# Atelier

# Conception d'une plateforme LMS

**PRÉSENTÉ PAR**

MORAIS PEREIRA David,

ZAOUALI Olfa,

BEN KEMIL Mezen,

TATANG DOUANLA Jopcelin

**CLASSE**

Mastère 1 DEV WEB

**DATE DE REMISE**

11-02-2025

Atelier ..... 1

1. Une cartographie de l’architecture : ..... 3
2. Une justification des choix d’architecture : ..... 4
3. Deux exemples d’implémentation technique : ..... 5
4. Un README explicatif : ..... 9

Résumé de l’Architecture : ..... **Erreur ! Signet non défini.**

# 1. Une cartographie de l'architecture :

## a. Identification des différentes briques et services

| Fonctionnalité | Outils | Services |
|---|---|---|
| Gestion des utilisateurs | - Base de données des utilisateurs<br>- Mécanisme d'authentification sécurisé (OAuth2, JWT) | - Service d'inscription des utilisateurs<br>- Service de connexion (authentification)<br>- Service de gestion des rôles et permissions |
| Gestion des cours | - Base de données des cours<br>- Moteur de gestion des cours (création, édition, organisation) | - Service de création et édition de cours<br>- Service de gestion des modules de cours<br>- Service de partage des documents et ressources pédagogiques |
| Système de correction automatique | - Base de données des évaluations (résultats des examens, soumissions)<br>- Moteur de correction automatique (algorithmique) | - Service de correction d'examens<br>- Service de correction de code pour les soumissions de devoirs |
| Moteur de recommandation | - Base de données des performances d'apprentissage des étudiants<br>- Moteur d'analyse et de recommandation | - Service d'analyse des données d'apprentissage des étudiants<br>- Service de génération de recommandations de cours |
| Forum et messagerie | - Base de données des discussions/messages<br>- Serveur de messagerie en temps réel (WebSockets) | - Service de gestion des forums<br>- Service de messagerie privée<br>- Service de notifications en temps réel |
| Gestion des certificats et badges | - Base de données des certificats et badges obtenus<br>- Moteur de validation des acquis et génération de certificats | - Service de validation des acquis<br>- Service de génération de certificats et attribution de badges |

### b. Distinction entre les éléments en Microkernel et Microservices

| Module | Architecture | Justification |
|---|---|---|
| **Gestion des utilisateurs** | **Microservices** | Permet de gérer indépendamment l'authentification, les rôles et les permissions. Facilite l'intégration avec des services externes (OAuth…). |
| **Gestion des cours** | **Microkernel** | Noyaux du LMS et peut avoir des plugins supplémentaire |
| **Correction automatique** | **Microservices** | Nécessitant une grande scalabilité pour supporter un grand nombre de copie. |
| **Moteur de recommandation** | **Microservices** | Basé sur l'analyse des données d'apprentissage. Doit être évolutif et peut nécessiter de l'IA. |
| **Forum et messagerie** | **Microservices** | Indépendant du LMS, peut être remplacé par un service externe (ex : Slack). |
| **Gestion des certificats et badges** | **Microkernel** | Dépasse la complexité du besoin pas besoin d'en faire un microservice, peut être implémentée en tant que plugin dans le Microkernel de gestion des cours. |

### c. Présentation des API et des bases de données utilisées

| Fonctionnalité | API | Base de données |
|---|---|---|
| Gestion des utilisateurs | **REST API** : Pour la gestion des utilisateurs | **SQL (MySQL)** : Gérer des informations relationnelles sur les utilisateurs (inscription, rôles, etc.) |
| Gestion des cours | **REST API :** Pour ajouter, supprimer, modifier des cours et obtenir des détails | **SQL (MySQL)** : Structuration des cours, modules et leur relation avec les utilisateurs |
| Système de correction | **REST API** : Pour corriger les examens | **SQL (MySQL)** : Stocker les soumissions, évaluations, résultats de correction |
| Moteur de recommandation | **REST API / Kafka** : Recommandations basées sur les performances d'apprentissage | **NoSQL (MongoDB)** : Pour analyser les données non structurées sur les performances des étudiants |
| Forum et messagerie | **WebSockets** : Communication en temps réel | **NoSQL (MongoDB)** : Stocker les messages, discussions et notifications |
| Gestion des certificats | **REST API** : Créer et gérer les certificats | **SQL (MySQL)** : Stocker les certificats et badges des étudiants |

## 2. Une justification des choix d'architecture :

### a. Microkernel vs Microservices : Choix d'Architecture pour les fonctionnalités ?

- **Microkernel :**

**Fonctionnalités :** Gestion des cours, gestion des certificats et badges.
**Justification :** Ces briques sont centrales au système LMS. Le modèle **Microkernel** permet de garder un noyau stable et flexible tout en ajoutant des plugins ou modules supplémentaires sans perturber le système. Cela garantit la stabilité, la cohérence et l'extensibilité des fonctionnalités fondamentales comme la gestion des cours et des certificats.

