

Projet Architecture et CL

Titre du dossier :

Construction d'une plateforme de formation en ligne – Sujet 6

Matière : Architecture et Conception Logiciel

Niveau : Master 2

Spécialité ou Parcours : IDO

A l'attention de BURON William

Session : Janvier 2025

GRIMAUD Carla & BER William

Université Catholique de l'Ouest

Institut des Mathématiques Appliquées

Année Universitaire : 2024-2025



CHARTRE DE NON PLAGIAT

Protection de la propriété intellectuelle

Tout travail universitaire doit être réalisé dans le respect intégral de la propriété intellectuelle d'autrui. Pour tout travail personnel, ou collectif, pour lequel le candidat est autorisé à utiliser des documents (textes, images, musiques, films etc.), celui-ci devra très précisément signaler le crédit (référence complète du texte cité, de l'image ou de la bande-son utilisés, sources internet incluses) à la fois dans le corps du texte et dans la bibliographie. Il est précisé que l'UCO dispose d'un logiciel anti-plagiat dans lms.uco.fr, aussi est-il demandé à tout étudiant de remettre à ses enseignants un double de ses travaux lourds sur support informatique.

Cf. « Prévention des fraudes à l'attention des étudiants »

Nous soussignés, Grimaud Carla et Ber William, étudiants en Master 2 MIASHS nous engageons à respecter cette chartre.

Fait à Angers le 10/01/2025

Signature (pour la version imprimée) :

Sommaire

Introduction	II
1- Expression du besoin	III
2- Conception et Caractéristiques de l'API	VI
3- Conception et Organisation de la WebApp.....	X
4- Conception et Implémentation du Batch	XIII
5- Utilisation de Docker.....	XV
6- Résultats du projet - Visuels.....	XVII
7- Organisation du travail – Perspectives	XXI
Conclusion	XXIII

Introduction

Le projet « **Portail de formation et suivi des apprenants** » a pour objectif de concevoir une plateforme en ligne intuitive et performante dédiée à la gestion et au suivi des formations. Cette solution permet aux apprenants de consulter leur planning, réserver des cours et suivre leur progression, tout en offrant aux formateurs un tableau de bord pour gérer les cours et analyser les performances des étudiants.

Les fonctionnalités clés du projet :

- **Pour les apprenants** : inscription, recherche de cours avec filtres avancés, consultation de l'historique et du planning des cours à venir.
- **Pour les formateurs** : création et gestion de cours avec contraintes (horaires, nombre maximum d'élèves), accès aux statistiques et suivi détaillé des progrès des étudiants.
- **Traitement automatisé** : génération hebdomadaire de récapitulatifs individuels pour les apprenants, consolidant les cours programmés la semaine suivante.

Le projet repose sur une **architecture logicielle robuste**, comprenant une API REST et une interface utilisateur dynamique, ainsi qu'une gestion des données via une base PostgreSQL. La containerisation avec Docker Compose assurera une portabilité optimale pour le déploiement.

Ce projet mettra en pratique les concepts fondamentaux de conception logicielle tels que l'**architecture N-tiers**, le **patron MVC** et la **gestion des contraintes métiers**, garantissant un système évolutif et performant.

1- Expression du besoin

Le projet « **Portail de formation et suivi des apprenants** » répond à la nécessité de concevoir une plateforme numérique qui facilite l'accès aux formations tout en optimisant la gestion et le suivi des cours pour les formateurs et les apprenants. Il vise à offrir une solution centralisée et ergonomique intégrant des fonctionnalités avancées pour répondre aux besoins.

1.1 Besoins des utilisateurs

Pour les étudiants :

- **Inscription et connexion** : Création d'un compte utilisateur via une interface intuitive, avec génération automatique d'un identifiant unique (pas de mot de passe, connexion via un identifiant seulement).
- **Consultation des cours** : Accès à un planning détaillé des cours à venir, organisé selon leurs niveaux (L1, L2, L3).
- **Historique et suivi** : Visualisation de la progression à travers un historique de cours et des graphiques pour suivre leur évolution.
- **Recherche avancée** : Recherche d'un cours spécifique à l'aide de filtres par matière, date et date. Les résultats afficheront uniquement les cours disponibles.

Pour les formateurs :

- **Création et gestion des cours** : Possibilité d'ajouter des cours selon des contraintes de disponibilité (une session matin et une session après-midi seulement pour une même journée).
- **Suivi des apprenants** : Accès à un tableau de bord affichant les statistiques des élèves, leur progression et la liste des participants inscrits à un cours spécifique.
- **Visualisation du planning** : Vue d'ensemble des cours à venir pour une journée, une semaine ou un mois donné.

1.2 Besoins techniques et fonctionnels

Batch automatisé : Génération hebdomadaire (le vendredi soir à 17h) de fichiers individuels récapitulant les cours planifiés pour la semaine suivante pour chaque apprenant. Ces fichiers doivent être facilement exportables (non-besoin d'envoi d'un mail à l'étudiant dans le cadre du projet).

Gestion des contraintes : Limitation du nombre d'élèves par cours (selon une capacité maximale définie par le formateur).

- Gestion des conflits d'inscription pour garantir la disponibilité des créneaux horaires.

API REST :

- Gestion des utilisateurs (création, modification, consultation).
- Gestion des cours (CRUD : création, modification, suppression, consultation).
- Accès aux données de progression et de planning des apprenants.

1.3 Objectifs

Accessibilité et ergonomie : Une interface utilisateur intuitive permettant une navigation fluide pour tous les profils d'utilisateurs.

