

Projet Fullstack

Contexte :

Conception libre en équipe de 3 personnes minimum d'une API backend et son mini Frontend

Contrainte technique nécessitant une capacité d'appropriation rapide d'une technologie :
L'architecture doit inclure un cluster Kafka, un ou plusieurs communications via des topics
devront être implémentées

Sujet libre, thème : Plateforme de référencement de jeux-vidéos

Technologies Backend : Java, Spring-Boot, Kafka
Technologies de Front : libres (MVP attendu)

Outillage : DevOps, Docker

Votre contact :
David THIBAU
david.thibau@gmail.com

Déroulé du projet et points de validation:

Journée de lancement : Jeudi 19 Février

Présentation projet, constitution des groupes, échanges questions/réponses

1er Milestone 26 Février :

- Mise en place plateforme DevOps
- Document de conception incluant :
 - User Stories
 - Techno et architecture identifiées

- Planning des sprints

Livrable :

Le documentation de conception

2nd Milestone : 27 Mars

- Revue de code,
- Amélioration pipeline CI/CD
- Définition des environnements d'intégration et de production

Livrable : MVP fonctionnel

- Dépôt Git,
- Documentation de démarrage/installation de l'application

Release et Soutenance 24 Avril

- Présentation :
 - Preuve du respect des spécifications : Couverture des Tests $\geq 70\%$
 - Pipeline CI/CD complète
 - Dimensionnement de l'environnement de production
 - Release 1.0 stocké dans un artefact repository
 - Démonstration de la 1.0 en condition de production

Exemple d'attendus fonctionnels (MVP) :

Gestion d'un catalogue de jeux-vidéos

Fonctionnalités minimales attendues :

- Liste des jeux
- Détail d'un jeu
- Création / modification / suppression (au moins création + liste)
- Différents mode d'alimentation du catalogue par partenaire externe

- Recherche ou filtrage simple (ex : genre, année)

Sécurité (niveau attendu) :

Validation des entrées (DTO + contraintes)

Gestion d'erreurs cohérente (codes HTTP, messages, exceptions)

Mesure de sécurité simple (au choix) :

- Protection des endpoints d'écriture (auth simple / clé API / configuration Spring Security), ou
- Justification d'un périmètre volontairement non authentifié, avec mesures minimales (ex: limitation, séparation des routes)

Un audit de sécurité doit être passé avec un agent dont le rôle "expert sénior en sécurité" aura été au préalable défini.

Ressources à mobiliser :

- Justifications techniques (choix, compromis, limites)
- Découpage du projet (couches / modules / packages)
- Documents de conception (architecture, modèle de données, flux Kafka)
- Mobilisation de nouvelles connaissances (Spring Boot, Kafka, Docker, front)
- Conception (API, DTO, erreurs, persistance)
- Respect d'une architecture cohérente et maintenable
- Capacité à travailler en équipe (organisation, conventions, revues)
- Définition des priorités (MVP, jalons, arbitrages)
- Définition des tests à mettre en place (stratégie et périmètre)
- S'intégrer dans une chaîne de production (CI simple, exécution automatique)
- Capacité à tenir des délais (planning, "done", stabilisation)

DevOps (simple) :

Code source sous Git (main + branches de feature)

Pipeline CI à chaque push/PR incluant au minimum :

- Build
- Exécution des tests
- Rapport de couverture
- Containerisation :
- Packaging sous forme d'images docker

- docker-compose pour exécuter la stack complète en environnement CI, tests d'intégration ou e2e
Monitoring :
- Endpoint de santé (ex : Actuator /health)
- Logs lisibles et exploitables
Documentation technique
- README permettant un lancement "from scratch" (pré-requis, run, tests, démo)

Livrables attendus :

