FSBM

FACULTE DES SCIENCES BEN M'SICK
UNIVERSITÉ HASSAN II DE CASABLANCA

Data Science & Big Data
Master DSBD

MASTER : BIG DATA ET DATA SCIENCE

MODULE : JEE & FRAMEWORKS

ENCADRE PAR :

PR. ABDESSAMAD BELANGOUR

APPLICATION DE GESTION DE RÉSIDENCES ÉTUDIANTES

Réalisé par

BENICHE Mohamed Fadel

2024-2025

Résumé

Le projet "ResidenceManager" est une application web conçue pour la gestion de résidences étudiantes. Son objectif principal est d'optimiser les processus administratifs et d'améliorer l'expérience utilisateur pour les étudiants.

Les fonctionnalités clés de l'application comprennent :

- ✓ Gestion des chambres : consultation, filtrage et réservation des chambres.
- ✓ Gestion des étudiants : inscription, gestion des profils et consultation de l'historique des paiements.
- ✓ Gestion des paiements : réalisation et suivi des paiements par les étudiants.
- ✓ Gestion des incidents : signalement des problèmes et suivi de leur résolution.
- ✓ Outils administratifs : gestion des étudiants, des techniciens, des paiements, et des statistiques pour l'administration.

L'architecture de l'application repose sur une structure multicouche, avec un backend développé en Spring Boot, une base de données MySQL et une communication via API RESTful. L'application est construite avec un souci de performance, de sécurité et d'accessibilité, et s'appuie sur des technologies éprouvées pour assurer sa fiabilité.

Les outils utilisés pour le développement comprennent Spring Boot, MySQL Workbench et IntelliJ IDEA.

Enfin, l'application propose une interface utilisateur intuitive pour les étudiants et les administrateurs, permettant une navigation aisée et un accès rapide aux différentes fonctionnalités.

Liste des figures

Sommaire

Introduction Générale

Le projet de développement de l'application de gestion des résidences étudiantes vise à moderniser et automatiser la gestion quotidienne des résidences en offrant une plateforme centralisée et accessible pour l'administration et les étudiants. Ce système web permettra de gérer de manière fluide et sécurisée les chambres, les inscriptions des étudiants, les paiements, les incidents de maintenance, et d'autres opérations administratives essentielles. Grâce à une interface utilisateur intuitive et réactive, les étudiants pourront facilement consulter la disponibilité des chambres, effectuer des paiements en ligne, signaler des incidents et suivre leur historique, tandis que l'administration pourra gérer les données des étudiants, effectuer un suivi des paiements, et assigner des tâches de maintenance aux techniciens.

Dans ce cadre, ce projet de développement s'appuie sur des technologies modernes telles qu'avec Maven, et une base de données MySQL. La communication entre le frontend et le backend est assurée via des API RESTful.

Ce rapport de conception présente les différentes étapes du projet, les diagrammes UML qui illustrent la structure et le fonctionnement du système, ainsi que le cahier des charges détaillant les objectifs et exigences fonctionnelles du projet et le cote réalisation.

Objectifs du Projet

L'objectif principal de ce projet est de développer une application web moderne et fonctionnelle qui facilitera la gestion des résidences étudiantes en centralisant l'ensemble des opérations administratives et en offrant une expérience utilisateur fluide et sécurisée pour les étudiants et l'administration. Les objectifs spécifiques du projet sont les suivants :

1. **Simplifier la gestion des chambres :**
    - Permettre à l'administrateur de gérer facilement les chambres (ajout, modification, suppression, consultation), et d'afficher en temps réel leur disponibilité.
2. **Faciliter l'inscription et la gestion des étudiants :**
    - Offrir une interface d'inscription simple pour les étudiants, permettant à ces derniers de créer leur profil, de consulter leurs informations, et de suivre leurs paiements.
3. **Gérer efficacement les incidents de maintenance :**
    - Offrir aux étudiants la possibilité de signaler des incidents (plomberie, électricité, etc.), et permettre à l'administration de suivre l'affectation et la résolution de ces incidents en temps réel.
4. **Fournir des outils de gestion et de suivi pour l'administration :**
    - Mettre en place des outils de gestion pour les administrateurs, permettant de consulter des statistiques sur le taux d'occupation des chambres, les paiements effectués et en retard, ainsi que les incidents en cours.
5. **Offrir une interface utilisateur intuitive et responsive :**
    - Développer une interface web qui soit facile à utiliser, responsive et accessible sur tous les appareils (ordinateurs, tablettes, smartphones).