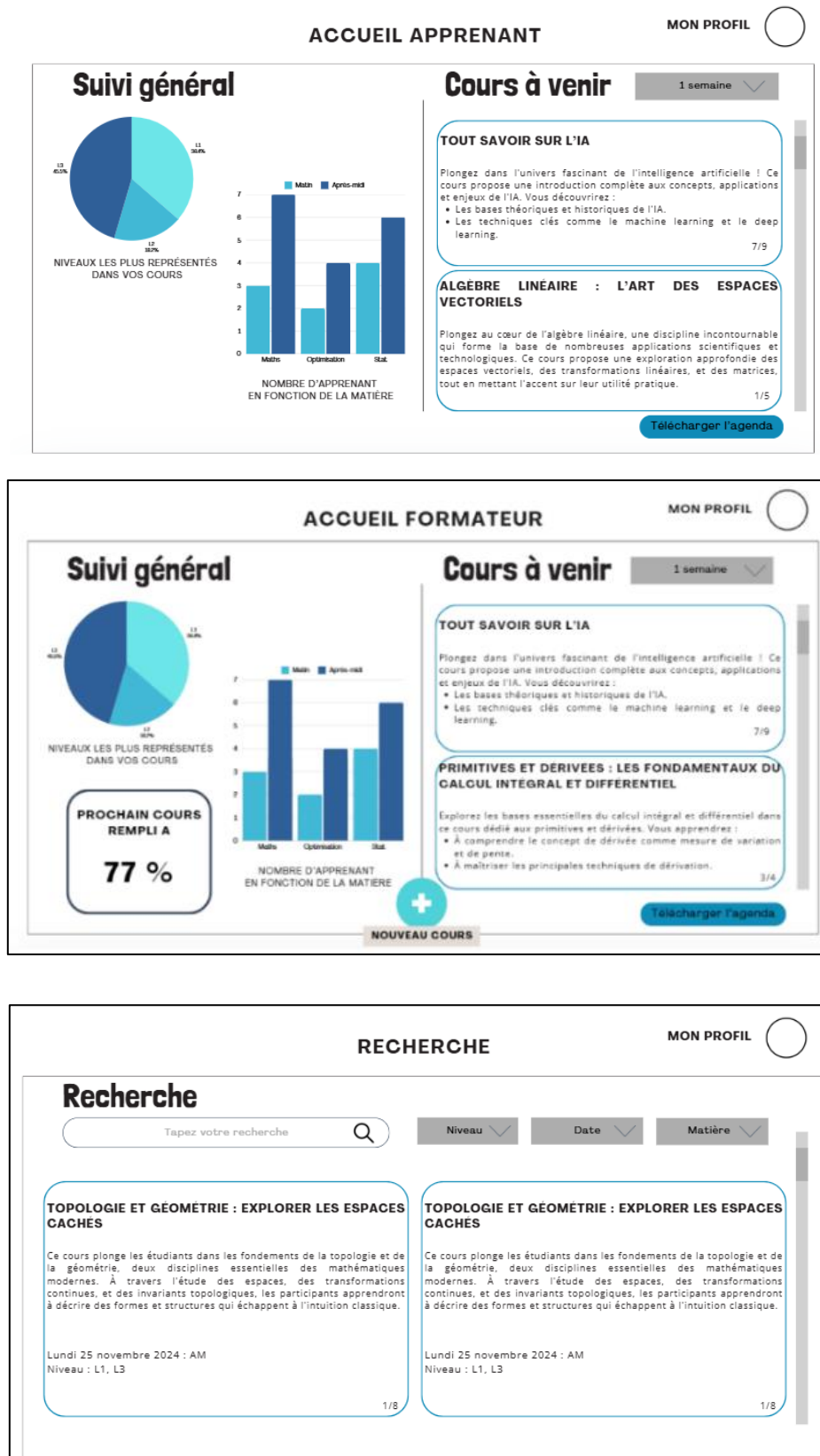
Automatisation : Réduction des tâches manuelles grâce à des traitements automatisés pour la gestion des cours.

Scalabilité et robustesse : Une architecture capable de gérer l'évolution des besoins en termes de fonctionnalités et de volume d'utilisateurs.

Portabilité : Intégration de l'environnement avec Docker Compose pour simplifier les déploiements.

1.4 Maquette du site

Nous avons réalisé une maquette du site avec les 3 pages principales. Cette maquette permet de visualiser les attendues et l'organisation des éléments de notre site :



2- Conception et Caractéristiques de l'API

L'API a été construite selon des principes d'architecture logicielle afin d'assurer modularité, évolutivité et maintenabilité. En s'appuyant sur des concepts tels que la **séparation des préoccupations** et le **patron d'architecture MVC (Modèle-Vue-Contrôleur)**, l'API est structurée pour répondre aux besoins des utilisateurs tout en garantissant une intégration facile avec d'autres systèmes.

De plus, l'API est conçue en **couches**, permettant une meilleure organisation du code et une indépendance entre les différentes parties de l'application. Cette structure favorise la scalabilité et la testabilité en isolant les responsabilités

2.1 Objectifs et principes de conception

Modularité et cohérence :

L'API a été conçue avec une **cohésion forte** et un **couplage faible** entre ses composants. Chaque module, correspondant à une entité (Étudiant, Formateur, Cours, Réservation), est autonome et peut être développé, testé et déployé indépendamment. Cette modularité facilite l'ajout de nouvelles fonctionnalités ou entités dans le futur.

Communication standardisée :

L'API repose sur des **connecteurs RESTful**, une norme bien adaptée pour la communication entre systèmes. Elle utilise les verbes HTTP standard (GET, POST, PUT, DELETE) pour effectuer des opérations sur les ressources, garantissant une compatibilité avec divers clients et plateformes.

Architecture de l'API :

En adoptant le patron d'architecture **MVC**, l'API sépare clairement les responsabilités :

- **Modèle** : Définit la structure des données stockées en base et leur persistance via JPA/Hibernate.
- **Vue** : Gérée via Thymeleaf pour fournir une interface utilisateur dynamique.
- **Contrôleur** : Sert de pont entre la vue et le modèle, reçoit des requêtes, envoie des réponses JSON et interagit avec le service pour récupérer ou modifier les données.

L'architecture de notre projet suit une **Architecture en couche** qui organise le code de manière claire et efficace, cette architecture divise l'application en couches fonctionnelles distinctes,

chacune ayant une responsabilité spécifique. Les couches communiquent de manière linéaire.

Au sommet, nous avons les contrôleurs (controllers) qui gèrent les requêtes des utilisateurs et servent de point d'entrée de l'application. Ces contrôleurs communiquent avec la couche service, qui contient toute la logique métier de l'application. La couche service interagit à son tour avec la couche repository, responsable de la communication avec la base de données. Au cœur de cette architecture se trouvent les modèles (models) qui représentent les entités de notre application : Cours, Etudiant, Formateur et Reservation. Cette organisation permet une séparation claire des responsabilités où chaque couche joue un rôle spécifique, rendant le code plus maintenable et évolutif.



2.2 Caractéristiques de l'API

Construction des 4 models de l'API

- **Étudiant** : Contient des informations comme id, nom, prénom, et le niveaux associé (L1, L2, L3).
- **Formateur** : Similaire à l'étudiant, mais destiné à la gestion des enseignants, il peut se voir attribuer plusieurs niveaux d'enseignement (L1 et/ou L2 et/ou L3).
- **Cours** : Contient les informations comme, id du cours, la matière, le titre, la description détaillée, la date, après-midi ou matin, id du formateur et le nombre d'élèves maximum acceptés pour ce cours.
- **Réservation** : Relie les étudiants aux cours pour gérer les inscriptions, elle contient id cours et id élève.