- **Microservices :**

**Fonctionnalités :** Gestion des utilisateurs, correction automatique, moteur de recommandation, forum et messagerie.
**Justification :** Ces briques nécessitent indépendance, évolutivité et scalabilité. **Microservices** permettent une gestion modulaire et l'évolutivité, où chaque service peut être développé, déployé et mis à l'échelle indépendamment. Cela est essentiel pour des services comme la gestion des utilisateurs, le moteur de recommandation ou la correction automatique, qui doivent être capables de gérer un volume élevé de données et d'utilisateurs.

### b. Bénéfices Évolutivité des Architectures Microkernel et Microservices ?

L'architecture Microservices permet d'assurer une grande évolutivité et une maintenance simplifiée pour des fonctionnalités comme la gestion des utilisateurs, la correction automatique, ou le moteur de recommandation, qui doivent être indépendantes et scalable en fonction du volume d'utilisateurs.

Le Microkernel, quant à lui, permet de garantir une stabilité et une extensibilité pour des fonctionnalités de base comme la gestion des cours et la gestion des certificats, tout en offrant la possibilité d'ajouter des plugins sans perturber le noyau du système.

En résumé, l'utilisation d'une architecture hybride (Microkernel + Microservices) optimise à la fois la maintenabilité et l'évolutivité, permettant de gérer des systèmes complexes avec des exigences diverses tout en garantissant la stabilité du système global.

## 3. Deux exemples d'implémentation technique :

### a. Exposition d'un service sous forme de Microservice (déclaration d'une API REST/gRPC)

```java
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/utilisateurs")
public class UtilisateurController {

    @PostMapping
    public String enregistrerUtilisateur() {
        return "Utilisateur enregistré";
    }

    @PostMapping("/connexion")
    public String connecterUtilisateur() {
        return "Utilisateur connecté";
    }

    @PutMapping("/{id}")
    public String mettreAJourUtilisateur(@PathVariable String id) {
        return "Utilisateur " + id + " mis à jour";
    }

    @DeleteMapping("/{id}")
    public String supprimerUtilisateur(@PathVariable String id) {
        return "Utilisateur " + id + " supprimé";
    }
}
```

### b. Mécanisme d'extension sous forme de Microkernel (exemple de relation entre le noyau et ses plugins)

**Architecture :**

Le Core contient les fonctionnalités principales, comme la gestion des cours (ajout, mise à jour des cours, etc.…).

Le Plugin (gestion des certificats) va interagir avec le Core pour récupérer les informations sur les cours ou les performances des étudiants, puis générer un certificat.

#### Core - Gestion des Cours (Microkernel)

Le Core fournit une API REST pour récupérer les informations des cours et des performances des étudiants. Ce service est indépendant du plugin.

```java
package com.example.microkernelcore.controller;

import com.example.microkernelcore.service.CoreService;
//imports

@RestController
@RequestMapping("/courses")
public class CoreController {

    private final CoreService coreService;

    public CoreController(CoreService coreService) {
        this.coreService = coreService;
    }

    // Endpoint pour récupérer un cours spécifique
    @GetMapping("/{courseId}")
    public String getCourseDetails(@PathVariable String courseId) {
        return coreService.getCourseDetails(courseId);
    }
}
```

```java
package com.example.microkernelcore.service;

import org.springframework.stereotype.Service;

@Service
public class CoreService {

    // Récupère les détails d'un cours en fonction de son ID
    public String getCourseDetails(String courseId) {
        // Logique de récupération des données du cours (exemple simple ici)
        return "Course details for course " + courseId + ": This is a Java programming course.";
    }
}
```

## Plugin - Gestion des Certificats

Le Plugin va se connecter au Core via l'API REST pour récupérer les informations de cours d'un étudiant et générer un certificat.

```java
package com.example.microkernelplugin.controller;

import com.example.microkernelplugin.service.PluginService;
//imports

@RestController
@RequestMapping("/certificates")
public class PluginController {

    private final PluginService pluginService;

    public PluginController(PluginService pluginService) {
        this.pluginService = pluginService;
    }

    // Endpoint pour générer un certificat pour un étudiant
    @PostMapping("/generate/{studentId}/{courseId}")
    public String generateCertificate(@PathVariable String studentId, @PathVariable String courseId) {
        return pluginService.generateCertificate(studentId, courseId);
    }
}
```

```java
package com.example.microkernelplugin.service;

//imports

@Service
public class PluginService {

    private final RestTemplate restTemplate;

    public PluginService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    // Appel à l'API du Core pour obtenir les détails du cours
    public String generateCertificate(String studentId, String courseId) {
        // Appel de l'API REST du Core pour récupérer les détails du cours
        String courseDetailsUrl = "http://localhost:8081/courses/" + courseId;
        String courseDetails = restTemplate.getForObject(courseDetailsUrl, String.class);

        // Générez un certificat en fonction des détails du cours
        return "Certificate generated for student " + studentId + " in course: " + courseDetails;
    }
}
```

