

DIZIN Jordan, EL BEZZARI Nahil, LESHOB Zakaria, MEZNI Rayen,  
CHAFAQI Rayane  
ING2 Génie des Systèmes d'Information



# JAKARTA EE

## Rapport de projet Java EE Présentation de CoffeeClasses Site de gestion de scolarité

Java EE  
HADDACHE MOHAMED  
CY Tech, 2024-2025  
**Date de rendu** : 30 novembre 2024



## Sommaire :

1) <u>Introduction</u> .....	3
2) <u>Conception de Coffee Classes</u> .....	3
3) <u>Implémentation de Coffee Classes</u> .....	5
a) Technologies utilisées.....	5
b) Fonctionnement de l'application.....	6
4) <u>Conclusion</u> .....	14

# 1 - Introduction

Dans le cadre du module de Java EE, il nous a été demandé de concevoir et développer une application web de gestion de scolarité visant à répondre aux besoins des étudiants, enseignants et administrateurs. L'objectif principal était de fournir une plateforme centralisée et efficace pour gérer des informations académiques telles que les inscriptions aux cours, les résultats, et les données personnelles.

Il s'agissait donc de mettre en œuvre nos connaissances et compétences en Java, mais aussi en développement web avec Jakarta EE et en gestion de données avec Hibernate. Nous avons également dû appliquer des concepts vus en cours, comme l'architecture MVC (Modèle-Vue-Contrôleur). Une version Spring Boot du projet a également été demandée, ce qui nous a permis de comparer les deux approches de développement web en Java.

Le projet CoffeeClasses tire son nom de l'icône de Java, une tasse de café. La version Jakarta EE, surnommée "Vanilla", référence le terme évoquant l'édition originale, non modifiée d'un jeu vidéo et évoque une "saveur", tandis que la version Spring Boot, appelée "Spring", s'inspire du framework tout en renforçant l'analogie avec une édition limitée de café.