Base de données relationnelle

Les entités sont mappées sur une base de données relationnelle grâce à **Spring Data JPA**, en utilisant des **transactions ACID** pour garantir la cohérence des données.

Endpoints REST

Chaque entité dispose d'un ensemble d'endpoints REST, permettant de gérer les différentes opérations CRUD (Create, Read, Update, Delete) pour chaque ressource. Par exemple, pour l'entité **Étudiant**, les endpoints suivants sont définis :

- **POST /api/etudiants** : Cet endpoint permet d'ajouter un nouvel étudiant dans la base de données. Un objet Etudiant est envoyé dans le corps de la requête, et l'étudiant est créé via la méthode `addEtudiant` du service `EtudiantService`. Une réponse avec un statut HTTP 201 (CREATED) est renvoyée pour indiquer que l'étudiant a été ajouté avec succès.
- **GET /api/etudiants** : Cet endpoint permet de récupérer la liste de tous les étudiants présents dans la base de données. La méthode `getAllEtudiants` du service est appelée pour récupérer tous les étudiants. La réponse est renvoyée avec un statut HTTP 200 (OK) accompagné d'une liste d'étudiants sous forme de JSON.
- **GET /api/etudiants/{id}** : Cet endpoint permet de récupérer un étudiant spécifique basé sur son identifiant (`id`). Si l'étudiant existe, la méthode `getEtudiantById` du service renvoie l'objet étudiant avec un statut HTTP 200. Si l'étudiant n'est pas trouvé, un statut HTTP 404 (NOT FOUND) est retourné.
- **PUT /api/etudiants/{id}** : Cet endpoint permet de mettre à jour les informations d'un étudiant spécifique. L'étudiant est d'abord recherché par son `id`, et s'il est trouvé, ses informations sont mises à jour en fonction des données envoyées dans le corps de la requête. Si la mise à jour est réussie, une réponse avec un statut HTTP 200 (OK) est renvoyée, indiquant que l'étudiant a été mis à jour. Si l'étudiant n'est pas trouvé, une réponse avec un statut HTTP 404 (NOT FOUND) est retournée.
- **DELETE /api/etudiants/{id}** : Cet endpoint permet de supprimer un étudiant spécifique de la base de données. En utilisant l'identifiant de l'étudiant, la méthode `deleteEtudiant` du service supprime l'étudiant. La réponse indique un statut HTTP 200 (OK) pour confirmer que l'étudiant a été supprimé avec succès.

Ces endpoints permettent de gérer de manière simple et efficace les opérations sur les entités. A terme, nous pourrions construire un **Swagger** afin de documenter notre API en rendant lisible son schéma (endpoints, méthodes http, paramètres, types de données).

2.3 Technologies utilisées

Spring Boot

L'API utilise Spring Boot pour simplifier le développement et le déploiement d'applications Java. Il permet une configuration minimale et une gestion automatisée des dépendances grâce à des annotations comme :

`@SpringBootApplication` : pour configurer l'application Spring Boot.

`@EnableFeignClients` : pour activer l'intégration de clients Feign pour la communication entre services.

Spring Data JPA

Spring Data JPA est utilisé pour la gestion des entités et l'accès à la base de données. Grâce aux annotations comme :

`@Entity` : pour définir les entités de la base de données.

`@Repository` : pour marquer les classes de dépôt de données, facilitant les opérations CRUD.

Thymeleaf

Thymeleaf est utilisé pour générer des pages HTML dynamiques. Les annotations comme :

`@Controller` : pour marquer les classes contrôleurs responsables de la gestion des vues.

`@GetMapping`, `@PostMapping` : pour lier les méthodes aux requêtes HTTP, souvent associées à des templates Thymeleaf pour rendre des pages web.

2.4 Avantages et robustesse de l'architecture

Évolutivité et maintenabilité

Grâce à la séparation des préoccupations et aux connecteurs REST, l'API est facilement extensible. Les nouvelles fonctionnalités peuvent être ajoutées sans affecter le fonctionnement existant. De plus, l'architecture en couche permet de modifier ou d'ajouter des couches sans perturber les autres, facilitant ainsi l'évolution et la maintenance du système.

Fiabilité et compatibilité

L'utilisation de transactions ACID et de normes reconnues (REST) assure la fiabilité et facilite l'intégration avec des systèmes tiers. L'architecture en couche permet également une gestion plus facile des erreurs et une meilleure traçabilité des actions grâce à l'isolation des responsabilités.

Intégration DevOps

L'API est compatible avec des outils modernes tels que **Docker**, permettant une **virtualisation légère** pour simplifier le déploiement et garantir la portabilité.

Cette API illustre un équilibre entre simplicité et robustesse, en répondant aux besoins actuels tout en restant adaptable aux évolutions futures. Elle constitue une base solide pour construire des systèmes scalables et performants, tout en intégrant des principes fondamentaux d'architecture logicielle.

3- Conception et Organisation de la WebApp

La WebApp a été conçue pour fournir une interface utilisateur conviviale permettant d'interagir avec les fonctionnalités exposées par l'API. En s'appuyant sur une architecture **MVC (Modèle-Vue-Contrôleur)**, la WebApp favorise la séparation des préoccupations, la modularité et la maintenabilité. Grâce à l'intégration de technologies modernes telles que **Spring Boot**, **Thymeleaf**, et **Feign**, elle garantit une interaction fluide avec l'API tout en offrant une expérience utilisateur optimisée.

3.1 Structure de la WebApp

Les modèles :

Les modèles de la WebApp représentent les entités manipulées : **Étudiant**, **Formateur**, **Cours**, et **Réservation**. Chaque modèle est conçu pour refléter les données manipulées par l'API et permet de structurer les échanges entre la WebApp et l'API. Ces entités incluent les attributs nécessaires à leurs opérations respectives, comme les informations d'identification, les relations (par exemple, les cours liés aux réservations), et les caractéristiques spécifiques (niveaux pour les étudiants et formateurs).

Les contrôleurs :

Chaque entité possède un **contrôleur dédié** qui centralise les actions utilisateur, telles que l'ajout, la modification, ou la suppression de données. Ces contrôleurs orchestrent les interactions entre la couche Vue (HTML) et l'API, garantissant une séparation claire des responsabilités. Par exemple, le contrôleur des étudiants (**ÉtudiantController**)

récupère la liste des étudiants via le client Feign et transmet ces données à la vue pour affichage.