Chapitre I

La Conception

Introduction

L'objectif de ce projet est de développer une **application web de gestion des résidences étudiantes**. Cette application permettra à l'administration de gérer les chambres, les résidents, les paiements, ainsi que les requêtes de maintenance, tout en centralisant les informations pour simplifier la gestion quotidienne de la résidence.

Le système inclura des fonctionnalités telles que la gestion des chambres, l'inscription des étudiants, le suivi des paiements, la gestion des incidents et la génération de rapports statistiques. Voici la description détaillée des diagrammes UML créés pour représenter le processus et la structure de l'application.

1. Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation illustre les interactions entre les acteurs (administration, étudiant, technicien) et les fonctionnalités du système.

Acteurs principaux :

- **Étudiant** : Un utilisateur qui s'inscrit, sélectionne une chambre, effectue des paiements, et signale des incidents.
- **Administrateur** : Un utilisateur avec un accès complet, capable de gérer les chambres, les étudiants, les paiements, et les incidents.

Cas d'Utilisation :

1.1. Étudiant :

- **S'authentifier** : L'étudiant doit s'authentifier pour accéder à son profil.
- **Sélectionner une chambre** : L'étudiant consulte les chambres disponibles et en sélectionne une.
- **Effectuer un paiement** : L'étudiant paie son loyer via l'application.
- **Faire un rapport d'incident** : L'étudiant signale un problème (plomberie, électricité).
- **Consulter la chambre** : L'étudiant consulte les détails de sa chambre (disponibilité, statut).

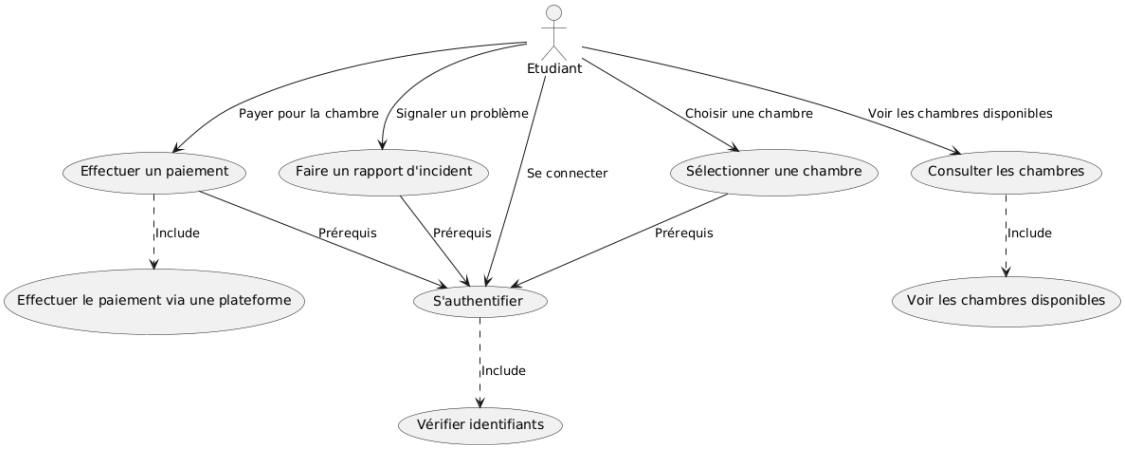

Figure 1 Cas d'utilisation : Etudiant

1.2. Administrateur :

- **Gérer les chambres** : L'administrateur peut créer, modifier ou supprimer des chambres.
- **Gérer les étudiants** : Inscription, consultation, et mise à jour des informations des étudiants.
- **Gérer les paiements** : Suivi des paiements des étudiants, génération de reçus, et envoi de rappels.
- **Gérer les incidents** : Gestion des incidents et affectation à des techniciens.