## 2 - Conception de CoffeeClasses

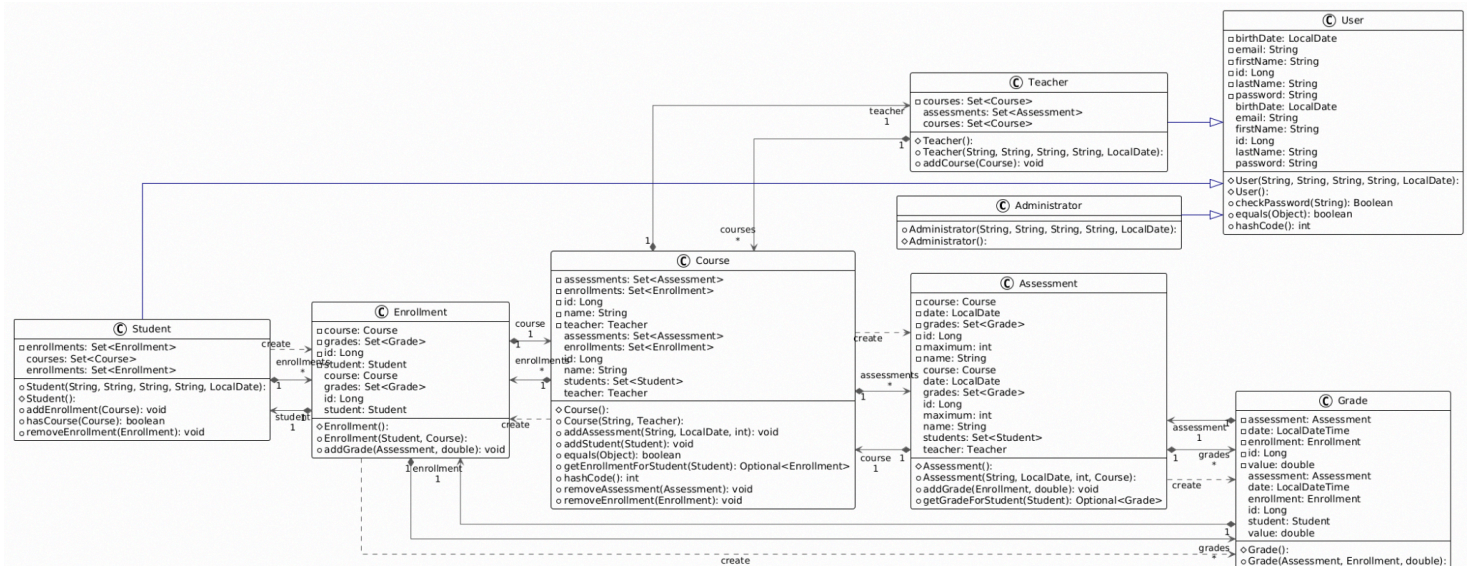
### Cahier des charges

La première étape du projet a été d'établir les entités nécessaires au fonctionnement du programme. En se référant à l'énoncé du projet :

- **Gestion des étudiants** : Il doit non seulement être possible d'enregistrer et de mettre à jour les informations des étudiants, mais aussi de les supprimer si nécessaire. Une liste des étudiants doit pouvoir être affichée, on doit pouvoir consulter les détails d'un étudiant, et enfin rechercher et filtrer les étudiants.
- **Gestion des enseignants** : De même pour les enseignants, cependant, il faut aussi pouvoir les affecter à des cours.
- **Gestion des cours** : On doit bien évidemment pouvoir les créer, les supprimer, afficher leur liste, et attribuer des cours aux étudiants et enseignants.
- **Saisie des notes** : Il faut que les professeurs puissent assigner des notes aux étudiants.
- **Gestion des résultats** : De plus, ces notes doivent être visibles, des moyennes doivent pouvoir être affichées, et il doit être possible de consulter les résultats par étudiant et par cours.

- **Authentification et autorisation** : 3 rôles doivent être gérés : Étudiant, Enseignant et Administrateur.

## Solution



Le modèle de CoffeeClasses repose sur une architecture conçue pour répondre aux exigences du cahier des charges. Pour cela, nous avons organisé le projet autour de huit entités distinctes, dont trois sont des extensions d'une entité principale. Ces entités sont structurées en deux groupes logiques dans le code source : "éléments" et "utilisateurs", afin de répondre aux besoins de gestion des données académiques et des rôles d'utilisateurs.

Le groupe "éléments" regroupe les entités représentant les données manipulées par l'application :

- **Résultats** : Notes obtenues par les étudiants.
- **Inscriptions** : Gestion des enregistrements des étudiants dans des cours.
- **Cours**
- **Évaluations** : Associent des résultats aux cours attribués aux étudiants.

Le groupe "utilisateurs" est dédié aux rôles dans l'application et regroupe quatre classes :

- **Étudiant**
- **Professeur**
- **Administrateur**
- **Utilisateur** (classe abstraite dérivée par les autres)

Cette structure exploite ici des fonctionnalités offertes par Java, telles que l'héritage et la modularité, pour simplifier la gestion des rôles. L'architecture en héritage assure

une gestion uniforme des utilisateurs, facilitant l'implémentation de fonctionnalités communes (comme la connexion) sans duplication de code, ce qui optimise et réduit la complexité du développement.

En intégrant cette structure avec Jakarta EE, Hibernate (et Spring Boot), le modèle de CoffeeClasses assure donc une gestion flexible et simple des données académiques et des utilisateurs.

## 3 - Implémentation de CoffeeClasses

### a) Technologies utilisées

#### Organisation du projet

CoffeeClasses a été développé avec Git, assurant un suivi des modifications via l'historique et facilitant le travail en équipe grâce aux branches. Maven a géré les dépendances et permettait de créer une archive WAR portable, tandis que le plugin Cargo (ou Spring Boot lui-même) intégrait un serveur Tomcat pour exécuter le projet. GitHub Actions automatise les tests de compilation à chaque commit.

