EHEI Oujda / GI & IG

Année académique 2025 - 2026

Fiche Projet de Fin d'Année

CommitEd : PLATEFORME D'ÉVALUATION INTELLIGENTE

Une approche Dev-First pour l'enseignement supérieur

Réalisé par :

EL AISSAOUI Iliass
MAKOURI Loqman
SGHIOURI IDRISSI Youness
TALEB Fayza

Sous l'encadrement de :

M. MOUHIB Imad

01 Décembre 2025

Table des matières

Liste des Illustrations

Figures

Tableaux

1 Proposition de Projet

1.1 Problématique

Les plateformes éducatives actuelles (type Google Classroom) manquent de spécificité pour l'enseignement de l'informatique. Elles traitent le code comme du texte brut, ignorant tout l'écosystème de développement (Tests, Versioning, Qualité).

Le besoin est de créer une plateforme qui intègre le workflow professionnel (Git, GitHub) directement dans le processus pédagogique, tout en offrant aux professeurs des outils d'analyse automatisés pour suivre la progression réelle des étudiants.

1.2 Méthodologie et Organisation

Pour mener à bien ce projet complexe en équipe, nous adoptons une approche rigoureuse :

- **Méthodologie Scrum :** Développement itératif par « Sprints » courts pour valider chaque module fonctionnalité par fonctionnalité.
- **Collaboration DevOps :** Utilisation intensive de Git & GitHub pour la gestion de versions.
- **Documentation Centralisée :** Mise en place d'un portail `/docs` (type Swagger/ OpenAPI) pour documenter nos APIs et faciliter l'interconnexion de nos micro-services.

1.3 Plan de Développement (Roadmap)

Le projet suivra 5 phases distinctes pour assurer une montée en charge progressive :

Phase 1 - Le Socle « Classroom » Implémentation des fonctionnalités fondamentales : gestion des utilisateurs, création de classes, publication de devoirs simples et soumission de fichiers.

Phase 2 - Statistiques de Base Développement d'un dashboard permettant au professeur de visualiser les taux de soumission, les retards et l'engagement basique des étudiants.

Phase 3 - Intégration GitHub (GitOps) L'automatisation du workflow technique. Le professeur publie un « Template », l'étudiant « Fork » le projet, et la soumission se fait via un « Push ». La plateforme gère les liaisons API avec GitHub.

Phase 4 - Analyse de Code (IA vs Fallback) Deux approches sont envisagées :

- *Plan A (Agentique) :* Un agent IA analyse le code étudiant (respect des principes SOLID, Clean Code) et génère un rapport automatique.
- *Plan B (Fallback) :* Un système de notation paramétrique permettant au professeur d'évaluer manuellement le code sur des métriques précises.

Phase 5 - Modèle Prédictif (ML) Développement d'un modèle ML en Python qui croise les rapports d'analyse et les notes pour prédire les difficultés futures des étudiants et suggérer des actions correctives.

1.4 Architecture & Stack Technologique

Nous avons opté pour une architecture **Micro-services** afin de découpler les responsabilités métier. Cette approche « Polyglotte » nous permet d'appliquer le principe du **« Best Tool**

for the Job » (le meilleur outil pour la tâche) plutôt que de nous enfermer dans un écosystème unique.

L'architecture globale est illustrée dans la Fig. 1.

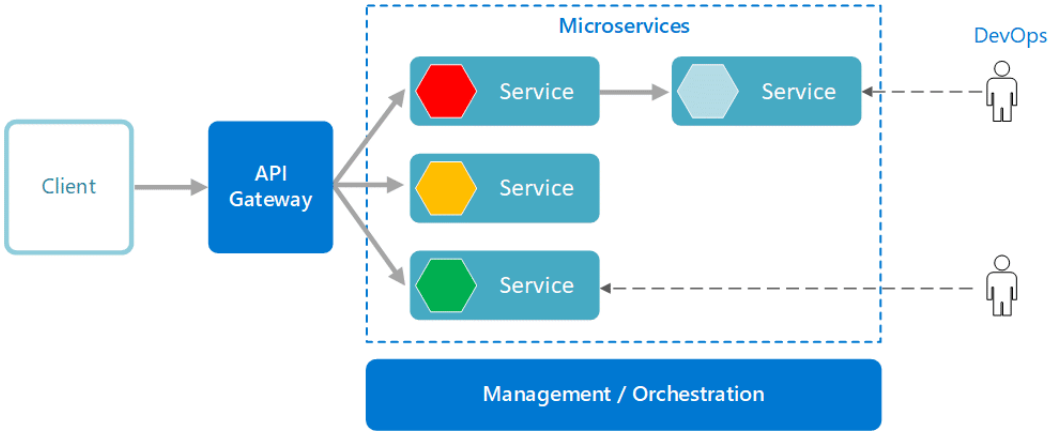

Fig. 1. – Diagramme d'Architecture Micro-services

1.4.1 Justification des Choix Technologiques

Chaque micro-service a des contraintes spécifiques (performance, rapidité de développement, écosystème IA) qui ont dicté le choix de la technologie :

