justification du choix

1. Cohérence du périmètre

Le projet constitue un MVP unique avec :

- Un frontend Angular
- Une API Spring Boot
- Une base H2
- Kafka

Il s'agit d'un seul produit applicatif, donc un seul repository est logique.

---

3. Fonctionnalités du MVP

Catalogue

- Liste des jeux
- Détail d'un jeu
- Recherche par titre
- Filtrage par genre ou année

Notes

- Un utilisateur peut attribuer une note à un jeu

- Calcul et affichage de la note moyenne

Favoris

- Ajouter un jeu à ses favoris
- Voir la liste de ses favoris
- Voir les favoris d’un autre utilisateur

Administration

- Ajouter un jeu au catalogue
- Modifier les informations d’un jeu
- Supprimer un jeu
- Importer un jeu depuis une API externe gratuite

Kafka

- Publication d’un événement lors :
    - De l’ajout d’un jeu
    - De la modification d’un jeu
    - De l’import d’un jeu externe
- Possibilité d’utiliser un topic dédié pour les demandes d’import, consommé par un service backend
-

1. Ajout du mécanisme d’inscription et de sécurité

1.1 Objectif

L’ajout d’un système d’inscription et d’authentification permet :

- La gestion personnalisée des favoris et des notes.
- La distinction entre utilisateurs standards et administrateurs.
- La protection des opérations sensibles du catalogue.

La solution retenue repose sur Spring Security avec authentification par JWT (JSON Web Token).

1.2 Modèle Utilisateur

Un modèle User est introduit dans le système. Il contient :

- Identifiant unique
- Nom d’utilisateur
- Adresse email
- Mot de passe
- Rôle (USER ou ADMIN)

Les mots de passe ne sont jamais stockés en clair. Ils sont hachés avec l’algorithme BCrypt afin de garantir leur protection en cas de compromission de la base de données.

1.3 Authentification par JWT

L’authentification repose sur un mécanisme stateless.

Fonctionnement :

1. L’utilisateur s’inscrit via un endpoint public.
2. Il se connecte avec ses identifiants.
3. Le serveur vérifie les informations.
4. Un token JWT signé est généré et retourné au client.

5. Le frontend inclut ce token dans l'en-tête Authorization de chaque requête.

Ce mécanisme évite la gestion de session côté serveur et facilite la scalabilité de l'application.

1.4 Gestion des rôles et autorisations

Deux rôles sont définis :

- USER : accès aux fonctionnalités standards (consultation, notes, favoris).
- ADMIN : accès aux opérations de gestion (ajout, modification, suppression, import).

La sécurisation des routes suit le principe de moindre privilège :

- Les routes d'authentification sont publiques.
- Les routes métier nécessitent une authentification.
- Les routes d'administration nécessitent le rôle ADMIN.

1.5 Mesures de sécurité complémentaires

Les mesures suivantes sont mises en place :

- Désactivation du CSRF pour une API REST stateless.
- Configuration CORS restreinte au frontend Angular.
- Validation des données en entrée.
- Gestion centralisée des erreurs d'authentification et d'autorisation.

1.6 Processus d'inscription avec validation par email

Objectif

Le mécanisme d’inscription permet :

- La création sécurisée d’un compte utilisateur.
- La vérification de l’adresse email.
- La prévention des comptes frauduleux ou automatisés.
- Le renforcement global de la sécurité du système.

1.6.1 Processus global d’inscription

Le processus d’inscription est structuré en plusieurs étapes afin de garantir la validité des données et la sécurité du compte.

Étapes fonctionnelles :

1. L’utilisateur accède au formulaire d’inscription sur le frontend Angular.
2. Il renseigne :
    - Nom d’utilisateur
    - Adresse email
    - Mot de passe
3. Le frontend envoie une requête au backend via un endpoint public.
4. Le backend :
    - Vérifie que l’email n’est pas déjà utilisé.
    - Vérifie la validité des données (format email, complexité mot de passe).
    - Hache le mot de passe avec BCrypt.
    - Crée un compte utilisateur avec le statut "non activé".
5. Le système génère un token de validation unique associé à l’utilisateur.

6. Un email est envoyé contenant un lien de confirmation.
7. L'utilisateur clique sur le lien reçu.
8. Le backend valide le token et l'email de l'utilisateur et active définitivement le compte.

Le compte devient alors utilisable.