Clients Feign :

Les **clients Feign** simplifient la communication REST entre la WebApp et l'API. Chaque client est une interface dédiée à une entité, encapsulant les appels HTTP nécessaires pour interagir avec les endpoints REST de l'API. Par exemple, `EtudiantClient` permet de récupérer la liste des étudiants (GET `/api/etudiants`) ou d'ajouter un nouvel étudiant (POST `/api/etudiants`).

Les pages HTML :

La partie Vue de la WebApp repose sur **Thymeleaf**, un moteur de template qui permet d'intégrer dynamiquement les données dans les pages HTML. Les fichiers HTML, placés dans `src/main/resources/templates`, gèrent des fonctionnalités spécifiques comme l'inscription, la connexion, et la gestion des cours. Chaque page est liée à un contrôleur et affiche les données récupérées depuis l'API en temps réel, en utilisant les attributs Thymeleaf comme `th:text` et `th:each`.

3.2 Séparation entre l'API et la WebApp

Modularité et évolutivité

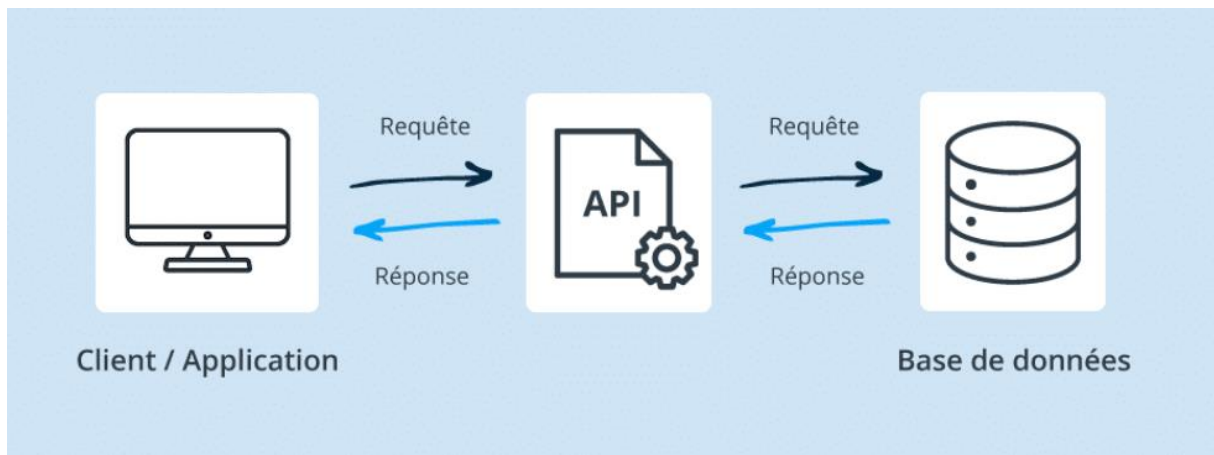
La séparation entre la WebApp et l'API favorise la modularité. De plus, cette séparation permet de faire évoluer la WebApp ou l'API indépendamment, sans impact direct sur l'autre composant.

Réutilisabilité

En isolant l'API, ses fonctionnalités peuvent être réutilisées dans divers contextes, tandis que la WebApp peut intégrer d'autres APIs si nécessaire. Par exemple, la WebApp pourrait consommer des services tiers (authentification, gestion de paiements) sans dépendre exclusivement de l'API locale.

Sécurité

La séparation des préoccupations améliore la sécurité en centralisant la logique métier et les contrôles dans l'API. Les données sensibles ne transitent pas directement dans la WebApp, ce qui limite les risques d'expositions inutiles.



3.3 Points forts de la WebApp

- **Interface utilisateur intuitive** : Grâce à Thymeleaf et Bootstrap, l'interface offre une expérience fluide et moderne.
- **Communication simplifiée avec l'API** : Feign facilite les interactions RESTful, réduisant la complexité des appels réseau.
- **Facilité de maintenance** : La séparation entre modèles, contrôleurs et vues garantit une gestion claire et évolutive des composants.
- **Évolutivité** : La WebApp peut être enrichie avec de nouvelles fonctionnalités ou consommations API tierces sans impact majeur.

En conclusion, la WebApp illustre une architecture bien pensée, où la séparation entre API et interface utilisateur assure modularité, évolutivité, et flexibilité, tout en répondant aux besoins des utilisateurs finaux de manière efficace.

4- Conception et Implémentation du Batch

Le batch est un composant essentiel du système, conçu pour automatiser les tâches répétitives ou complexes liées à la gestion des données. Ces tâches incluent notamment l'importation et l'exportation de données entre la base et des fichiers texte. En s'appuyant sur **Spring Batch**, une solution robuste et modulaire, le système assure une exécution efficace des traitements de données tout en garantissant la traçabilité et la scalabilité.

4.1 Objectifs

Le batch hebdomadaire a pour objectif de générer automatiquement, chaque vendredi soir à 17h, un fichier individuel pour chaque étudiant. Ce fichier contient un récapitulatif des cours planifiés pour la semaine suivante, incluant les détails des matières, des horaires, et des formateurs associés. Ce processus permet aux étudiants d'avoir une visibilité claire sur leur emploi du temps à venir et d'extraire facilement ces informations.

4.2 Fonctionnement et planification

Le batch suit une architecture de traitement par lots, reposant sur le framework **Spring Batch**, et s'articule autour de trois étapes principales :

Lecture des données :

- Extraction des données de la base, notamment les informations sur les étudiants, leurs inscriptions, et les cours prévus.
- Utilisation de `JdbcCursorItemReader` pour récupérer efficacement les données depuis des tables relationnelles.

Traitement des données :

- Filtrage et formatage des informations pour structurer un fichier lisible et organisé pour chaque étudiant.
- Cette étape inclut la validation des données, pour garantir la cohérence et la complétude des informations.