Tableau 1. – Matrice de décision technologique

Service & Techno	**Justification du Choix**
Frontend *React.js*	**Pourquoi React ?** Son architecture à base de composants et le DOM virtuel offrent une fluidité indispensable pour une SPA. **Pourquoi ici ?** Le dashboard nécessite une gestion d'état complexe (temps réel, graphiques) que React gère nativement.
Service Core *Symfony (PHP)*	**Pourquoi Symfony ?** Framework mature pour le développement rapide avec un ORM (Doctrine) puissant. **Pourquoi pour le Core ?** Ce service gère la logique « administrative ». Symfony permet de développer ces CRUD 2x plus vite qu'en Java.
Service Orchestrator *Spring Boot (Java)*	**Pourquoi Spring ?** L'écosystème Java est inégalé pour la stabilité, le typage fort et la gestion du multithreading. **Pourquoi pour l'Orchestrator ?** Ce service est le moteur critique (Git, Docker, I/O intensifs). La robustesse de la JVM est impérative ici.
Service Analytics *ASP.NET Core (C#)*	**Pourquoi .NET ?** Performance d'exécution proche du C++ et la bibliothèque LINQ, atout majeur pour la data. **Pourquoi pour l'Analytics ?** LINQ nous permet d'écrire des requêtes de filtrage complexes de manière plus lisible et performante que du SQL pur.
Service Intelligence *Python (FastAPI)*	**Pourquoi Python ?** La lingua franca de l'IA et de la Data Science (Pandas, Scikit-learn).
Infrastructure *Docker & Oracle*	**Docker :** Indispensable pour l'isolation et la création de « Sandboxes » éphémères pour le code étudiant. **Oracle Database :** Choisi pour sa conformité ACID stricte et pour simuler un environnement « Grand Compte ».

1.5 Répartition Prévisionnelle des Tâches

- **Membre 1 :** Architecture Frontend & Intégration UX.
- **Membre 2 :** Backend Core (Symfony) & Base de données.
- **Membre 3 :** Backend Orchestrator (Spring) & Intégration GitHub API.
- **Membre 4 :** Services Analytics (.NET) & Intelligence Artificielle (Python).

2 Plan de Montée en Compétence Technologique

Date : 09 December 2025
Contexte : Préparation à la phase de développement (Début prévu : J+30)
Objectif : Acquérir les compétences techniques requises pour l'architecture Micro-services Polyglotte validée.

2.1 Socle Commun & Outils Transverses (Toute l'équipe)

Ces compétences sont obligatoires pour garantir la collaboration DevOps et la qualité documentaire.

Ressources recommandées :

- Git: Atlassian Tutorials, GitHub Docs, Pro Git (livre gratuit en ligne).
- Docker: Documentation Docker officielle, Udemy/Coursera courses, Docker Hands-on Labs.
- Typst: Typst Official Documentation, community examples sur GitHub.

Technologie	Modules & Détails d'Apprentissage	Durée Estimée
Git & GitHub (Flow)	• Stratégie de branches (GitFlow ou GitHub Flow). • Gestion des Pull Requests (Code Review). • Résolution de conflits avancée. • Utilisation des GitHub Actions (CI/CD basique). • Mini-projet : Configurer un workflow GitFlow sur un repo test avec 2-3 branches.	2 Jours
Docker & Conteneurisation	• Écriture de `Dockerfile` • `Docker Compose` pour orchestrer les 5 services en local. • Notions de réseaux Docker (Bridge) pour la comm. inter-services. • Mini-projet : Créer un `docker-compose.yml` pour lancer 2-3 services localement et tester la communication.	4 Jours
Typst (Documentation)	• Syntaxe de base et mise en page. • Création de Templates pour les rapports.	1 Jour

Technologie	Modules & Détails d'Apprentissage	Durée Estimée
	• Gestion des bibliographies et références croisées. • Mini-projet : Créer un template de rapport pour la documentation du projet.	

2.2 Formations Spécifiques par Rôle (Micro-services)

2.2.1 Membre 1 : Frontend & UX (Service Frontend)

Responsable de l'interface utilisateur et des tableaux de bord interactifs.

Approche pédagogique : Apprendre par la construction. Démarrer avec un composant simple (ex: liste d'étudiants), puis progresser vers un dashboard avec état global et graphiques. Parallèle avec Figma pour les maquettes.

Ressources :

- React: Official React Docs (version 18+), scrimba.com, udemy.
- Tailwind: Tailwind CSS Docs + Tailwind UI components.
- Figma: Figma Learning (YouTube), design systems basics.

