Université Nord Américaine privée

Institut International Technologie

Architectures logicielles évoluées : Framework Spring – Mini projet

s1-2025-2026

Sujet proposé : Application de gestion d'un centre de formation

Contexte général

Une école d'ingénieurs souhaite mettre en place une application web permettant :

- aux **administrateurs** de gérer les étudiants, les formateurs, les cours et les inscriptions ;
- aux **étudiants** de consulter leurs cours, leurs notes et de s'inscrire en ligne ;
- aux **formateurs** de gérer les cours qu'ils dispensent et de saisir les notes.

L'application doit être pensée de manière **modulaire et évolutive**, en exploitant les **principes de l'architecture Spring Boot** et les **bonnes pratiques de conception orientée objet**.

Objectifs pédagogiques

1. **Architecture logicielle :**
   - Organisation du projet en couches (controller + service + repository + entity).
   - Gestion du cycle de vie du projet avec **Maven** (build, test, packaging, dépendances).
   - Injection de dépendances via **Spring IoC Container**.
   - Gestion des beans et des services métier.

2. **Persistance des données :**
   - Utilisation de **Spring Data JPA** avec **Hibernate**.
   - Modélisation UML (diagramme de cas d'utilisation , diagramme de déploiement, diagramme de classes, diagramme de séquences).
   - Utiliser **ORM** pour la création d'un modèle relationnel cohérent.

3. **Deux modes de présentation :**

   - **SSR (Server-Side Rendering)** : Interface d'administration en **Thymeleaf + Bootstrap**.

   - **CSR (Client-Side Rendering)** : Exposition d'une **API REST** consommée par un mini-client SPA (par exemple Angular ou React, selon le niveau des étudiants).

Modules fonctionnels suggérés

1. Gestion des entités principales :

- **Étudiant** : matricule, nom, prénom, email, date d'inscription.
- **Formateur** : id, nom, spécialité, email.
- **Cours** : code, titre, description, formateur, liste d'étudiants inscrits.
- **Inscription** : date, étudiant, cours.
- **Note** : valeur, étudiant, cours.

2. Fonctionnalités de base :

- **Étudiant** : Ajouter, modifier, supprimer, consulter les étudiants
- **Formateur**: Ajouter, modifier, consulter, associer aux cours
- **Cours** : Créer, planifier, lier à un formateur
- **Inscription** : Inscrire un étudiant à un cours, annuler une inscription
- **Notes** : Attribuer et consulter les notes

3. Gestion administrative avancée

- Gérer les sessions pédagogiques (semestres, années scolaires).
- Gérer les spécialités (Informatique, Réseaux, IA, etc.).
- Gérer les groupes d'étudiants (ex. : groupe TP1, TP2).
- Affecter des cours à plusieurs groupes.

4. Planning et emploi du temps

- Planifier des séances de cours (date, heure, salle).
- Permettre aux étudiants de consulter leur emploi du temps.
- Empêcher les conflits d'horaires (ex. : un étudiant ou un formateur ne peut pas être affecté à deux cours au même moment).

5. Reporting et statistiques

- Afficher la moyenne générale d'un étudiant.
- Calculer le taux de réussite d'un cours.
- Générer un rapport PDF (avec JasperReports ou iText) sur les notes.
- Visualiser un tableau de bord des cours les plus suivis.

6. Communication et notifications

- Envoyer un email automatique à un étudiant lors d'une inscription.
- Notifier le formateur quand un étudiant s'inscrit ou se désinscrit d'un cours.
- (Optionnel) Intégrer un service d'envoi d'email simulé (MockMailService).

7. Sécurité et rôles utilisateurs

- Authentification (via Spring Security, login + mot de passe).
- Autorisations selon le rôle :

✓ **ADMIN** : gestion complète du système.
✓ **FORMATEUR** : gestion des cours et notes.
✓ **ETUDIANT** : consultation des cours, notes, emploi du temps.

- Gestion du profil utilisateur (changement de mot de passe, mise à jour des informations).

8. Distribution SSR / CSR

- Interface Thymeleaf (SSR) : pour l'administration et la gestion du contenu.
- API REST (CSR) : exposer les données pour un client Angular/React.
  - **/api/etudiants** : CRUD étudiants.
  - **/api/cours** : liste des cours disponibles.
  - **/api/inscriptions** : gestion des inscriptions.
- ARC pour tester les endpoints REST.

9. Gestion technique et devops (bonus)

- Sauvegarde et restauration des données (script SQL ou service de backup).
- Déploiement via Docker Compose (base de données + app Spring Boot).
- Profil dev (H2 en mémoire) et prod (MySQL/PostgreSQL).

10. Suggestion pour le rapport UML

Les étudiants peuvent modéliser :

- Diagramme de cas d'utilisation (rôles : admin, étudiant, formateur).
- Diagramme de classes avec les relations enrichies (Cours ↔ Formateur ↔ Groupe ↔ Étudiant).
- Diagramme de séquence pour un scénario "Inscrire un étudiant à un cours".
- Diagramme de composants pour représenter l'architecture SSR/CSR.

Architecture logique (à modéliser)

- ✓ **Couche Web**: Spring MVC (Controller + Thymeleaf) +API REST
- ✓ **Couche Service:** Logique métier
- ✓ **Couche Persistance:** Spring Data JPA (Repositories +entités)
- ✓ **Database** : MySQL

Travail demandé

1. Déposer le code source du travail réalisé
2. Préparer un rapport décrivant la solution (spécification, architecture, conception détaillée et captures de réalisation)
3. Préparer une vidéo pour présenter la réalisation