1.6.2 Activation du compte par email

Afin de garantir que l'adresse email fournie est valide et contrôlée par l'utilisateur, un mécanisme d'activation est mis en place.

Fonctionnement :

- Lors de l'inscription, un token unique est généré.
- Ce token est stocké en base de données avec :
    - Une date d'expiration
    - Une référence à l'utilisateur
- Un email est envoyé contenant un lien du type :

  https://application.com/activate?token=XYZ
- Lorsque l'utilisateur clique sur ce lien :
    - Le backend vérifie la validité du token.
    - Si le token est valide et non expiré, le compte est activé.
    - Le token est ensuite invalidé.

Ce mécanisme permet d'éviter :

- Les inscriptions avec des emails invalides.
- Les créations massives automatisées.
- L'usurpation d'identité.

1.6.3 Envoi d’email

L’envoi d’email est réalisé côté backend via un service SMTP.

Caractéristiques :

- Utilisation d’un serveur SMTP (ex : Gmail, Mailtrap pour développement).
- Envoi d’un email HTML simple contenant :
    - Message de bienvenue
    - Lien d’activation
- Gestion des erreurs d’envoi.

En environnement de développement, un service de type Mailtrap peut être utilisé afin d’éviter l’envoi réel d’emails.

1.6.4 Sécurisation du mécanisme

Les mesures suivantes sont appliquées :

- Le mot de passe est haché avant stockage.
- Le token d’activation est aléatoire et difficilement prédictible.
- Le token possède une durée de validité limitée.
- Un compte non activé ne peut pas se connecter.
- Les messages d’erreur ne révèlent pas d’informations sensibles.

1.6.5 Intégration dans l’architecture existante

Le mécanisme d’inscription s’intègre dans l’architecture 3-tiers :

Controller
→ Réception des requêtes d’inscription et d’activation

Service
→ Validation des données
→ Hachage du mot de passe
→ Génération du token
→ Envoi d'email

Repository
→ Persistance de l'utilisateur et du token

Résultat attendu

À l'issue de ce mécanisme :

- L'utilisateur peut créer un compte sécurisé.
- Son adresse email est vérifiée.
- Le système distingue les comptes activés et non activés.
- L'authentification JWT n'est possible qu'après activation.

---

2. Intégration de l'API externe FreeToGame

2.1 Objectif

L'API FreeToGame (https://www.freetogame.com/api/games) est utilisée pour importer automatiquement des jeux gratuits dans le catalogue.

Cela permet :

- D'enrichir rapidement la base de données.
- De démontrer l'intégration d'un service externe.
- D'illustrer une architecture extensible.

2.2 Processus d’import

Le processus d’import est le suivant :

1. Un administrateur déclenche un import via un endpoint sécurisé.
2. Le backend envoie une requête HTTP vers l’API externe.
3. Les données JSON reçues sont transformées en objets du modèle interne.
4. Les jeux sont enregistrés en base de données.
5. Un événement Kafka est publié pour chaque jeu importé.

L’appel à l’API externe est réalisé côté backend afin de garantir la maîtrise du flux de données et d’éviter l’exposition directe de services tiers au frontend.

2.3 Transformation des données

Les données issues de l’API externe sont adaptées au modèle interne :

- Mapping des champs.
- Conversion des formats de date.
- Filtrage des jeux déjà existants.
- Validation des données avant persistance.

Cette étape assure la cohérence du modèle métier et évite les incohérences en base.

---

3. Intégration avec Kafka

Conformément aux exigences du projet, chaque action importante génère un événement :

- Ajout d’un jeu.
- Modification d’un jeu.

- Import d’un jeu externe.

Ces événements sont publiés dans un topic dédié.

Objectifs :

- Démontrer une communication asynchrone.
- Illustrer une architecture distribuée.
- Permettre une extensibilité future (logs, analytics, notifications).

Un service consommateur peut également traiter les demandes d’import via un topic spécifique.

---

4. Intégration dans l’architecture existante

L’ajout de la sécurité et de l’import respecte l’architecture 3-tiers :

- Controller : gestion des requêtes HTTP.
- Service : logique métier (authentification, import, publication Kafka).
- Repository : accès aux données via JPA.

La sécurité est appliquée transversalement via un filtre JWT.

L’import externe est isolé dans un service dédié afin de respecter le principe de séparation des responsabilités.