Technologie	Modules Spécifiques	Durée Estimée
React.js (Ecosystème)	• React Hooks (`useState`, `useEffect`, `useContext`). • Gestion d'état global (Zustand ou Redux Toolkit). • **Routing** : React Router v6 (Protection des routes). • Gestion des erreurs et loading states. • Mini-projet : Composant « Gestion d'Étudiants » avec liste, création, suppression (état local puis global).	8 Jours
UI & Dataviz	• **Tailwind CSS** : Configuration, Grid system, Responsive design. • **Recharts** ou **Chart.js** : Pour les graphiques du Dashboard (Phase 2). • **Axios** : Intercepteurs pour gérer les tokens JWT. • Mini-projet : Dashboard basique avec 2-3 graphiques et appels API mockés.	3 Jours

Technologie	Modules Spécifiques	Durée Estimée
Figma	• Maquettage des écrans principaux, prototypes interactifs. • Collaboration et handoff aux développeurs (export, design tokens). • Wireframes et prototypes des 5 écrans clés (Login, Classes, Assignments, Dashboard, Student Profile). • Mini-projet : Prototype interactif d'au moins 3 écrans avec transitions.	3 Jours

2.2.2 Membre 2 : Backend Core (Service Gestion)

Responsable de la logique administrative (CRUD Étudiants, Classes) et de la persistance principale.

Approche pédagogique : Construire un CRUD robuste (Symfony) + connaissance profonde de la BD (Oracle, cursors, vues). Exécuter localement avec Docker Compose. Intégration avec le Frontend en Sprint 1.

Ressources :

- Symfony: Documentation Symfony 6/7, SymfonyCasts (screencasts pratiques).
- Doctrine ORM: Doctrine Documentation officielle, exemples avancés.
- Oracle: Oracle Learning Paths (gratuit), SQL Tuning Advisor dans SQL Developer.

Technologie	Modules Spécifiques	Durée Estimée
Symfony 7 (PHP)	• Structure MVC et Injection de Dépendances. • Composant **Serializer** (Transformation Entity → JSON). • Validation des données (Asserts). • Sécurité (Guard Authenticator pour JWT). • Gestion d'erreurs et logging. • Mini-projet : API CRUD complet pour Étudiants et Classes avec validation et JWT.	8 Jours
Base de Données	• **Doctrine ORM** : Mapping avancé, Relations, Migrations.	6 Jours

Technologie	Modules Spécifiques	Durée Estimée
	• **Oracle Database** : Connexion via OCI8/PDO_OCI, spécificités SQL Oracle. • Concepts SQL utiles pour le front : Cursors, Vues (views) et Fonctions/ Procédures stockées pour faciliter l'extraction et la transformation des données vers le front-end. • Requêtes préparées, transactions et optimisation de base (indexation, plan d'exécution). • Gestion des users, permissions et auditing. • Mini-projet : Schéma BD complet avec cursors pour export de données, vues pour agrégation, et fonctions pour cas métier complexes.	

2.2.3 Membre 3 : Orchestrator (Service Moteur)

Le cœur critique. Doit manipuler Git programmatiquement et gérer les conteneurs étudiants.

Approche pédagogique : Maîtriser Spring Boot en tant qu'orchestrateur central. Deux défis majeurs : (1) Cloner et analyser des repos Git via JGit, (2) Lancer des conteneurs étudiants via Docker API. Commencer par chacun isolément, puis intégrer.

Ressources :

- Spring Boot: Spring.io (guides et documentation), Baeldung tutorials, Spring Academy (cours officiel gratuit).
- JGit: Eclipse JGit documentation, exemples GitHub.
- Docker Java API: docker-java library (GitHub: docker-java/docker-java — actively maintained), official Docker SDK documentation.

Technologie	Modules Spécifiques	Durée Estimée
Spring Boot 3 (Java)	• Spring Web (REST Controllers). • Gestion asynchrone (`@Async` pour les tâches longues). • **Spring Security** (Validation des tokens Gateway).	8 Jours

Technologie	Modules Spécifiques	Durée Estimée
	• Gestion des exceptions et monitoring basique. • Mini-projet : API REST pour orchestrer 2-3 tâches simples (créer un conteneur, récupérer un fichier).	
Automatisation & I/O	• **Eclipse JGit** : Librairie Java pour cloner/analyser des repos Git (Phase 3). • **Docker Java API** : Pour lancer des conteneurs « Sandbox » depuis le code Java. • Gestion des fichiers (NIO.2) pour scanner le code étudiant. • Gestion des timeouts et ressources (mémoire, CPU des conteneurs). • Mini-projet : Script Java qui clone un repo de test, l'analyse (compte fichiers, lignes de code), et génère un rapport simple.	6 Jours

2.2.4 Membre 4 : Analytics & Intelligence (Services .NET & Python)

Responsable du traitement de données haute performance et de l'IA.

Approche pédagogique : Deux piliers : (1) .NET pour l'agrégation performante et LINQ, (2) Python pour Data Science et IA. Commencer par pandas/numpy, puis scikit-learn. Construire un mini-modèle prédictif dès la Semaine 3.

