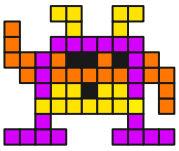


MJEC

Atelier

Conception d'une plateforme MLS

11 février 2025

Groupe MJEC – Vive le travail
Clément VITRAT-GUTIERREZ, Enzo CHAMANIER, Manon LAFOSSE, Julia GROSSI, Dylan CHATELAIN, Tom FREGONESE, Shun LASSAL, Enzo SALVATI

Table des matières

Introduction Atelier

A. Contexte

Vous êtes une équipe d'architectes logiciels en charge de concevoir une plateforme de gestion d'apprentissage en ligne (LMS) destinée aux universités et centres de formation. Cette plateforme doit être modulable, scalable et maintenable pour répondre aux besoins évolutifs des établissements partenaires.
Votre mission est de proposer une architecture intégrant à la fois une logique Microkernel et Microservices, en justifiant vos choix techniques et conceptuels.

B. Fonctionnalités à gérer

Le LMS doit inclure les fonctionnalités suivantes :

- Gestion des utilisateurs : Inscription, connexion, gestion des rôles (étudiant, enseignant, administrateur)
- Gestion des cours : Création, édition, organisation en modules, partage de documents
- Système de correction automatique : Correction d'examens, évaluations de code et soumissions de devoirs
- Moteur de recommandation : Suggestions de cours basées sur l'historique des étudiants
- Forum et messagerie : Communication entre étudiants et enseignants
- Gestion des certificats et badges : Attribution et validation des acquis

C. Problématiques et réflexion

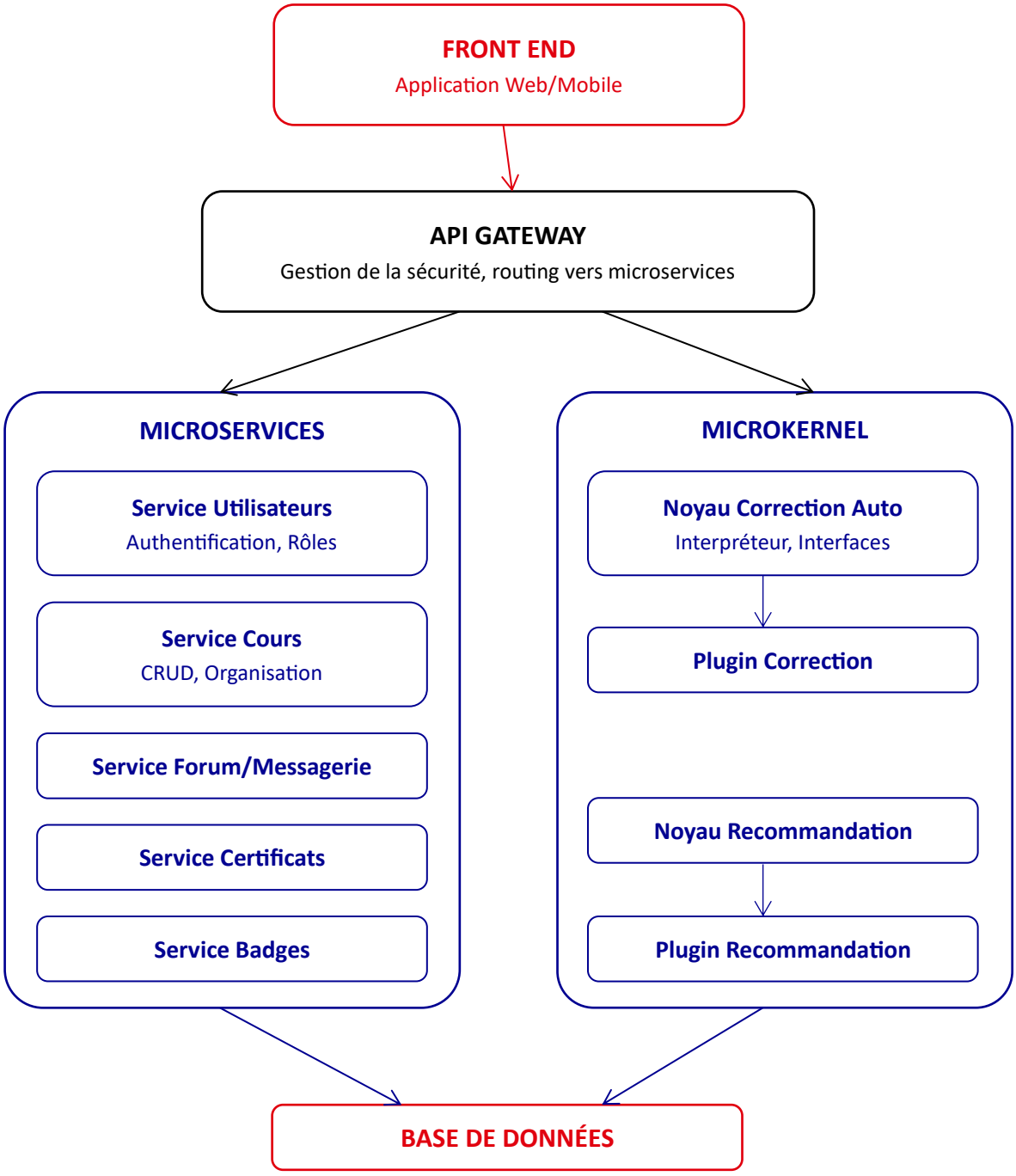
- Quels modules nécessitent une architecture Microkernel pour permettre une extension future sans réécrire le noyau ?
- Quels services doivent être découplés sous forme de Microservices pour garantir leur évolutivité et scalabilité ?
- Quels protocoles/API utiliser pour assurer une communication fluide entre ces architectures ?
- Quels choix de stockage sont pertinents en fonction des besoins de chaque brique ?
- Comment gérer la sécurité et l'observabilité d'un tel système ?