---

4. User Stories

En tant qu’utilisateur :

Je veux voir la liste des jeux disponibles.
Je veux consulter le détail d'un jeu.
Je veux rechercher un jeu par son titre.
Je veux filtrer les jeux par genre ou année.
Je veux ajouter un jeu à mes favoris.
Je veux voir la liste de mes jeux favoris.
Je veux voir les jeux favoris d'un autre utilisateur.
Je veux noter un jeu.

En tant qu'administrateur :

Je veux ajouter un jeu au catalogue.
Je veux modifier les informations d'un jeu.
Je veux supprimer un jeu du catalogue.
Je veux importer des jeux depuis une source externe gratuite.
Je veux que chaque ajout, modification ou import publie un événement via Kafka.

Users use cases :

UC-01 – Consulter le catalogue de jeux

Acteur principal : Utilisateur
Précondition : Aucune
Postcondition : La liste des jeux est affichée

Scénario nominal

1. L'utilisateur accède à la page catalogue.
2. Le frontend envoie une requête GET à l'API.
3. Le backend récupère les jeux en base.
4. La liste des jeux est affichée.

Extensions

- 3a. Aucun jeu disponible → affichage d'un message "Aucun jeu disponible".

UC-02 – Consulter le détail d’un jeu

Acteur principal : Utilisateur
Précondition : Le jeu existe
Postcondition : Les informations détaillées du jeu sont affichées

Scénario nominal

1. L’utilisateur clique sur un jeu.
2. Le frontend envoie une requête GET avec l’identifiant.
3. Le backend retourne les informations détaillées.
4. Les détails sont affichés (description, genre, année, note moyenne).

Extensions

- 2a. Jeu inexistant → message d’erreur.

UC-03 – Rechercher un jeu par titre

Acteur principal : Utilisateur
Précondition : Aucune
Postcondition : Liste filtrée affichée

Scénario nominal

1. L’utilisateur saisit un titre dans la barre de recherche.
2. Le frontend envoie une requête avec paramètre `title`.
3. Le backend filtre les résultats.
4. Les jeux correspondants sont affichés.

UC-04 – Filtrer les jeux par genre ou année

Acteur principal : Utilisateur
Précondition : Aucune
Postcondition : Liste filtrée affichée

Scénario nominal

1. L'utilisateur sélectionne un filtre (genre/année).
2. Le frontend envoie la requête correspondante.
3. Le backend retourne les jeux filtrés.
4. Les résultats sont affichés.

UC-05 – Créer un compte

Acteur principal : Utilisateur non authentifié
Précondition : Email non utilisé
Postcondition : Compte créé en statut "non activé"

Scénario nominal

1. L'utilisateur remplit le formulaire d'inscription.
2. Les données sont envoyées au backend.
3. Le backend valide les données.
4. Le mot de passe est haché (BCrypt).
5. Le compte est créé (non activé).
6. Un token d'activation est généré.
7. Un email de confirmation est envoyé.

Extensions

- 3a. Email déjà utilisé → message d’erreur.
- 3b. Données invalides → message de validation.

UC-06 – Activer son compte

Acteur principal : Utilisateur inscrit
Précondition : Token valide et non expiré
Postcondition : Compte activé

Scénario nominal

1. L’utilisateur clique sur le lien reçu par email.
2. Le backend vérifie le token.
3. Le compte est activé.
4. Le token est invalidé.

Extensions

- 2a. Token expiré ou invalide → message d’erreur.

UC-07 – Se connecter

Acteur principal : Utilisateur activé
Précondition : Compte activé
Postcondition : JWT généré

Scénario nominal

1. L’utilisateur saisit email et mot de passe.
2. Le backend vérifie les identifiants.
3. Un JWT est généré.

4. Le token est renvoyé au frontend.
5. Le frontend l’ajoute aux requêtes futures.

Extensions

- 2a. Mauvais identifiants → erreur.
- 2b. Compte non activé → accès refusé.

UC-08 – Ajouter un jeu en favori

Acteur principal : Utilisateur authentifié
Précondition : Utilisateur connecté
Postcondition : Jeu ajouté aux favoris

Scénario nominal

1. L’utilisateur clique sur “Ajouter aux favoris”.
2. Le frontend envoie une requête authentifiée.
3. Le backend enregistre l’association User–Game.
4. Confirmation affichée.

Extensions

- 2a. Jeu déjà en favori → message informatif.