Ressources :

- ASP.NET Core: Microsoft Learn, Docs, PluralSight courses.
- LINQ: LINQ to Objects tutorials, Albahari's LINQ reference.
- Python: Official Docs, DataCamp ou Kaggle Courses, Kaggle Datasets.
- Scikit-learn: Scikit-learn User Guide, Hands-on ML with Scikit-learn (livre gratuit partiellement).

Technologie	Modules Spécifiques	Durée Estimée
ASP.NET Core (C#)	• Web API Architecture. • **LINQ** : Requêtes complexes sur collections et DB.	6 Jours

Technologie	Modules Spécifiques	Durée Estimée
	• Entity Framework Core (Optimisation des requêtes lecture seule). • Pagination et caching pour les gros volumes de données. • Mini-projet : API .NET qui agrège les données d'étudiants (via appels aux autres services) et expose des métriques simples (nombre de soumissions par classe, etc.).	
Python & Data Science	• **FastAPI** : Création d'endpoints rapides pour l'inférence. • **Pandas** : Nettoyage et structuration des données CSV/JSON. • **Scikit-learn** : Régression/Classification pour le modèle prédictif (Phase 5). • **LangChain** (Optionnel) : Si utilisation d'analyse de code via LLM. • Visualisation de données (Matplotlib, Seaborn) pour validations. • Mini-projet (Semaine 2-3) : Charger un dataset CSV fictif (résultats étudiants), le nettoyer avec pandas, entraîner un modèle simple (ex: prédire si l'étudiant réussira) avec scikit-learn, exposer un endpoint FastAPI pour l'inférence.	10 Jours

Note courte : actuellement en étude active — Python : `pandas`, `numpy`, `scikit-learn` (prise en main et exercices pratiques).

2.3 Plan d'Apprentissage Détaillé (4 Semaines) — Équilibre Conception + Technique

Principes : Ce mois prépare le **terrain pour le développement Phase 1**. L'accent est mis sur les **fondamentaux + contrats API + schémas BD**, pas sur une application fonctionnelle complète. La vraie production démarre en Mois 2.

2.3.1 Semaine 1 : Socle Commun & Mise en Pratique

Objectif : Maîtriser Git, Docker et les outils transversaux. Chaque équipe configure son environnement de base.

Travail parallèle :

- **Tous (jours 1-2) :** Git/GitHub Workflow (branches, PRs, merges, CI/CD basique avec Actions).
  - Mini-projet : Chacun crée une branche feature, pushes une PR avec description, code review en binôme.
  - Temps investi : **2 jours**.
- **Tous (jours 3-4) :** Docker & Compose fondamentaux (Dockerfile, images, réseaux, volumes).
  - Mini-projet : Créer 3 services simples (app web dummy + DB + cache) dans un `docker-compose.yml` et les faire communiquer.
  - Temps investi : **2 jours**.
- **Tous (jour 5) :** Typst basics pour la documentation.
  - Temps investi : **1 jour**.
- **En parallèle (Conception) :** Débuter les diagrammes UML (cas d'usage, séquences pour Phase 1).

Livrables fin Semaine 1 :

- Repos de chaque service initialisés avec structure de base + workflows GitHub.
- Docker Compose local fonctionnel pour tous les 5 services (dummy).
- Premiers diagrammes UML validés.

2.3.2 Semaine 2 : Socle Technique + Schéma de Données

Objectif : Apprendre les fondamentaux du langage assigné. Commencer à concevoir le schéma BD et les contrats d'API (OpenAPI).

Travail par rôle (apprentissage) :

- **Frontend (Membre 1) :** React Hooks + basic routing (2-3 jours) + intro Figma (1 jour).
  - Mini-projet : Composant stateless simple (ex: liste avec props, pas d'état global encore).
  - Ressources : React official docs, 1 scrimba course.
  - Temps : **4 jours**.
- **Backend Core (Membre 2) :** Symfony structure + Doctrine basics (3 jours) + schéma BD Oracle (2 jours).
  - Mini-projet : Scaffold une entité Symfony simple (ex: `Student`), créer les migrations.
  - Ressources : Symfony docs, SymfonyCasts.
  - Temps : **5 jours**.
- **Orchestrator (Membre 3) :** Spring Boot REST basics (2-3 jours) + intro Git API (1 jour).
  - Mini-projet : Simple REST endpoint (ex: `GET /health`, `POST /echo`).
  - Ressources : Spring.io guides, Baeldung.
  - Temps : **4 jours**.
- **Analytics (Membre 4) :** FastAPI + pandas basics (2-3 jours) + numpy (1 jour).
  - Mini-projet : FastAPI endpoint qui lit un CSV fictif et retourne des stats (moyenne, count).

- Ressources : FastAPI docs, Real Python.
- Temps : **4 jours**.