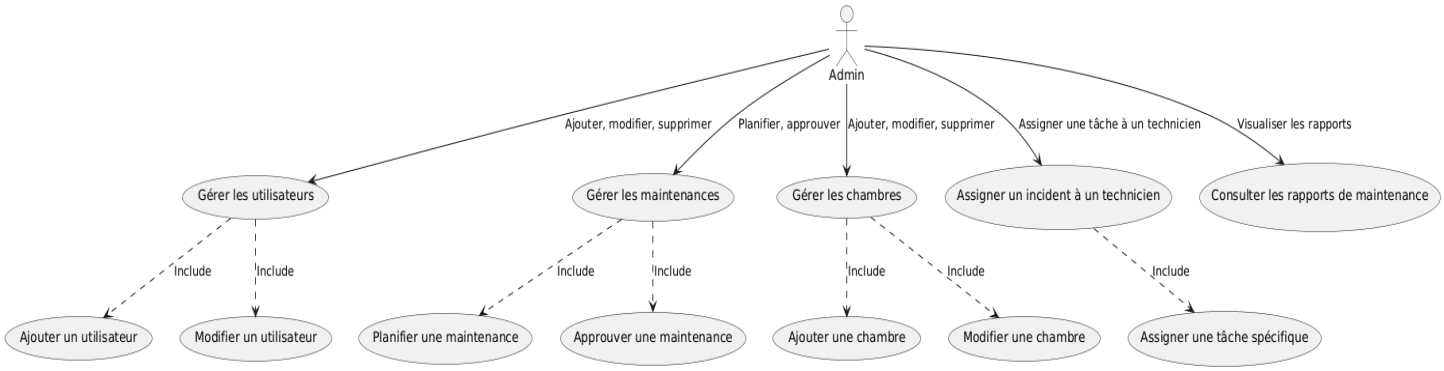

Figure 2 Cas d'utilisation : Admin

2. Diagrammes de Classes

Les diagrammes de classes représentent la structure de données du système et les relations entre les différentes entités.

Classes principales :

- **Chambre** : Contient des attributs tels que la taille, les équipements, et le statut de la chambre.
- **Etudiant** : Contient les informations personnelles, l'historique des paiements, et le statut de la chambre.
- **Paiement** : Représente les paiements effectués par les étudiants, avec des informations sur le montant, la date et le statut.
- **Maintenance** : Contient les détails de chaque incident signalé, tels que le type de problème et son statut de résolution.
- **Technicien** : Assigne et résout les incidents.
- **Admin**

Les relations entre les classes sont les suivantes :

- **Etudiant** peut avoir plusieurs **Paiements**.
- Une **Chambre** peut être occupée par plusieurs **Etudiants** au fil du temps.
- Un **Etudiant** peut signaler plusieurs **Incidents**.
- Un **Incident (Maintenance)** est assigné à un **Technicien** qui le résout.