Écriture des fichiers :

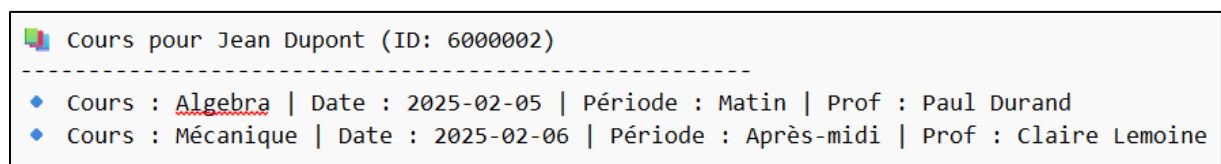
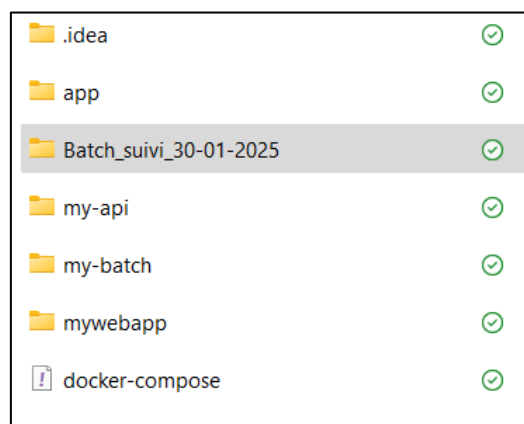
- Génération de fichiers texte individuels contenant les détails des cours, avec des noms de fichiers spécifiques (`recap_nom_prenom.txt`).
- Stockage des fichiers dans un répertoire dédié pour une consultation ultérieure.

Ce batch hebdomadaire illustre une utilisation efficace de **Spring Batch** pour automatiser un processus clé du système. En centralisant les informations pertinentes et en les rendant accessibles aux étudiants, il optimise la gestion des cours tout en réduisant les efforts administratifs. Cette solution s'inscrit dans une architecture logicielle moderne, en alignement avec les concepts de modularité et de scalabilité mentionnés dans les bases de l'architecture.

```
// ✅ Exécution automatique tous les vendredis à 17h pour tous les étudiants
@Scheduled(cron = "0 00 17 * * FRI")   🧑 Ber William *           at 17:00 at Friday day
public void testScheduledExecution() {
    System.out.println("\n🔔 [Planificateur] Lancement du batch pour tous les
    triggerBatchJobForAllStudents();
}
```

Dans notre projet « my-batch », le fichier « BatchConfiguration » permet de paramétrer le moment auquel le batch se déclenche, ici il se déclenche tous les vendredis à 17h.

Lors de son exécution, un dossier avec la date du Batch est créé dans notre « workspace » dans lequel on retrouve chacun de nos étudiants qui ont des cours la semaine prochaine.



Chaque étudiant se voit attribuer un fichier .txt dans lequel il est renseigné ses cours de la semaine prochaine.

5- Utilisation de Docker

5.1 Avantages de Docker

Docker est utilisé dans notre projet pour simplifier le déploiement et assurer la portabilité de l'API ainsi que de la webapp. En encapsulant l'application dans des conteneurs, Docker permet de créer des environnements de développement, de test et de production cohérents, indépendants de la machine hôte. Grâce à Docker, il devient facile de déployer l'application sur n'importe quel serveur, sans avoir à se soucier des différences de configuration entre les environnements. De plus, l'utilisation de Docker Compose nous permet de gérer plusieurs conteneurs simultanément, facilitant ainsi la gestion de la base de données, du backend et du frontend en une seule commande. Cette approche assure une mise en production rapide et fiable, tout en réduisant les risques liés à l'infrastructure et aux configurations spécifiques aux machines. Docker offre ainsi une solution efficace pour la virtualisation légère, la portabilité et la gestion des dépendances dans le projet.

5.2 Utilisation d'un Dockerfile

Le **Dockerfile** est un script qui contient les instructions nécessaires à la création d'une image Docker personnalisée pour notre application. Il définit l'environnement de l'application, y compris le système d'exploitation de base, les dépendances, les configurations et les étapes de construction. Dans notre projet, chaque application (API et webapp) possède son propre Dockerfile qui décrit précisément comment l'environnement de l'application doit être construit, ce qui inclut l'installation de dépendances comme Java ou les packages nécessaires à l'exécution de l'application. Grâce à ce fichier, il est possible de créer des images Docker reproductibles, assurant ainsi que l'application fonctionne de manière identique sur toutes les machines, de la machine locale au serveur de production.

5.3 Principe et lancement du docker-compose

Docker compose permet de définir et de gérer des applications multi-conteneurs à l'aide d'un fichier de configuration YAML (`docker-compose.yml`). Dans ce fichier, nous définissons les services qui composent notre application, comme le serveur backend (`api`), la base de données, et la webapp. Grâce à Docker Compose, il devient possible de démarrer, arrêter et

interagir avec ces services en une seule commande, simplifiant ainsi la gestion de l'infrastructure et de l'intégration des composants. Cela permet de garantir que toutes les parties du système sont configurées de manière cohérente, peu importe l'environnement d'exécution, tout en facilitant le déploiement et les mises à jour de l'application.

NB : nous avons ajouté le fuseau horaire convenable (Europe/Paris) afin que le docker-compose interagisse au mieux avec notre Batch : **TZ: Europe/Paris**

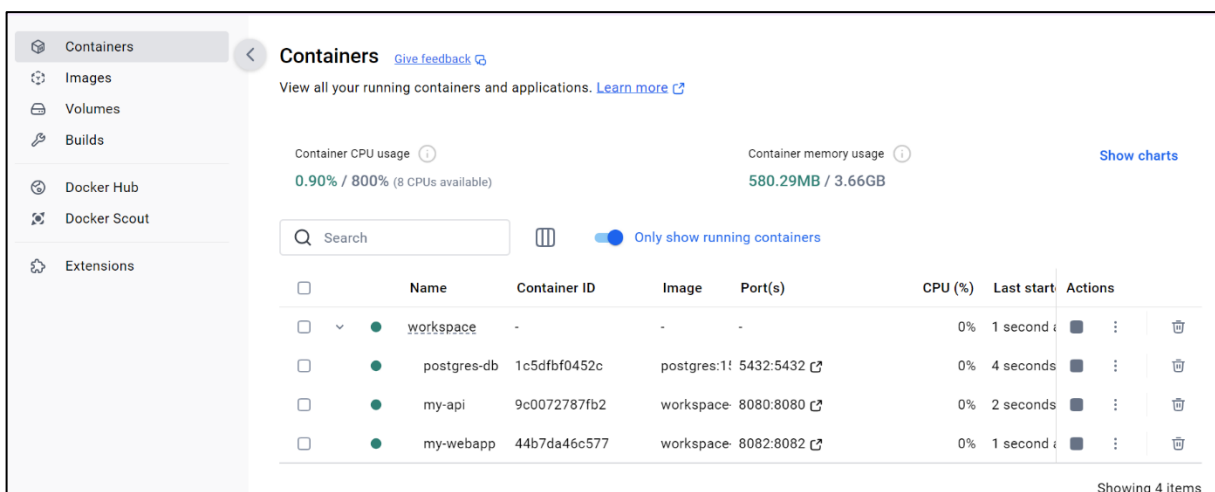
Pour lancer les services définis dans le fichier docker-compose.yml, il suffit d'exécuter la commande suivante dans le terminal à partir du répertoire contenant ce fichier :

docker-compose up

Cette commande va :

1. Lire le fichier docker-compose.yml pour identifier les services à démarrer (comme l'API, la webapp, et la base de données).
2. Vérifier si les images Docker nécessaires sont déjà présentes. Si elles ne le sont pas, Docker Compose va automatiquement construire les images à partir des Dockerfiles associés.
3. Démarrer tous les services dans des conteneurs Docker indépendants, chacun fonctionnant sur son propre environnement isolé, mais interagissant les uns avec les autres selon les configurations définies dans le fichier.