Travail parallèle (Conception) :

- **Backend Core + Orchestrator :** Concevoir le schéma BD Oracle (Phase 1 minimal : Users, Classes, Students, Assignments, Submissions).
- **Tous :** Définir les contrats d'API (OpenAPI specs) pour Phase 1 :
  - Frontend ← → Backend Core (authentification, CRUD classes/étudiants).
  - Frontend ← → Analytics (GET /stats).
  - Orchestrator ← → (pas de consommation externe Phase 1, juste tâches asynchrones en interne).

Livrables fin Semaine 2 :

- Chaque service expose une « hello world » API endpoint.
- Schéma BD Phase 1 (tables, relations) documenté en SQL + diagrammes ER.
- OpenAPI specs (YAML) pour Backend Core + Analytics endpoints.
- Wireframes des 3-4 écrans clés sur Figma (Login, Dashboard, Classes).

—

2.3.3 Semaine 3 : Intégration API & Approfondissement

Objectif : Comprendre comment les services vont parler. Approfondir les libs clés. Finaliser la conception.

Travail par rôle :

- **Frontend (Membre 1) :** Axios setup + mock API calls (1 jour) + Tailwind + Recharts intro (1-2 jours).
  - Mini-projet : Créer un écran de « liste d'étudiants » qui appelle un endpoint mocké (JSON local ou Postman mock server).
  - Temps : **3 jours**.
- **Backend Core (Membre 2) :** Doctrine relations avancées + Serializer pour JSON (1-2 jours) + JWT auth basics (1 jour) + cursors/vues Oracle (1-2 jours).
  - Mini-projet : Endpoints `GET /students`, `POST /students`, `GET /students/{id}` avec vraie BD Oracle (schema Phase 1).
  - Temps : **5 jours**.
- **Orchestrator (Membre 3) :** Spring async + Docker API basics (2 jours) + intro JGit cloning (1 jour).
  - Mini-projet : Endpoint `POST /jobs` qui lance une tâche asynchrone fictive (simule repos Git, retourne un ID).
  - Temps : **3 jours**.
- **Analytics (Membre 4) :** Scikit-learn intro (2 jours) + FastAPI POST endpoint (1 jour) + LINQ basics pour .NET (1 jour).
  - Mini-projet : Entraîner un modèle de classification simple sur un dataset fictif (ex: prédire succès/échec étudiant). Exposer endpoint `/predict` en FastAPI.
  - Temps : **4 jours**.

Travail parallèle (Conception) :

- **Tous :** Finaliser OpenAPI specs et tester les contrats avec Postman/Insomnia (mock calls).
- **Backend Core :** Optimiser schéma BD (indexation, vues pour agrégations).
- **Orchestrator :** Concevoir l'architecture interne (files de jobs, événements).

Livrables fin Semaine 3 :

- Frontend : Écran fonctionnel (même avec données mockées) qui appelle l'API Backend.
- Backend Core : CRUD endpoints réels sur Oracle.
- Orchestrator : Async job endpoint opérationnel.
- Analytics : Modèle entraîné + endpoint `/predict` live.
- **Important :** Tous les services documentés via OpenAPI/Swagger.
- Conception : Schémas BD finalisés, diagrammes UML complets, architecture décidée.

—

2.3.4 Semaine 4 : Mini-PoC + Validation

Objectif : Prouver que 2 services majeurs peuvent se parler. Valider l'architecture. Préparer le démarrage officiel du développement Phase 1.

PoC Réaliste (pas la full app) :

Scénario simple : Frontend affiche une liste d'étudiants (GET) et peut en créer un (POST), données du Backend Core.

Tâches :

- **Frontend ↔ Backend Core :**
  - Frontend appelle réellement `GET /students` et `POST /students` du Backend.
  - Affiche la liste avec Tailwind styling + wireframe login (JWT token passing).
  - Temps : Frontend **1-2 jours**, Backend **1 jour** d'intégration finale.
- **Analytics PoC (optionnel mais appréciable) :**
  - Backend Core expose `GET /analytics/class/{id}/stats` (agrégation via vues Oracle).
  - Analytics simule un appel : données brutes → modèle scikit-learn → retour prédiction simple.
  - Temps : **1-2 jours**.
- **Orchestrator readiness (pas de PoC live, mais architecture validée) :**
  - Démontrer la structure de tâche asynchrone (même si vide).
  - Documenter comment Phase 1 va déclencher les jobs (webhooks, polling).
  - Temps : **1 jour** documentation + review.

Validation :

- Tests manuels : Scénario Frontend → Backend fonctionnel end-to-end.
- Swagger/OpenAPI docs générés automatiquement et validés.
- Performance basique (temps de réponse < 500ms pour requêtes simples).
- Logs et error handling en place.

Livrables fin Semaine 4 :

- **PoC fonctionnel :** Frontend parle au Backend, CRUD students opérationnel.

- **Repos clean :** Code organisé, commitable sans hacks.
- **Documentation complète :** Postman collection, OpenAPI YAML, README de chaque service.
- **Checklist Phase 1 :** Schéma BD validé, contrats API finalisés, dépendances listées.
- **Roadmap évolutive :** Planning détaillé des 2-3 premières sprints de Phase 1 (basé sur les 5 phases du projet).