#### Base de données

Bien que MySQL (ou MariaDB) soit une option simple et déjà installée, PostgreSQL a été choisi pour ses fonctionnalités avancées, ses meilleures performances dans certains cas, et une gestion des erreurs plus efficace. Ce choix s'est révélé pertinent, car PostgreSQL a facilité la localisation de deux dysfonctionnements du projet.

#### Frontend

CoffeeClasses utilise un système de templating simple mais fonctionnel. Les pages JSP, stockées dans le dossier "WEB-INF/views", sont organisées en sous-dossiers : "contents" pour les contenus, "layouts" pour les interfaces, et "pages" pour relier les contenus aux interfaces. Chaque page "contenu" est automatiquement associée à un fichier CSS. Une tentative d'utilisation de SCSS a été abandonnée en raison de problèmes de flexibilité sous Jakarta EE et de compilation sous Windows.

#### Backend

Le backend de CoffeeClasses repose sur différentes architectures selon la version. Dans la version Jakarta, l'application utilise des Servlets pour gérer les requêtes HTTP, des DAOs pour l'accès aux données, des Services pour la logique métier, des Filters en middleware pour vérifier l'authentification des utilisateurs et leur rôle avant d'accéder à une page, ainsi que des entités pour modéliser les données.

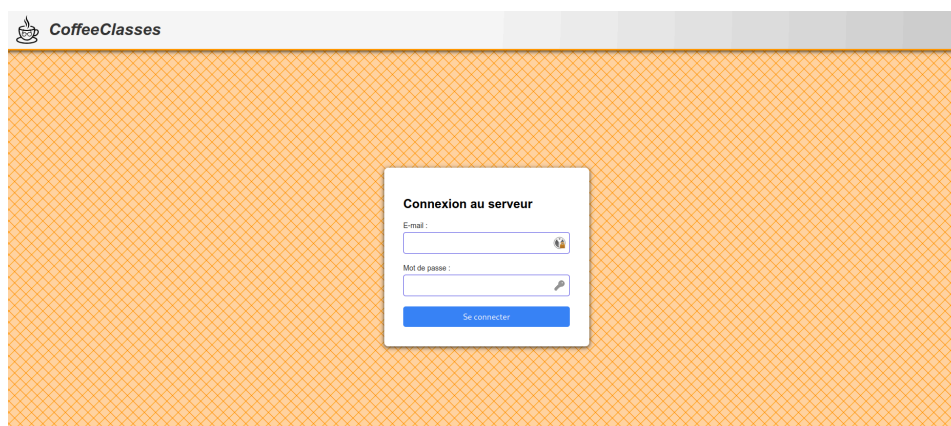
Dans la version Spring Boot, l'architecture a été simplifiée avec des Controllers pour gérer les requêtes, des Repository pour l'accès aux données, des Services pour la logique métier, des Interceptors en middleware pour l'interception et la vérification de l'authentification et des rôles des utilisateurs avant l'accès aux pages, et des DTOs pour transférer les données entre les différentes couches de l'application. On retrouve également les entités de la version Jakarta.

### Autres

- **Envoi de mail** : Dans la version Jakarta EE, l'envoi de mails est géré par Jakarta Mail, une solution déjà intégrée dans l'API Jakarta. Pour la version Spring Boot, l'envoi de mails utilise le support intégré du framework Spring, simplifiant l'intégration et la gestion des notifications par email.
- **Génération de fichiers PDF** : La génération de fichiers PDF est réalisée avec la bibliothèque iTextPdf, permettant de créer dynamiquement des relevés de notes pour les étudiants.
- **Journalisation** : La journalisation des événements du système est assurée par Logback Classic en combinaison avec SLF4J, permettant une gestion flexible et performante des logs pour suivre l'exécution de l'application. La version Spring Boot se sert des dépendances de Spring.
- **Hashage des mots de passe** : Nous utilisons Password4J, une bibliothèque fiable pour le hashage des mots de passe avant leur stockage dans la base de données.