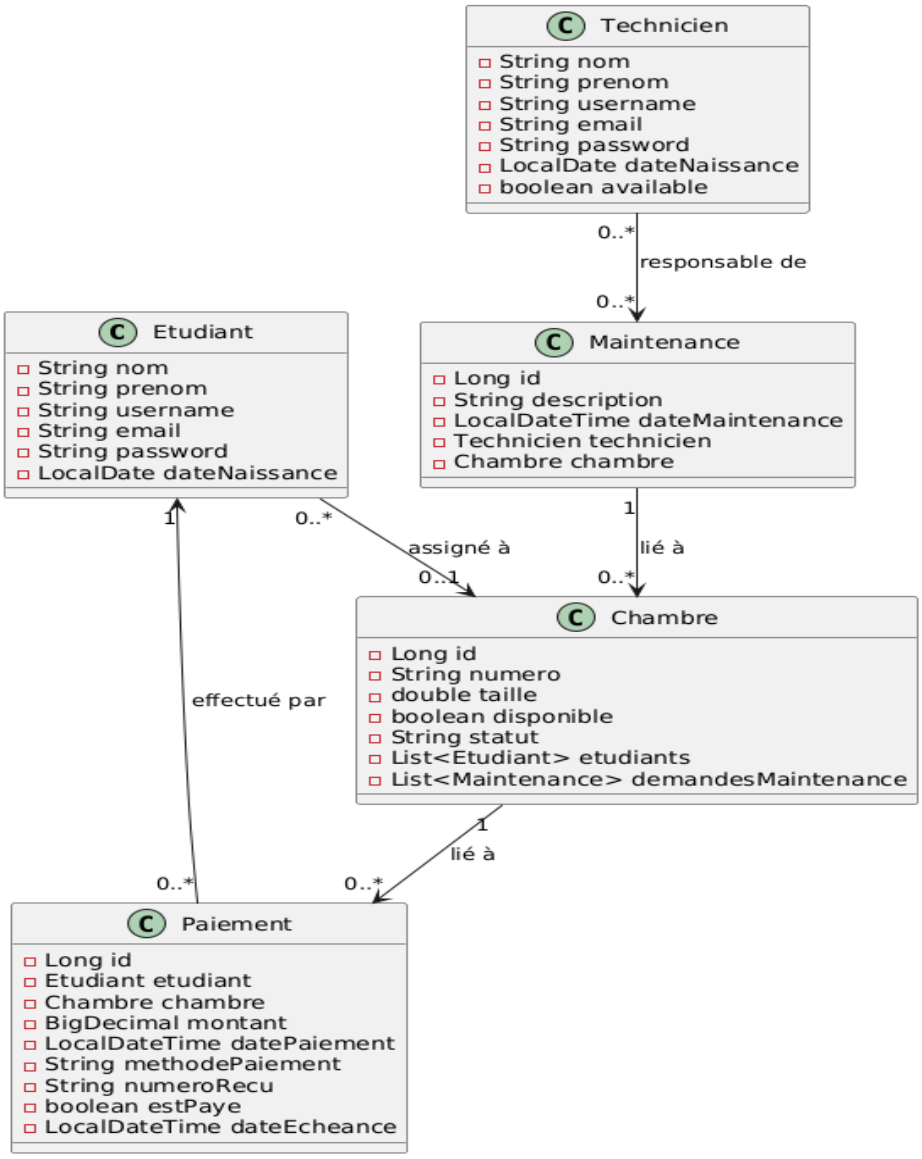

Figure 3 Diagramme de classe

3. Diagramme de Séquence

Le diagramme de séquence montre l'enchaînement des actions lors de l'authentification d'un étudiant et des interactions avec les autres fonctionnalités. Ce diagramme illustre les étapes suivantes :

1. **Authentification de l'étudiant** :
   - L'étudiant s'authentifie avec ses identifiants.
   - Le système vérifie les identifiants dans la base de données.
   - Si l'authentification est réussie, l'étudiant peut procéder aux autres actions.
2. **Sélection de la chambre** :
   - Une fois authentifié, l'étudiant consulte les chambres disponibles et en sélectionne une.
3. **Effectuer un paiement** :

- L'étudiant effectue un paiement pour la chambre sélectionnée.
- Le système vérifie le solde et confirme le paiement.

4. **Faire un rapport d'incident** :
   - Si un incident est signalé, l'étudiant soumet un rapport d'incident via l'application.
   - Le système enregistre l'incident et l'assigne à un technicien pour résolution.