Au-delà de Semaine 4 :

- Todos pour Phase 1 (Semaine 1-4 du mois prochain) : intégration Frontend complète, auth JWT, tests unitaires, CI/CD pipeline.

2.4 Recommandations

- **API Design:** Standardiser les endpoints et produire un fichier `OpenAPI` minimal pour chaque service.
- **Sécurité basique:** JWT + HTTPS, validation côté serveur, gestion des rôles pour les endpoints critiques.
- **SQL & Perf:** Indexation, utilisation de vues pour agrégation côté BD, et usage d'`EXPLAIN`/plans pour requêtes lourdes.

2.5 Planning Prévisionnel Simplifié

La formation se déroule en parallèle de la phase de conception (Mois 1).

- **Semaine 1 :** Socle Commun (Git, Docker) + Début Conception UML.
- **Semaine 2 :** Début des formations spécifiques langage (React, PHP, Java, C#).
- **Semaine 3 :** Approfondissement Bibliothèques Spécialisées (JGit, Pandas, Recharts) + Maquettage.
- **Semaine 4 :** PoC (Proof of Concept) : Connecter 2 services ensemble (ex: React appelle Symfony) + Finalisation Dossier Conception.

Jalons clés :

- Fin Semaine 1 : Environnement validé, tous les repos locaux fonctionnels.
- Fin Semaine 2 : Chaque service expose une API CRUD basique.
- Fin Semaine 3 : Intégration entre-services (Frontend appelle Backend, Analytics reçoit données du Core).
- Fin Semaine 4 : PoC complet fonctionnel, prêt pour la Phase 1 de développement officiel.

3 Spécification des Besoins

Cette section détaille les exigences fonctionnelles et non fonctionnelles du système **Commit Ed**. L'analyse découle de la problématique identifiée : offrir une plateforme d'évaluation alignée sur les workflows professionnels (Dev-First).

3.1 Besoins Fonctionnels

Les besoins fonctionnels sont regroupés par domaine d'application, correspondant à l'architecture distribuée du projet.

3.1.1 Gestion des Utilisateurs et Classes (Service Core)

Ce module constitue le socle administratif de la plateforme.

- **Authentification et Autorisation** : Le système doit permettre une authentification sécurisée (JWT) pour trois rôles distincts : Administrateur, Professeur, et Étudiant.
- **Gestion des Classes** : Le professeur doit pouvoir créer des classes, générer des codes d'invitation uniques et gérer la liste des inscrits.
- **Gestion des Devoirs (Assignments)** : Le professeur doit pouvoir créer un devoir en spécifiant un titre, une description, une date limite et lier un dépôt GitHub « Template ».

3.1.2 Workflow Pédagogique & GitOps (Service Orchestrator)

Ce module est le cœur critique assurant l'immersion « Dev-First ».

- **Distribution de Code** : Lorsqu'un étudiant démarre un devoir, le système doit faciliter la création de son dépôt de travail (via Fork automatique ou Clonage de Template).
- **Détection de Soumission** : Le système doit être capable de détecter quand un étudiant a soumis son travail (via Webhook GitHub ou action manuelle de l'étudiant).
- **Clonage Automatisé** : Le système doit pouvoir récupérer le code de l'étudiant dans un environnement temporaire pour l'analyser.

3.1.3 Analyse et Évaluation (Services Intelligence & Orchestrator)

- **Exécution Sécurisée** : Le code étudiant doit être exécuté dans des environnements isolés (Conteneurs Docker) pour ne pas affecter le serveur principal.
- **Analyse de Code** : Le système doit exécuter des tests unitaires ou des outils de qualité (linter) sur le code soumis.
- **Rapport de Feedback** : Un rapport simple doit être généré après l'analyse, indiquant si les tests sont passés ou échoués.
- **Fallback Manuel** : Le professeur doit pouvoir consulter la note automatique et la modifier manuellement si nécessaire.

3.1.4 Suivi et Intelligence (Service Analytics)

- **Tableaux de Bord (Dashboard)** : Visualisation des statistiques clés : nombre de soumissions par devoir et moyenne de la classe.
- **Alertes Basiques** : Identification des étudiants n'ayant rien soumis à 24h de la date limite.
- **Analyse de Difficulté** : Le système doit pouvoir identifier les devoirs ayant un taux d'échec anormalement élevé.

3.2 Besoins Non-Fonctionnels

Ces exigences définissent les critères de qualité technique réalistes pour la réussite du projet.

3.2.1 Sécurité et Configuration

- **Isolation Basique** : Les conteneurs exécutant le code étudiant ne doivent pas avoir d'accès privilégié au système hôte (pas de montage de volume racine).
- **Sécurité des Accès** : Les mots de passe utilisateurs doivent être hachés et les tokens API ne doivent pas être exposés dans le code source.
- **Configuration Externe** : Les paramètres sensibles (URLs de base de données, Clés API) doivent être chargés via des variables d'environnement (`.env`) et non écrits en dur.