UC-09 – Consulter ses favoris

Acteur principal : Utilisateur authentifié
Précondition : Utilisateur connecté
Postcondition : Liste des favoris affichée

Scénario nominal

1. L’utilisateur accède à sa page “Favoris”.
2. Le backend récupère les jeux associés.
3. La liste est affichée.

UC-10 – Consulter les favoris d’un autre utilisateur

Acteur principal : Utilisateur authentifié
Précondition : Utilisateur cible existant
Postcondition : Liste affichée

Scénario nominal

1. L’utilisateur consulte le profil d’un autre utilisateur.
2. Le backend récupère ses favoris.
3. Les favoris sont affichés.

UC-11 – Noter un jeu

Acteur principal : Utilisateur authentifié
Précondition : Utilisateur connecté
Postcondition : Note enregistrée et moyenne recalculée

Scénario nominal

1. L’utilisateur attribue une note.
2. Le frontend envoie la note au backend.
3. Le backend enregistre ou met à jour la note.
4. La moyenne est recalculée.

5. La nouvelle moyenne est affichée.

Extensions

- 3a. Note déjà existante → mise à jour.

UC-12 – Se déconnecter

Acteur principal : Utilisateur authentifié
Précondition : Utilisateur connecté
Postcondition : Session supprimée côté client

Scénario nominal

1. L'utilisateur clique sur "Déconnexion".
2. Le frontend supprime le JWT.
3. L'accès aux routes sécurisées est bloqué.

Planning du projet avec priorisation

Sprint 1 – 1 semaine

Objectif : Mise en place du socle technique et du catalogue

Priorité Haute (Critique pour livrable)

- Initialisation du repository GitHub
- Mise en place du backend Spring Boot
- Mise en place du frontend Angular
- Configuration de la base H2

- Création du modèle Game
- Implémentation du CRUD complet des jeux :
    - Liste
    - Détail
    - Création
    - Modification
    - Suppression
- Recherche par titre
- Filtrage par genre ou année

Ces éléments constituent le socle fonctionnel minimal. Sans eux, le MVP n’est pas exploitable.

Priorité Moyenne (Stabilité et qualité)

- Premiers tests unitaires (service + repository)
- Dockerisation du backend

Ces éléments assurent la qualité et la portabilité, mais ne bloquent pas la démonstration fonctionnelle immédiate.

Priorité Basse (Optimisation)

Aucune tâche classée en priorité basse sur ce sprint, car il est centré sur la mise en place du cœur applicatif.

Livrable attendu en fin de sprint

- API fonctionnelle pour la gestion du catalogue
- Frontend permettant d’afficher la liste et le détail des jeux

---

Sprint 2 – 2 semaines

Objectif : Fonctionnalités utilisateurs et intégration Kafka

Fonctionnalités supplémentaires

Priorité Haute (Fonctionnel MVP)

- Création d'un modèle User simplifié
- Système de favoris
- Système de notes
- Calcul de la note moyenne
- Import d'un jeu depuis une API externe gratuite

Ces fonctionnalités définissent la valeur ajoutée principale du projet par rapport à un simple CRUD.

Kafka

Priorité Haute (Exigence projet)

- Mise en place du cluster Kafka
- Implémentation d'un Producer
- Publication d'événements lors de l'ajout, modification ou import d'un jeu

Ces éléments sont essentiels pour démontrer l'architecture distribuée demandée.

Priorité Moyenne

- Implémentation d’un Consumer
- Traitement d’une demande d’import via un topic dédié

Ces éléments renforcent la démonstration technique mais peuvent être simplifiés si le temps devient contraint.

Qualité et configuration

Priorité Haute

- docker-compose complet (backend + frontend + Kafka)
- README expliquant le lancement du projet

Ces éléments sont indispensables pour la reproductibilité du projet.

Priorité Moyenne

- Ajout de tests unitaires complémentaires
- Tests d’intégration
- Mise en place d’une pipeline GitHub Actions
    - Build
    - Exécution des tests
    - Rapport de couverture
- Endpoint Actuator /health

Ces tâches améliorent la robustesse et la démonstration professionnelle du projet.

Priorité Basse

- Documentation Swagger

Elle améliore la lisibilité technique mais ne bloque ni l’exécution ni la démonstration du MVP.

Livrable attendu en fin de sprint

- MVP complet avec gestion des favoris et des notes
- Import externe fonctionnel
- Kafka démontré
- Application entièrement exécutable via docker-compose