## Explication du processus de communication

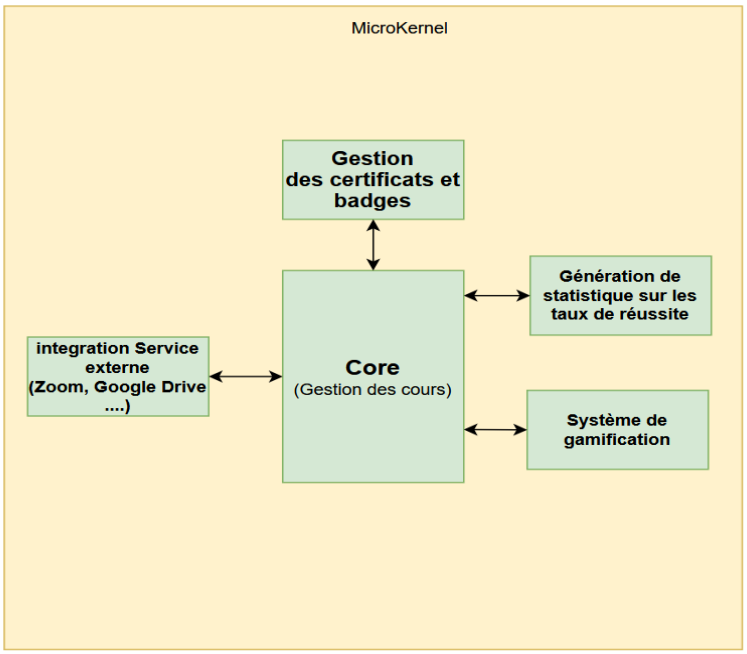
Étape 1 : Lorsqu'un étudiant termine un cours, le plugin de gestion des certificats est appelé pour générer un certificat pour cet étudiant dans un cours spécifique.

Étape 2 : Le plugin (via PluginService) fait une requête HTTP GET à l'API REST du Core pour récupérer les informations du cours associé à l'étudiant. Cela est réalisé à l'aide de RestTemplate, un outil de Spring permettant de faire des appels HTTP facilement. Dans la méthode generateCertificate() du service, RestTemplate envoie la requête à l'API du Core, récupère les données du cours et les retourne sous forme d'objet.

Étape 3 : Le plugin récupère les données du cours via RestTemplate, les analyse, puis génère un certificat pour l'étudiant.

Étape 4 : Le certificat généré est retourné à l'utilisateur via l'API du plugin.

=> Il s'agit ici d'un exemple simplifié, mais le même principe s'applique pour les autres plugins, tels que l'analyse des performances, le système de gamification, et d'autres fonctionnalités. Vous trouverez ci-dessous une présentation graphique illustrant la manière dont ces plugins communiquent avec le Core.




## 4. Un README explicatif :

### a. Les technologies choisies

- **Frameworks :** Spring Boot → l'un des meilleurs Frameworks pour combiner une architecture Microkernel et Microservices, offrant un noyau extensible tout en permettant le développement de services indépendants et scalables.
- **Langages :** Java → choix naturel pour Spring Boot, performant et robuste.
- **Bases de données :**
    - **MySQL** → pour stocker des données relationnelles structurées (utilisateurs, cours, certificats).
    - **MongoDB** → pour des données semi-structurées et volumineuses (messagerie).
- **Middleware :**

- **Kafka** → système de messagerie distribué utilisé pour l'échange de données entre services en temps réel.

- **Orchestration :**
  - **Docker** → facilite le déploiement des services sous forme de conteneurs isolés.

### b. Évolution du système

- **Intégration d'IA pour les recommandations et la correction automatique :**
  - L'analyse des performances des étudiants pourrait être optimisée par **du Machine Learning** pour offrir des recommandations personnalisées de cours.
  - Une IA pourrait aussi **automatiser la correction des examens** et fournir des feedbacks détaillés.
- **Amélioration de l'engagement des utilisateurs avec des assistants IA :**
  - Intégration d'un **chatbot IA** basé sur GPT pour répondre aux questions des étudiants en temps réel.
  - Développement d'un **tuteur virtuel intelligent** capable d'adapter les exercices en fonction du niveau et du rythme d'apprentissage de chaque utilisateur.
- **Expansion du LMS vers des formats d'apprentissage immersifs :**
  - Ajout d'un **module de réalité virtuelle (VR)** permettant des formations interactives dans un environnement immersif.
  - Intégration de la **réalité augmentée (AR)** pour proposer des expériences d'apprentissage plus engageantes (exemple : dissections virtuelles pour les cours de biologie).