3.2.2 Performance et Fluidité

- **Traitement en Arrière-plan** : L'analyse de code ne doit pas geler l'interface utilisateur. L'étudiant doit pouvoir continuer à naviguer pendant que son code est analysé.
- **Temps de Chargement** : Les pages principales (Dashboard, Liste des devoirs) doivent se charger rapidement (affichage des données sans latence excessive).

3.2.3 Fiabilité

- **Gestion des Erreurs** : Si l'analyse du code échoue (ex: erreur de compilation), le système doit le signaler proprement à l'étudiant au lieu de planter ou d'afficher une page blanche.
- **Intégrité des Données** : Les notes et les soumissions ne doivent pas être perdues. L'utilisation d'une base de données relationnelle (Oracle) garantit la persistance.

3.2.4 Maintenabilité et Déploiement

- **Code Propre** : Le code source doit suivre les bonnes pratiques de chaque langage (PSR pour PHP, Conventions Java/C#) pour faciliter la relecture par les autres membres.
- **Déploiement Unifié** : L'application complète doit pouvoir être lancée sur une machine de développement via une seule commande (ex: `docker-compose up`).
- **Documentation API** : Les points d'entrée (Endpoints) utilisés entre le Frontend et le Backend doivent être documentés (même succinctement) pour éviter les erreurs d'intégration.

3.2.5 Ergonomie

- **Feedback Visuel** : L'utilisateur doit toujours savoir ce qui se passe (ex: afficher un « spinner » de chargement pendant une requête).
- **Navigation Intuitive** : Un utilisateur (Prof ou Étudiant) doit pouvoir trouver les actions principales (Soumettre, Corriger) en moins de 3 clics.

4 Analyse Fonctionnelle et Cas d'Utilisation

L'analyse fonctionnelle vise à identifier les interactions entre les utilisateurs (acteurs) et le système **Commit Ed**. Nous avons modélisé ces interactions via le formalisme UML (Unified Modeling Language) pour délimiter précisément le périmètre du projet.

4.1 Identification des Acteurs

Avant de présenter le diagramme global, il est nécessaire de définir les rôles interagissant avec la plateforme. Nous distinguons les acteurs primaires (utilisateurs humains) des acteurs secondaires (systèmes externes).

Acteurs Primaires (Initiateurs) :

- **L'Étudiant** : Cœur de cible du projet. Il cherche à pratiquer, soumettre son code et obtenir un feedback rapide.
- **Le Professeur** : Il orchestre le cours, crée les devoirs, configure les environnements techniques et valide les notes.
- **L'Administrateur** : Il assure la maintenance du système, la gestion des comptes utilisateurs et la surveillance des logs.
- **Le Visiteur** : Représente un utilisateur non authentifié souhaitant s'inscrire.

Acteurs Secondaires (Systèmes Tiers) :

- **GitHub API** : Utilisé pour l'automatisation des flux Git (Fork, Clone, Pull Request).
- **Docker Engine** : Sollicité par le système pour créer les environnements d'exécution isolés (Sandboxing).

4.2 Diagramme des Cas d'Utilisation Global

Le diagramme suivant illustre l'ensemble des fonctionnalités du système, regroupées par modules logiques. Il met en évidence la centralisation des fonctionnalités communes autour de l'acteur abstrait « Utilisateur Authentifié ».