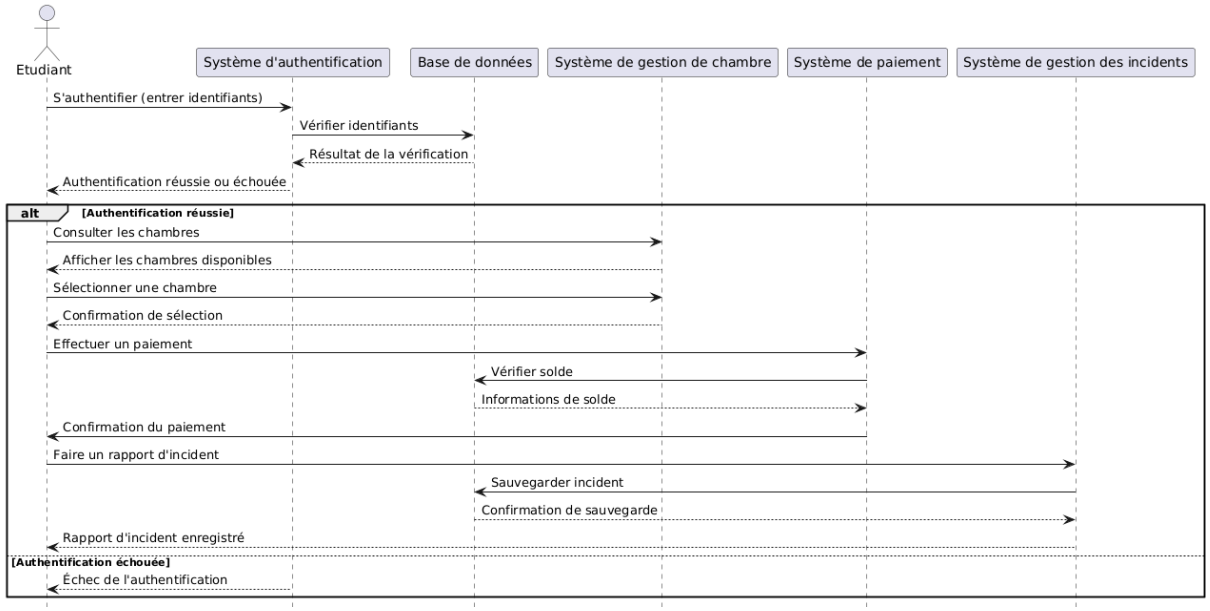

Figure 4 Diagramme de séquence

Conclusion

Ce projet de gestion des résidences étudiantes a été conçu pour offrir une solution centralisée et efficace permettant de gérer les chambres, les paiements, les étudiants et les incidents. Les diagrammes UML fournis permettent de mieux comprendre les relations entre les différents composants du système et d'assurer une mise en œuvre réussie et fluide du processus. L'application vise à simplifier la gestion quotidienne de la résidence et à offrir une expérience utilisateur optimale pour les étudiants et l'administration.

Chapitre II

Cahier des Charges - Application de Gestion de Résidences Étudiantes

Introduction

L'application de gestion de résidences étudiantes a pour objectif de centraliser la gestion des chambres, des étudiants, des paiements, des incidents et des requêtes de maintenance dans une plateforme web accessible à tous les utilisateurs concernés. Ce cahier des charges définit les objectifs, les exigences, et les spécifications de l'application.

1. Objectifs du Projet

L'objectif principal du projet est de développer une application web permettant à l'administration de gérer efficacement les chambres, les étudiants, les paiements, et les requêtes de maintenance. L'application doit également permettre aux étudiants de :

- Sélectionner une chambre disponible.
- Effectuer des paiements.
- Signaler des incidents.
- Consulter leurs informations et leurs paiements.

Les administrateurs pourront quant à eux :

- Gérer les chambres et leur état.
- Gérer les étudiants inscrits, leurs informations et leur historique.
- Suivre les paiements des étudiants.
- Gérer et assigner les requêtes d'incidents aux techniciens.
- Consulter les statistiques générales (taux d'occupation, paiements, incidents).

2. Périmètre du Projet

Le projet couvre les fonctionnalités suivantes :

- Gestion des chambres : création, modification, suppression, consultation.
- Gestion des étudiants : inscription, profil étudiant, mise à jour des informations.
- Suivi des paiements : effectuer des paiements, consulter l'historique des paiements, générer des reçus.
- Gestion des incidents : signalement des incidents, affectation des incidents à un technicien, suivi de la résolution.
- Statistiques générales : taux d'occupation, paiements en retard, incidents en cours.