Les conteneurs en cours d'exécution sont visibles sur Docker :



The screenshot shows the Docker Desktop interface. On the left is a sidebar with navigation options: Containers (selected), Images, Volumes, Builds, Docker Hub, Docker Scout, and Extensions. The main area is titled 'Containers' and includes a subtitle 'View all your running containers and applications. Learn more'. It displays system metrics: 'Container CPU usage 0.90% / 800% (8 CPUs available)' and 'Container memory usage 580.29MB / 3.66GB'. A search bar and a toggle for 'Only show running containers' are present. Below is a table of running containers:

	Name	Container ID	Image	Port(s)	CPU (%)	Last start	Actions
<input type="checkbox"/>	workspace	-	-	-	0%	1 second	[Stop] [Refresh] [Delete]
<input type="checkbox"/>	postgres-db	1c5dfbf0452c	postgres:11	5432:5432	0%	4 seconds	[Stop] [Refresh] [Delete]
<input type="checkbox"/>	my-api	9c0072787fb2	workspace	8080:8080	0%	2 seconds	[Stop] [Refresh] [Delete]
<input type="checkbox"/>	my-webapp	44b7da46c577	workspace	8082:8082	0%	1 second	[Stop] [Refresh] [Delete]

Showing 4 items

6- Résultats du projet - Visuels

Dans cette partie, l'objectif est de montrer le rendu visuel du site et les différentes fonctionnalités qu'il renferme.

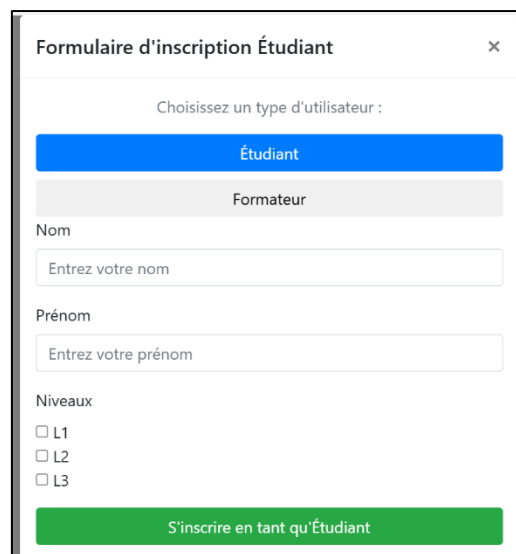
6.1 Page de démarrage – Connexion/Inscription

A partir du moment où on accède à la page de démarrage du site, on a la possibilité de se connecter avec son identifiant (Etudiant ou Formateur). Si l'identifiant n'existe pas dans l'API, un message rouge apparaît « **ID non reconnu ou inexistant.** »



The screenshot shows a login form titled "Connexion". It features a label "Votre ID" above a text input field with the placeholder "Entrez votre ID". Below the input field, a red error message reads "ID non reconnu ou inexistant.". There are two buttons: a blue "Se connecter" button and a green "S'inscrire" button.

Si vous n'avez pas d'identifiant, il suffit de « S'inscrire » ! En premier lieu, on sélectionne dans le formulaire d'inscription si nous sommes un utilisateur « Formateur » ou « Etudiant ». Ensuite, il est nécessaire de rentrer les différents champs disponible et de s'inscrire en tant que Etudiant ou Formateur.



The screenshot shows a modal titled "Formulaire d'inscription Étudiant". It starts with the instruction "Choisissez un type d'utilisateur :". There are two buttons: a blue "Étudiant" button (which is selected) and a grey "Formateur" button. Below these are input fields for "Nom" (placeholder: "Entrez votre nom") and "Prénom" (placeholder: "Entrez votre prénom"). Under the heading "Niveaux", there are three checkboxes: "L1", "L2", and "L3". At the bottom is a green button labeled "S'inscrire en tant qu'Étudiant".

Une fois l'inscription effectuée, nous revenons sur la page pour se connecter et un identifiant est fourni à l'utilisateur. Il suffit donc de rentrer dans son espace avec l'identifiant fourni.7

Connexion

Inscription réussie ! Votre ID de connexion est : 6000001

Votre ID

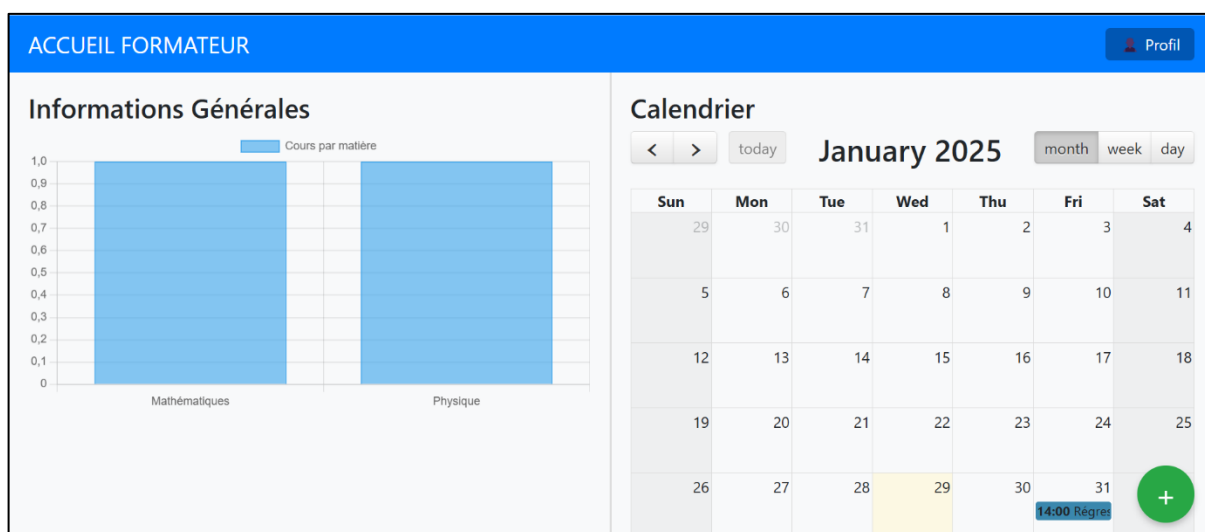
Entrez votre ID

Se connecter

S'inscrire

6.2 Espace Formateur

Si l'utilisateur connecté est un formateur, il arrive sur cette page. La partie gauche renseigne les « Informations Générales » avec quelques statistiques. La partie droite affiche le « Calendrier » du formateur avec l'ensemble des cours, qu'il a créé, qui sont répertoriés dans son agenda, le calendrier permet de voir les cours sur la journée (day), la semaine (week) ou le mois (month). Le bouton « Profil » permet au formateur de consulter ses informations personnelles et le bouton + lui permet de créer un cours auquel les étudiants pourront s'inscrire.