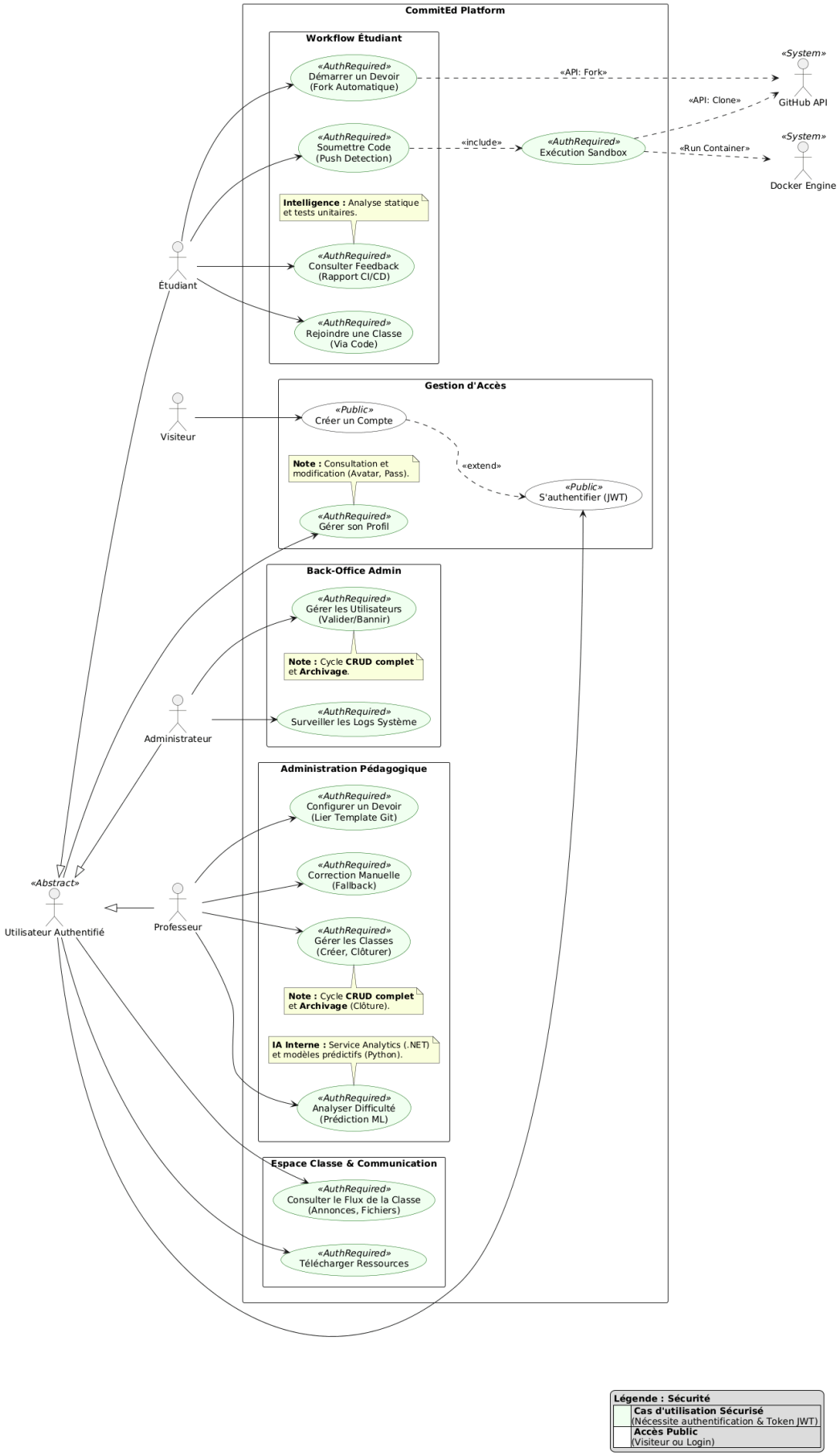

Fig. 2. – Diagramme des Cas d'Utilisation Global de Commit Ed

4.3 Description des Modules Fonctionnels

Le système est découpé en plusieurs zones de responsabilité, comme illustré dans la Fig. 2.

4.3.1 Gestion d'Accès et Socle Commun

Tous les acteurs héritent de fonctionnalités transversales via l'entité **Utilisateur Authentifié**. Cela inclut l'authentification sécurisée (JWT), la gestion du profil et l'accès aux ressources partagées de la classe. Cette factorisation simplifie la gestion des droits.

4.3.2 Administration Pédagogique (Zone Professeur)

Ce module concentre les outils de création de contenu. Le cas d'utilisation « Gérer les Classes » englobe le cycle de vie complet (Création, Modification, Archivage). Le professeur dispose également d'outils avancés comme la prédiction de difficulté, propulsée par notre service interne d'Intelligence Artificielle.

4.3.3 Workflow Étudiant (Zone Critique)

C'est le cœur de la valeur ajoutée « Dev-First ». Contrairement aux LMS classiques, le workflow étudiant est étroitement lié aux outils professionnels :

- **Démarrer un Devoir** déclenche automatiquement un « Fork » sur GitHub.
- **Soumettre Code** n'est pas un simple upload, mais une détection de « Push » Git qui active le moteur d'intégration continue.

4.3.4 Moteur d'Exécution et Systèmes Externes

Les cas d'utilisation techniques, tels que l'exécution en Sandbox, sont isolés. Ils agissent comme une interface entre la logique métier et les acteurs secondaires (Docker, GitHub), garantissant que la complexité technique est masquée pour l'utilisateur final.

5 Annexe : Journal de Suivi (Logbook)

5.1 Séance 1 : Validation du Sujet

Date : 01 Décembre 2025, 13:00
Présents :

- Mr. Imad HOHIB (Encadrant),
- EL AISSAOUI Iliass (Étudiant)
- SGHIOURI IDRISSI Youness (Étudiant)
- TALEB Fayza (Étudiante)

Durée : 20 minutes
Lieu : Salle 17

Remarques de l'Encadrant :

Décisions prises :
Sujet validé par l'encadrant.

Prochaines étapes :

- **tache 1** : Doc qui contient les formations des techno avec durée estimé (2j max)
- **tache 2** : use case
- **tache 3** : Besoins fonctionnels et nn fonctionnels