### b) Fonctionnement de l'application

#### Connexion



Cette page s'affiche lorsqu'une personne non connectée tente d'accéder à un lien sur le serveur. Elle est gérée par la servlet **LoginServlet** (ou **LoginController** dans le cas de la version Spring). Une requête GET renvoie cette page, tandis que le

formulaire intégré envoie une requête POST à la servlet correspondante, qui se charge de valider les informations fournies avant de connecter l'utilisateur.

Pour garantir une gestion rigoureuse des valeurs potentiellement absentes, le type **Optional** de Java est largement utilisé dans l'ensemble du programme.

```
// Login

@RequestMapping(value="/login", method = RequestMethod.GET)
public String getLoginPage(HttpSession session) {
    if (session.getAttribute("user") == null) {
        return JSP_PATH;
    } else {
        return "redirect:/panel/home";
    }
}

@RequestMapping(value = "/login", method = RequestMethod.POST)
public String login(LoginRequestDTO loginDTO, HttpSession session, Model model) {
    Optional<User> user = authService.authenticate(loginDTO);
    if (user.isPresent()) {
        session.setAttribute("userId", user.get().getId());
        return "redirect:/panel/home";
    } else {
        model.addAttribute("errorMessage", "Mot de passe ou mail invalide.");
        return JSP_PATH;
    }
}

}

You, il y a 22 heures | 1 author (You)
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public Optional<User> authenticate(LoginRequestDTO login) {
        Optional<User> userOpt = userRepository.findByEmail(login.getEmail());
        if (userOpt.isPresent() && userOpt.get().checkPassword(login.getPassword())) {
            return userOpt;
        } else {
            return Optional.empty();
        }
    }
}
```

Un point crucial souvent ignoré : les mots de passe ne doivent **jamais** être stockés en clair dans la base de données, une pratique indispensable pour assurer la sécurité. Nous utilisons l'algorithme Argon2id pour générer un hash sécurisé, qui est stocké. Lors de la connexion, le mot de passe saisi est re-haché avec les mêmes paramètres, et les hashes sont comparés, garantissant ainsi la validation sans jamais exposer le mot de passe.

Par ailleurs, seul l'identifiant de l'utilisateur est conservé dans la session. Cela évite toute désynchronisation avec la base de données : à chaque chargement de page, les informations de l'utilisateur sont récupérées via son identifiant et ajoutées en attribut de requête.

```

You, il ya 2 semaines | 1 author (You)
@WebFilter(filterName = "userFilter")
public class UserFilter implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;
        HttpSession session = httpRequest.getSession(create:false);

        // Check if user is logged in
        if (session == null || session.getAttribute(name:"userId") == null) {
            httpResponse.sendRedirect(httpRequest.getContextPath() + "/login");
            return;
        }

        // Transfer user to request attributes for convenience
        User user = UserService.getInstance().find((Integer) session.getAttribute(name:"userId")).orElseThrow(() -> new DataNonsenseException(mess
        request.setAttribute(name:"user", user);

        // Continue the filter chain
        chain.doFilter(request, response);
    }

    @Override
    public void destroy() {}
}

```

## Interface

The screenshot shows the 'Default Student' page of the 'CoffeeClasses' application. The page has a header with the application name and navigation links. The main content area is titled 'Default Student' and contains a section for 'Cours et évaluations'. Under this section, there is a 'Test' card showing the instructor 'Default Teacher' and exam results: 'Errr : 20.0/20', 'Final : 25.0/50', and 'Moyenne : 15 / 20'. A button at the bottom of the card allows downloading the notes in PDF format.