En cliquant sur son profil, on a la possibilité de voir ses informations mais également de se déconnecter avec le bouton en haut à droite « Déconnexion ».

Profil Formateur Déconnexion

Bienvenue, Paul !

ID : 1000001
Prénom : Paul
Nom : Durand
Niveaux Enseignés :

- ☒ L1 : Oui
- ☐ L2 : Oui
- ☐ L3 : Non

[Retour à l'accueil](#)

Par rapport à l'ajout d'un cours, en appuyant sur le bouton +, on peut renseigner les différents champs et ainsi créer un nouveau cours.

ACCUEIL FORMATEUR Profil

Informations Générales

Cours par matière

Mathématiques

Ajouter un Nouveau Cours

Matière
Choisissez une matière

Titre du Cours
Titre du cours

Description
Entrez une description du cours

Date
jj/mm/aaaa

Matin ou Après-midi
Après-midi

Nombre d'Élèves Maximum
Entrez le nombre maximum d'élèves

[Créer le Cours](#) [Fermer](#)

January 2025 month week day

Tue	Wed	Thu	Fri	Sat
31	1	2	3	4
7	8	9	10	11
14	15	16	17	18
21	22	23	24	25
28	29	30	31	1
4	5	6	7	8

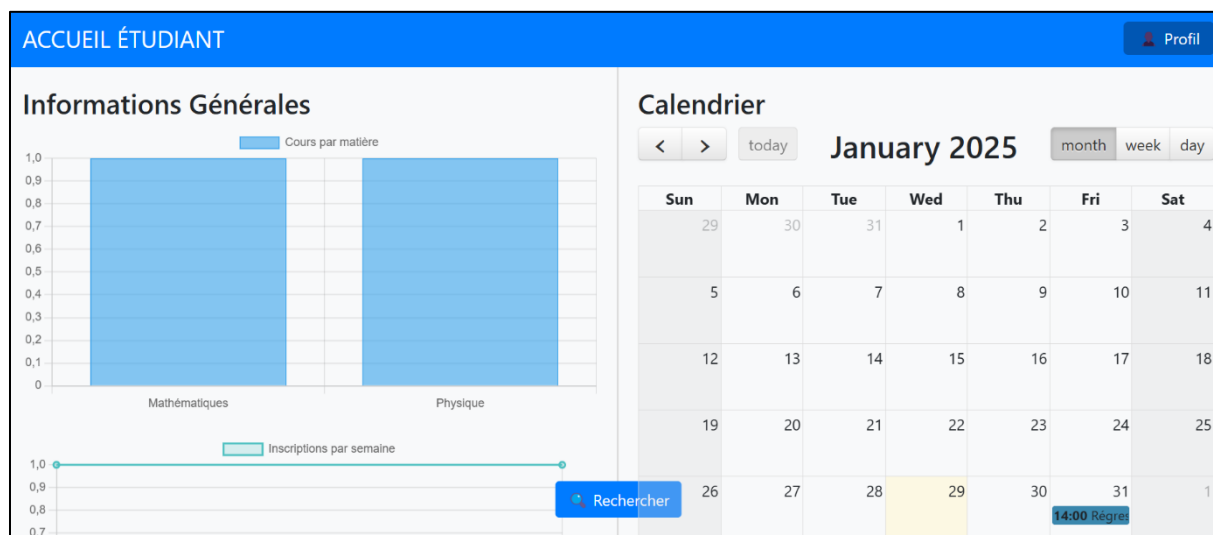
09:00 Algebra

09:00 Ecriture

+

6.3 Espace Etudiant

Comme pour le Formateur, si l'utilisateur se connecte en tant qu'Etudiant, il arrive sur la page « Accueil Etudiant ». De manière identique aux formateurs, l'élève retrouve l'évolution de sa progression à travers plusieurs statistiques ainsi que son emploi du temps avec l'ensemble des cours pour lesquels il est inscrit. L'étudiant, à la différence du formateur, a la possibilité de « Rechercher » un cours de s'y inscrire.



En cliquant sur « Rechercher », on arrive sur cette page. La page est fonctionnelle, elle permet à l'étudiant de filtrer les cours disponibles par « Date » et « Matière » et d'ensuite de « rechercher » le cours. En cliquant sur « S'inscrire », l'étudiant est affecté au cours et en retournant à l'accueil, son nouveau sera inscrit dans son emploi du temps.

Un élève ne peut pas s'inscrire deux fois au même cours, et si le cours est complet alors le cours est noté comme complet dans la recherche.

The screenshot shows the 'RECHERCHE' interface. At the top, there are search filters for 'Date' (with a date input field) and 'Matière' (with a dropdown menu). A 'Rechercher' button is next to the filters, and a 'Retour à l'accueil' button is on the right. The main content area lists two courses: 'Régression linéaire' and 'Tableaux périodique'. Each course entry includes a description, the subject, the date, the time, and a 'S'inscrire' button. The 'Régression linéaire' course also shows 'Places prises : 1 / 5'.