D. Livrables attendus

Chaque groupe devra produire :

1. Une cartographie de l'architecture :
   - Identification des différentes briques et services
   - Distinction entre les éléments en Microkernel et Microservices
   - Présentation des API et des bases de données utilisées
2. Une justification des choix d'architecture :
   - Pourquoi certaines briques ont été implémentées en Microkernel et d'autres en Microservices ?
   - Quels bénéfices cela apporte en termes de maintenance et d'évolutivité ?
3. Deux exemples d'implémentation technique :
   - Exposition d'un service sous forme de Microservices (déclaration d'une API REST/gRPC)
   - Mécanisme d'extension sous forme de Microkernel (exemple de relation entre le noyau et ses plugins)
4. Un README explicatif :
   - Décrire les technologies choisies (langages, bases de données, middleware, orchestration)
   - Expliquer comment le système pourrait évoluer à l'avenir

1. Cartographie de l'architecture



A. Microservices principaux

- Service Utilisateurs :
  - → Gère l'authentification
  - → Gère l'autorisation
  - → Gère les rôles (étudiant, enseignant, admin)
- Service Cours :
  - → Gère le cycle de vie des cours (création, édition, organisation, documents)
- Service Forum/Messagerie :
  - → Gère la communication (posts, fils de discussions, messageries)
- Service Certificats :
  - → Gère l'émission
  - → Gère la validation des certificats
- Service Badges :
  - → Gère l'attribution de badges en fonction des accomplissements

B. Microkernel

- Système de correction automatique :
  - → Définit les interfaces et le cœur de la logique
  - → Plug-ins de correction
- Moteur de recommandation :
  - → Noyau qui définit des interfaces pour la recommandation (historique, scoring)

C. API Gateway

- Point d'entrée pour tous les clients

D. Bases de Données

- Base de données relationnelles (PostgreSQL/MySQL) :
  - → Gestion des utilisateurs
  - → Gestion des cours
  - → Gestion des certificats
- Base de données NoSQL (MongoDB) :
  - → Stocker les posts de forum
  - → Stocker la messagerie
  - → Stocker les logs ou historiques d'apprentissage pour la recommandation
- Object Stockage :
  - → Pour le partage et le stockage de documents volumineux

2. Justification des choix d'architectures

A. Pourquoi utiliser le Microkernel ?

- Système de correction automatique :
  → Possibilité d'ajouter de nouvelles fonctionnalités
  → Les plugins suivent la même interface imposée par le noyau
  → Les changements d'implémentation sont isolés dans le plugin

- Moteur de recommandation :
  → Les algorithmes de recommandation évoluent souvent
  → On isole le noyau des algorithmes spécifiques
  → Facilite l'évolution de la plateforme

B. Pourquoi utiliser des Microservices ?

- Scalabilité :
  → Fonctionnalité comme la gestion des utilisateurs
  → Fonctionnalité comme la gestion des cours
  → Fonctionnalité comme la gestion des forums

- Évolutivité :
  → Chaque service évolue à son propre rythme
  → Possibilité de changer la stack technologique d'un microservices sans impacter les autres

- Déploiement indépendant :
  → Possibilité de déployer un nouveau correctif ou fonctionnalité sans redéployer l'ensemble

- Isolation des pannes :
  → Un problème dans le Service Forum n'empêche pas l'accès au Service Cours
  → Un problème dans le Service Forum n'empêche pas l'accès au Service de Correction

C. Communication Inter Services

- API Gateway :
  → Reçoit les requêtes externes
  → Distribue aux microservices correspondants

- Communication interne :
  → REST : simplicité et popularité

D. Choix de Stockage

- Gestion des Utilisateurs / Rôles :
  → Base relationnelle (PostgreSQL)