Chaque page du site, gérée par une servlet (ou un contrôleur), appelle le fichier d'interface correspondant : **pre-auth-layout.jsp** pour la page de connexion ou **panel-layout.jsp** pour les autres. Elle transmet le titre de la page et le nom du fichier de contenu à inclure.

```

src > main > webapp > WEB-INF > views > pages > panel > student > <> student-summary.jsp > ?
You, il y a 6 jours | 1 author (You)
1  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
2  <jsp:include page="/WEB-INF/views/layouts/panel-layout.jsp">
3      <jsp:param name="title" value="Résumé" />
4      <jsp:param name="contentPage" value="student-summary" />
5  </jsp:include>
6

```

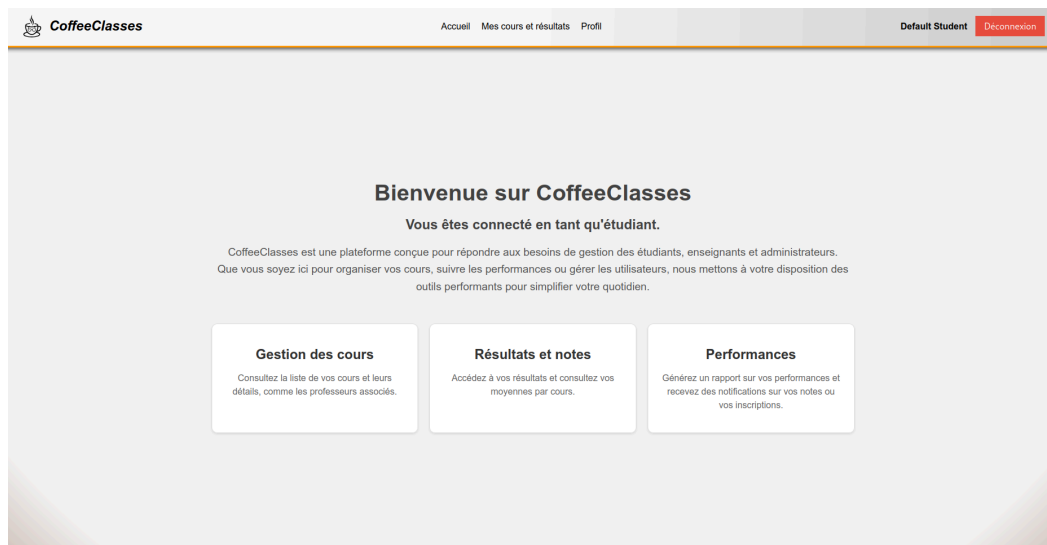


[illegible]

```
<!-- Liens de navigation -->
<div class="panel-links">
  <a href="{pageContext.request.contextPath}/panel/home">Accueil</a>
  <c:choose>
    <c:when test="{userType == 'Administrator'}">
      <a href="{pageContext.request.contextPath}/panel/admin/users">Utilisateurs</a>
      <a href="{pageContext.request.contextPath}/panel/admin/courses">Cours</a>
    </c:when>
    <c:when test="{userType == 'Teacher'}">
      <a href="{pageContext.request.contextPath}/panel/teacher/assessments">Mes évaluations</a>
    </c:when>
    <c:when test="{userType == 'Student'}">
      <a href="{pageContext.request.contextPath}/panel/student/details">Mes cours et résultats</a>
    </c:when>
    <c:otherwise>
    </c:otherwise>
  </c:choose>
  <a href="{pageContext.request.contextPath}/panel/users/show?id={%= user.getId() %}">Profil</a>
</div>

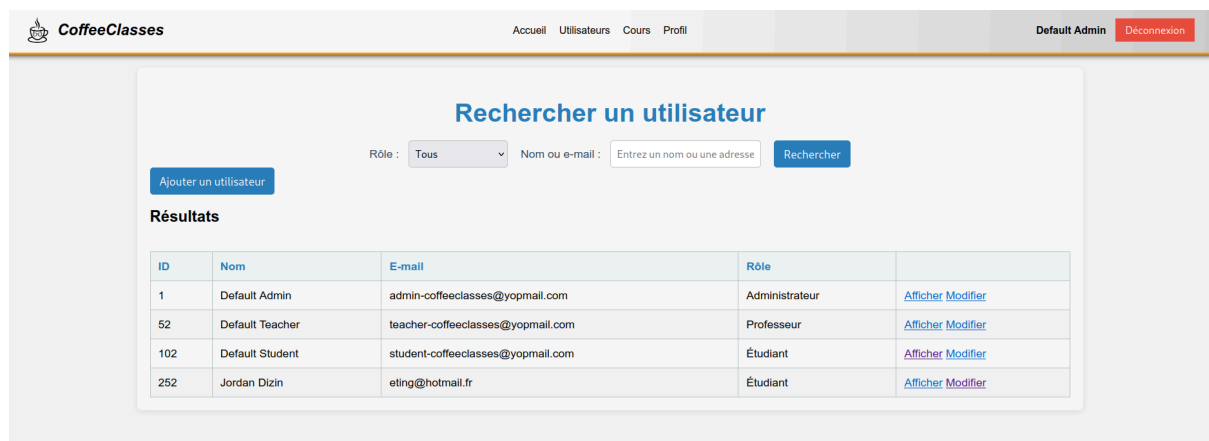
<!-- Nom d'utilisateur et déconnexion -->
<div class="user-actions">
  <span class="username">${user.firstName} ${user.lastName}</span>
  <a href="{pageContext.request.contextPath}/logout" class="logout-form">
    <button type="submit" class="logout-button">Déconnexion</button>
  </a>
</div>
```