- Code source sous Git
- Projet fonctionnel (backend + preuve Kafka + mini frontend)
- Documents de conception et d'analyse (format court)
- Documentation Swagger / OpenAPI
- Tests unitaires
- Tests d'intégration (au minimum sur API/service ou persistance)
- Test e2e (si frontend) ou scénario bout-en-bout reproductible
- Rapports de couverture de tests
- Configuration CI (GitHub Actions / GitLab CI / Jenkins)
- Images Docker + docker-compose
- Démo et présentation

Compétences mobilisées

Intermédiaires

Gérer

Anticiper un risque à l'échelle d'une solution informatique à taille humaine

Piloter

S'investir dans la vie d'un projet à taille humaine

Concevoir

S'inscrire dans une démarche de conception visant à proposer des solutions nouvelles évaluables

Produire

Produire des solutions informatiques distribuées, répondant à des besoins techniques spécifiés et garanties par des protocoles de tests

Compétent

Produire

Industrialiser une solution informatique : automatiser, déployer, maintenir, documenter

Concevoir

Intégrer une solution dans son écosystème en garantissant sa pérennité

Formaliser

Rechercher la validation client par le biais d'une analyse multi-dimensionnelle (fonctionnel, technique, risques, exploitation)

Piloter

Porter la responsabilité collective d'un projet, tout au long de son cycle de vie

Évaluation APC

Cohérence globale :

Sujet cohérent avec l'APC : situation réaliste, livrables observables, articulation conception → production → industrialisation

L'intégration Kafka ajoute une dimension "distribution" pertinente au niveau de compétence "Produire"

L'ajout DevOps minimal rend crédible la compétence "Industrialiser" au niveau attendu

Points de vigilance :

Risque de dérive de périmètre lié au Front "libre" : nécessité d'un MVP explicitement limité

Sécurité à cadrer au niveau "minimum attendu" pour éviter une exigence floue

Kafka à évaluer sur preuves (flux démontré, message structuré, documentation courte), pas uniquement sur présence de dépendances

Preuves attendues

Code + fonctionnement démontrable (API + Kafka)

Traces de pilotage (issues, commits, PR/MR, conventions)

Tests + couverture (qualité mesurée)

Reproductibilité (CI + Docker + README)

Conception justifiée (documents courts, décisions, limites)

Grille d'évaluation

Niveaux :

- 1 – Insuffisant : incomplet / non fonctionnel / non démontrable
- 2 – Fragile : fonctionne partiellement, qualité/justification limitée
- 3 – Satisfaisant : répond au besoin, cohérent, reproductible
- 4 – Excellent : robuste, propre, bien industrialisé, choix assumés

Produire – Solution distribuée testée (REST + Kafka)

- 1 : API incomplète, Kafka absent/non démontré, pas de tests.
- 2 : API OK partiellement, Kafka présent mais fragile, peu de tests.
- 3 : API fonctionnelle, Kafka démontré, tests pertinents.
- 4 : API + Kafka robustes, cas limites gérés, tests solides.

Concevoir – Architecture et pérennité

- 1 : pas d'architecture claire, dette forte.
- 2 : architecture simple mais incohérente ou non justifiée.
- 3 : architecture claire (couches/modules), décisions justifiées.
- 4 : conception propre, évolutive, décisions argumentées + limites.

Piloter – Travail d'équipe et cycle de vie

- 1 : pas d'organisation, contributions non traçables.
- 2 : organisation minimale, peu de revues, priorités floues.
- 3 : issues/PR/MR, revues, jalons respectés, priorités explicites.
- 4 : pilotage fluide, arbitrages tracés, amélioration continue.

Industrialiser – CI + Docker + documentation

1 : pas de CI, lancement non reproductible.

2 : CI ou Docker partiel, README incomplet.

3 : CI complète (build+tests+coverage), docker-compose OK, README clair.

4 : industrialisation propre + bonus (qualité statique, versioning, etc.).

Gérer / Anticiper – Risques (solution à taille humaine)

1 : risques ignorés, blocages non gérés.

2 : risques cités mais parades faibles.

3 : risques identifiés + parades réalistes (Kafka, planning, intégration).

4 : gestion proactive, décisions de réduction de scope pertinentes.