- Contenu des Cours :
  → Base de données relationnelle (PostgreSQL) pour les métadonnées (titres, chapitres…)
  → Base de données Object Storage pour les documents

- Forum / Messagerie
  → Base de données NoSQL (MongoDB)

- Logs / Historique / Traçabilité :
  → Base de données NoSQL (MongoDB)

E. Sécurité et Observabilité

- Sécurité :
  → Authentification gérée via le Service Utilisateur
  → API Gateway : validation de token
  → Communication HTTPS/TLS pour interfaces

- Observabilité :
  → Centralisation des Logs
  → Monitoring
  → Traçage distribué : trajectoire d'une requête à travers différents microservices et le Microkernel

3. Exemples d'implémentation technique

A. Exposition d'un service sous forme de Microservices

Déclaration d'une API REST pour le Service Utilisateurs

```yaml
openapi: 3.0.0
info:
  title: Service Utilisateurs
  version: 1.0.0
paths:
  /users:
    get:
      summary: Liste les utilisateurs
      responses:
        200:
          description: Liste d'utilisateurs
    post:
      summary: Crée un nouvel utilisateur
      requestBody:
        description: Données de l'utilisateur
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/User'
      responses:
        201:
          description: Utilisateur créé
  /users/{id}:
    get:
      summary: Retourne un utilisateur par ID
      parameters:
        - in: path
          name: id
          required: true
          schema:
            type: string
      responses:
        200:
          description: Détails de l'utilisateur
    put:
      summary: Met à jour un utilisateur
      ...
components:
  schemas:
    User:
      type: object
      properties:
        id:
          type: string
        email:
          type: string
        role:
          type: string
        ...
```

B. Mécanisme d'extension sous forme de Microkernel

Relation entre le noyau de Correction automatique et ses plugins

```java
// Interface commune dans le noyau
public interface CorrectionPlugin {
    // Méthode qui retourne le langage ou le type de soumission supporté
    String getSupportedLanguage();

    // Méthode de correction qui reçoit la soumission (ex. code, réponses)
    CorrectionResult correctSubmission(Submission submission);
}

// Implémentation d'un plugin pour le langage Python
public class PythonCorrectionPlugin implements CorrectionPlugin {
    @Override
    public String getSupportedLanguage() {
        return "python";
    }

    @Override
    public CorrectionResult correctSubmission(Submission submission) {
        // Logique spécifique à Python
        // On peut exécuter un conteneur Docker, lancer un script d'analyse, etc.
        return new CorrectionResult(...);
    }
}

// Noyau qui charge dynamiquement les plugins
public class AutoCorrectionKernel {
    private List<CorrectionPlugin> plugins;

    public AutoCorrectionKernel() {
        this.plugins = new ArrayList<>();
    }

    public void registerPlugin(CorrectionPlugin plugin) {
        plugins.add(plugin);
    }

    public CorrectionResult correct(Submission submission) {
        for (CorrectionPlugin plugin : plugins) {
           if (plugin.getSupportedLanguage().equalsIgnoreCase(submission.getLanguage()))
            {
                return plugin.correctSubmission(submission);
            }
        }
        throw new UnsupportedOperationException("Aucun plugin pour ce langage");
    }
}
```

Fonctionnement :

1. Le noyau définit l'interface (CorrectionPlugin).
2. Chaque plugin implémente la logique de correction.
3. Le noyau peut changer les changer les plugins au démarrage ou dynamiquement.

4. README Explicatif

A. Technologie choisies

Langages

- Backend (Microservices) : Java (Spring Boot) pour un socle robuste
- Microkernel : Java pour bénéficier de mécanismes de chargement dynamique

Bases de Données

- Relationnelle (PostgreSQL) pour les données structurées
- NoSQL (MongoDB) pour les données volumineuses
- Object Storage pour les contenus multimédias

Communication

- API Gateway : Nginx
- API Microservices : REST

Sécurité

- JWT pour l'authentification stateless
- OAuth2 pour l'intégration
- HTTPS/TLS pour toutes les connexions

Orchestration et déploiement

- Conteneurisation avec Docker
- Orchestrateur Kubernetes pour le déploiement scalable

Observabilité

- Stack Prometheus/Grafana pour le monitoring
- ELK pour la centralisation des logs

B. Evolution future du système

- Nouveaux plugins de correction :
  → Ajout de nouvelles langues
  → Ajout de types de projets

- Nouveaux algorithmes de recommandation :
  → Création ou mise à jour d'algorithmes de ML

- Microservices additionnels :
  → Service d'Analytics : tableau de bord pour les enseignants, suivi d'engagement…
  → Service de Gamification : plugins de badges supplémentaires, points…

- Extension à d'autres domaines :
  → Incorporation de la réalité virtuelle/augmentée
  → Gestion des parcours adaptatifs selon le profil de l'étudiant