## Page d'accueil



La page d'accueil explique simplement l'utilisation de l'outil aux nouveaux utilisateurs et les fonctionnalités disponibles selon leur rôle.

## Fonctionnalités pour les administrateurs



La page de gestion des utilisateurs affiche une liste que l'on peut filtrer via une fonctionnalité de recherche. Pour chaque utilisateur, il est possible de consulter ou modifier ses données personnelles, ainsi que de le supprimer.

La gestion des cours fonctionne de manière similaire.



En version Jakarta, chaque fonctionnalité est généralement associée à une servlet, avec une méthode pour afficher la page en GET et une autre pour effectuer l'action en POST. Pour les fonctions de recherche, les JSP renvoient vers leur servlet correspondante en GET avec les paramètres nécessaires.

Avec Spring Boot, toutes ces fonctionnalités sont centralisées de manière simple et directe dans **CourseAdminController** et **UserAdminController**.

```
J CourseAdminController.java 2 x
src > main > java > fr > cyu > coffeeclasses > spring > controller > panel > admin > J CourseAdminController.java > {} fr.cyu.coffeeclasses.spring.controller.panel.admin
26 public class CourseAdminController {
27     // Delete courses
28
29     @GetMapping("/delete")
30     public String showDeletePage(@RequestParam(name = "id") Long id, HttpServletRequest request) {
31         Optional<Course> target = courseRepository.findById(id);
32         if (target.isPresent()) {
33             request.setAttribute("target", target.get());
34             return "panel/admin/course-management/course-delete";
35         } else {
36             throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Cours non trouvé.");
37         }
38     }
39
40     @PostMapping("/delete")
41     public String deleteUser(CourseDeleteRequestDTO dto) {
42         Optional<Course> target = courseRepository.findById(dto.getId());
43         if (target.isPresent()) {
44             courseRepository.delete(target.get());
45             return "redirect:/panel/admin/courses";
46         } else {
47             throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Cours non trouvé.");
48         }
49     }
50
51     // Edit
52
53     @GetMapping("/edit")
54     public String showEditPage(@RequestParam(name = "id") Long id, HttpServletRequest request) {
55         Optional<Course> target = courseRepository.findById(id);
56         if (target.isPresent()) {
57             request.setAttribute("targetCourse", target.get());
58             request.setAttribute("teachers", userRepository.searchUsersByRoleAndString(role:Teacher.class, search:null));
59             return "panel/admin/course-management/course-edit";
60         } else {
61             throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Cours non trouvé.");
62         }
63     }
64
65     @PostMapping("/edit")
66     public String editCourse(CourseEditRequestDTO dto, HttpServletRequest request) {
67         Optional<Course> targetOpt = courseRepository.findById(dto.getId());
68         if (targetOpt.isPresent()) {
69             // Get teacher
70             Teacher teacher = (Teacher) userRepository.findById(dto.getTeacher()).orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOU
71             Course target = targetOpt.get();
72         }
73     }
74 }
```