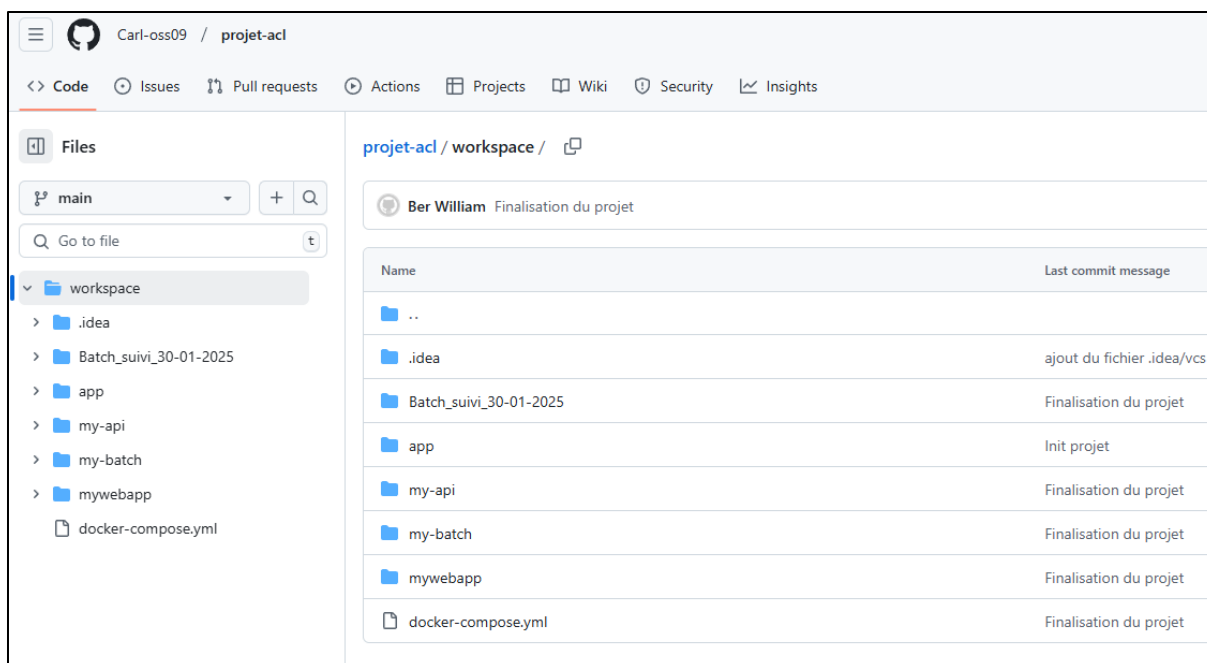
7- Organisation du travail – Perspectives

Dans cette section, nous expliquons comment nous avons organisé notre travail et collaboré tout au long du projet. Nous verrons les outils utilisés, leur impact sur notre efficacité et les améliorations possibles pour de futurs projets.

7.1 Organisation du travail

Pour assurer une collaboration efficace et éviter les conflits de versions, nous avons utilisé **GitHub** comme plateforme de gestion du code source. Cet outil nous a permis de travailler chacun notre tour sur différentes fonctionnalités tout en conservant un historique clair des modifications.

Au départ, nous échangeions notre projet sous forme de fichiers compressés (zip) chaque fois que l'un de nous apportait une modification. La mise en place d'un dépôt GitHub partagé a grandement simplifié notre collaboration, rendant le travail plus fluide et mieux organisé.



De plus, grâce à cette organisation, il est désormais possible de transmettre le projet plus facilement.

7.2 Perspectives d'améliorations

Notre projet est techniquement complet, intégrant la construction et la maintenance de l'API, de la WebApp et du Batch. Nous avons livré une version répondant aux principales exigences du cahier des charges et assurant un bon fonctionnement global du site.

Cependant, certaines fonctionnalités supplémentaires auraient permis d'améliorer l'expérience utilisateur et de rendre le site plus abouti :

- **Sécurité renforcée** : L'ajout d'une base utilisateur avec gestion des mots de passe aurait permis une meilleure sécurisation des comptes.
- **Gestion des cours** : Un formateur devrait pouvoir modifier son cours tant qu'aucun élève n'y est inscrit.
- **Amélioration des graphiques et statistiques** : Un design UX plus soigné rendrait ces éléments plus attractifs et intuitifs. Pour un formateur, il pourrait rentrer un numéro d'étudiant qu'il suit, afin d'analyser ses progressions.
- **Esthétique du site** : Son apparence reste simple ; l'intégration de CSS avancé ou de composants Bootstrap offrirait une interface plus moderne et agréable.
- **Filtrage des cours** : Par manque de temps, nous n'avons pas pu implémenter le filtrage par niveau dans la barre de recherche. Et par la suite, il serait intéressant qu'un étudiant puisse accéder à une barre de recherche, dans laquelle il pourrait rentrer directement le titre d'un cours et le trouver.
- **Modification du profil** : Bien que notre API dispose des fonctionnalités nécessaires, la modification des informations personnelles via le profil utilisateur n'a pas encore été intégrée à l'interface.
- **Interactivité avec le calendrier** : Un étudiant pourrait cliquer sur un cours dans l'emploi du temps et afficher les détails de ce cours. Un formateur pour également le faire et également observer le nombre d'élèves inscrits à son cours.

Ces améliorations pourraient être envisagées dans une future version du projet afin d'optimiser son ergonomie et ses fonctionnalités.

Conclusion

Ce projet de développement d'une **plateforme de formation en ligne** nous a permis d'explorer et de mettre en pratique les **principes fondamentaux de l'architecture logicielle**. Nous avons conçu et implémenté une solution complète incluant une **API REST**, une **WebApp** et un **Batch automatisé**, tout en adoptant une **approche modulaire et évolutive**. L'ensemble repose sur des technologies modernes comme **Spring Boot**, **PostgreSQL**, **Thymeleaf** et **Docker**, garantissant ainsi **portabilité et maintenabilité**.

L'objectif principal n'était pas de livrer un produit totalement abouti ou prêt à être déployé en production, mais avant tout **d'approfondir notre compréhension des architectures logicielles** et des **principes de construction d'une application robuste et scalable**. En ce sens, nous avons pu expérimenter **l'organisation en couches**, **la gestion des bases de données relationnelles**, **la communication entre services REST**, ainsi que **l'automatisation des tâches répétitives via le batch**. Ces compétences sont essentielles pour aborder sereinement des projets à plus grande échelle dans un contexte professionnel.

Malgré la réussite globale du projet, certaines améliorations restent envisageables, notamment en matière **d'expérience utilisateur, de sécurité et d'interactivité**. L'ajout d'une gestion des comptes avec authentification sécurisée, un **design plus avancé avec Bootstrap**, ou encore des **fonctionnalités de filtrage et de personnalisation des cours** renforceraient encore davantage l'efficacité et l'ergonomie du site.

Enfin, **l'utilisation de GitHub** a joué un rôle central dans la gestion de notre collaboration, facilitant le suivi des modifications et la coordination entre nous. Ce projet nous a ainsi permis de développer des compétences techniques mais aussi organisationnelles, indispensables pour la gestion efficace d'un projet informatique en équipe.