## Fonctionnalités pour les professeurs

Les professeurs ne peuvent que gérer les évaluations des cours qu'ils gèrent, ainsi que les notes des élèves du cours associé pour chaque évaluation.

## Gestion des notes et évaluations

### Mes évaluations

Final

Noter/modifier les notes

Errr

Noter/modifier les notes

### Ajouter une évaluation

Cours à évaluer:

Test

Nom de l'évaluation:

Note maximale:

Date de l'évaluation :

jj / mm / aaaa

Ajouter

CoffeeClasses

Accueil Mes évaluations Profil

Default Teacher Déconnexion

#### Attribution des notes

Final

Cours : Test

Note maximale : 50/ 50

Nom	Prénom	Note
Default	Student	25.0
Jordan	Dizin	30.0

Enregistrer les notes

Leur profil affiche la liste des cours qu'ils gèrent.

CoffeeClasses

Accueil Mes évaluations Profil

Default Teacher Déconnexion

## Default Teacher

### Informations personnelles

Nom : Teacher

Prénom : Default

Adresse e-mail : teacher-coffeeclasses@yopmail.com

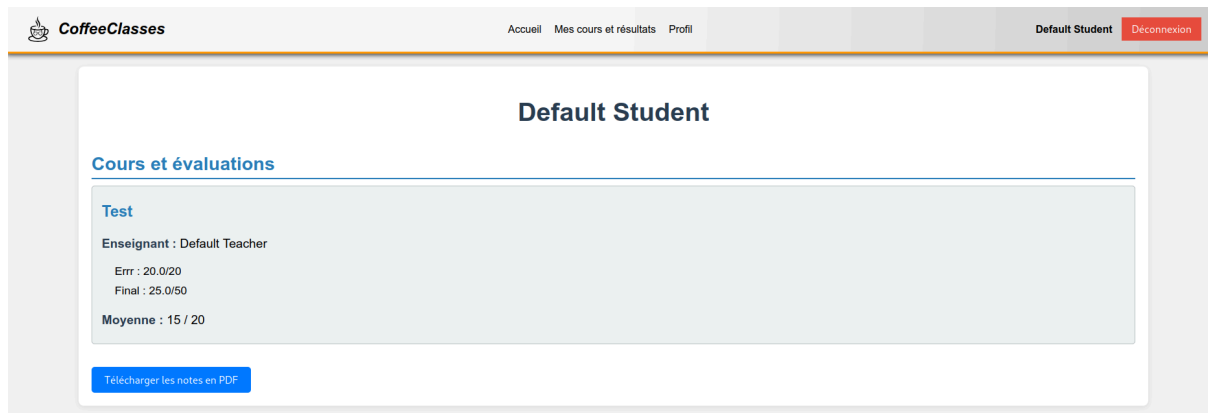
Date de naissance : 1990-05-15

### Cours

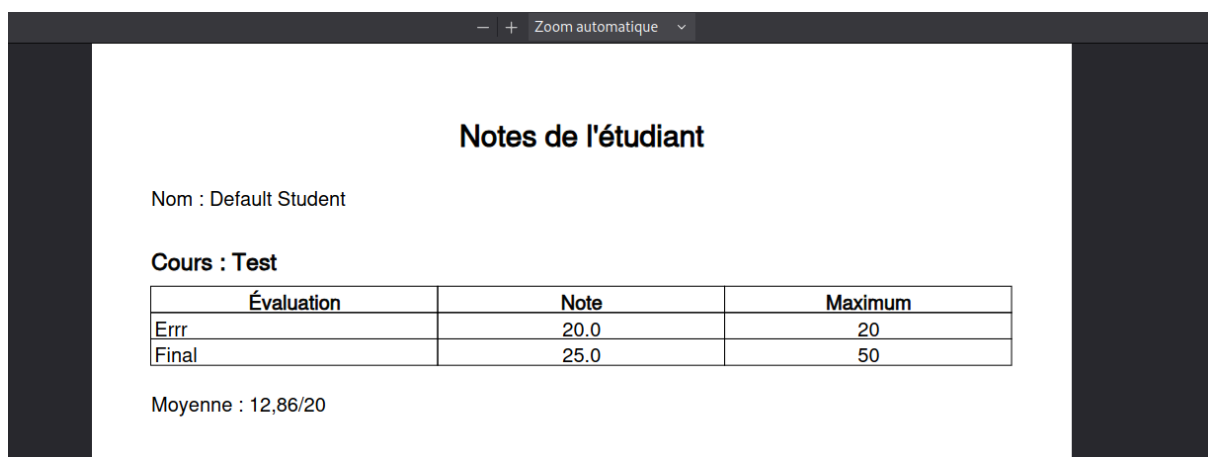
- Test

## Fonctionnalités pour les étudiants

Un étudiant peut simplement voir la liste de ses cours, évaluations et notes.



Un rapport de ses notes peut être téléchargé sous forme de PDF.



De plus, à chaque modification des notes ou des données personnelles d'un étudiant, un mail lui est envoyé.

```
12 @Service
13 public class MailService {
14     private static final Logger logger = LoggerFactory.getLogger(MailService.class);
15
16     @Autowired
17     private MailSender mailSender;
18
19     public void sendMail(User dest, String subject, String body) {
20         try {
21             SimpleMailMessage message = new SimpleMailMessage();
22             message.setTo(dest.getEmail());
23             message.setSubject(subject);
24             message.setText(body);
25             mailSender.send(message);
26         } catch (Exception e) {
27             logger.error("Error while sending email: {}", e.getMessage());
28         }
29     }
30 }
```

## 4 - Conclusion

Le projet CoffeeClasses nous a permis de mettre en œuvre une solution complète de gestion de scolarité, intégrant à la fois des aspects techniques de développement web et de gestion de données académiques. Cependant, plusieurs contraintes ont accompagné sa réalisation. Le manque de temps, combiné à la gestion de travaux dans d'autres matières, a rendu l'avancement du projet plus complexe. De plus, le travail en équipe a soulevé certaines difficultés de coordination, notamment lors de la mise en place de la structure du projet. La gestion des dépendances et la configuration initiale du projet ont également posé des problèmes techniques, nécessitant des ajustements et des résolutions de bugs imprévus.

Malgré ces défis, le projet répond aux exigences du cahier des charges. La comparaison entre Jakarta EE et Spring Boot a permis d'observer les avantages de chaque approche, tout en consolidant nos connaissances en développement Java, architecture MVC, et gestion de bases de données avec Hibernate et PostgreSQL. Ce projet a été l'occasion de renforcer notre maîtrise des technologies Java EE et Spring Boot, tout en nous permettant d'approfondir nos compétences pratiques en matière de développement d'applications web.

Il reste de nombreuses pistes d'amélioration, notamment en ce qui concerne la présentation de l'interface et l'organisation des fonctionnalités. L'interface utilisateur offre déjà une base fonctionnelle, mais pourrait être enrichie pour offrir une expérience encore plus intuitive et cohérente. De même, certaines fonctionnalités gagneraient à être réorganisées pour une meilleure accessibilité. Par ailleurs, commencer par la version Spring Boot aurait pu permettre de poser des fondations solides en se concentrant sur les parties essentielles du projet, comme la gestion des données et la logique métier, avant d'explorer les aspects plus techniques de Jakarta EE. Cela aurait probablement facilité la mise en place initiale et optimisé le temps de développement.