Le système inclut également des fonctionnalités de recherche et de consultation des chambres, des étudiants, et des incidents.

3. Exigences Fonctionnelles

1.1. Gestion des Chambres

- L'administrateur peut ajouter, modifier et supprimer des chambres.
- Chaque chambre aura des attributs tels que la taille, les équipements, le statut (disponible, occupée, en maintenance).
- Le système doit afficher la disponibilité des chambres en temps réel.

1.2.Gestion des Étudiants

- Les étudiants doivent pouvoir s'inscrire en ligne en fournissant leurs informations personnelles.
- Une fois inscrits, les étudiants peuvent consulter leur profil et leur chambre attribuée.
- Ils peuvent mettre à jour leurs informations de contact.

1.3.Gestion des Paiements

- L'étudiant pourra effectuer un paiement.
- Le système suivra l'historique des paiements des étudiants, avec des détails sur les montants dus, payés, et en retard.
- Des notifications par email seront envoyées en cas de retard de paiement.
- Les étudiants pourront générer et télécharger des reçus de paiement.

1.4.Gestion des Incidents

- Les étudiants pourront signaler des incidents liés à leur chambre (plomberie, électricité, etc.).
- Les incidents seront assignés à un technicien par l'administrateur.
- L'historique des incidents sera accessible par résident et par chambre.

1.5.Statistiques et Rapports

- L'administrateur pourra consulter des rapports de statistiques générales (taux d'occupation, paiements en retard, incidents en cours).

4. Exigences Non Fonctionnelles

4.1.Performance

- L'application doit être rapide et réactive, avec des temps de réponse minimaux.
- Les pages doivent se charger en moins de 3 secondes.

4.2.Sécurité

- Le système doit garantir la confidentialité des données des étudiants, notamment les informations personnelles et les paiements.
- Les étudiants devront se connecter avec un nom d'utilisateur et un mot de passe, et les données doivent être stockées de manière sécurisée.
- Des protocoles sécurisés tels que HTTPS seront utilisés pour le transfert des données.

4.3.Accessibilité

- L'application doit être accessible depuis tous les navigateurs modernes (Chrome, Firefox, Safari, Edge).
- L'interface doit être responsive et accessible sur tous les appareils (ordinateurs, tablettes, smartphones).

4.4.Maintenance

- L'application doit être facile à maintenir et à mettre à jour. Le code doit être bien structuré et documenté.
- Une interface d'administration simple et claire doit permettre la gestion des données sans nécessiter de connaissances techniques.

5. Technologies Utilisées

• **Frontend :** HTML, JavaScript, CSS, React.js

• **Backend :** Java (Maven), API RESTful

• **Base de données** : MySQL (géré avec MySQL Workbench)

Authentification :

- JWT (JSON Web Tokens) pour la gestion sécurisée des sessions des utilisateurs.

6. Architecture du Système

L'architecture de l'application sera basée sur une architecture **client-serveur**.

Le frontend sera développé en html et communiquera avec le backend via des API RESTful. Le backend sera responsable de la gestion des utilisateurs, des paiements, des incidents, et de la base de données. Le frontend affichera les informations récupérées par les API et permettra à l'utilisateur d'interagir avec le système.

7. Plan de Développement

Le développement de l'application se déroulera en plusieurs phases :

1. **Phase de Planification** : Définition des exigences, création des diagrammes UML, élaboration du cahier des charges.
2. **Phase de Conception** : Création de l'architecture du système et des bases de données.
3. **Phase de Développement** : Développement du frontend, du backend, de l'intégration des paiements et de l'authentification.
4. **Phase de Test** : Tests unitaires, tests d'intégration, tests de performance.
5. **Phase de Déploiement** : Mise en ligne de l'application et suivi de sa mise en production.
6. **Phase de Maintenance** : Correction des bugs et ajout de nouvelles fonctionnalités en fonction des retours des utilisateurs.

Conclusion

L'application de gestion des résidences étudiantes vise à faciliter la gestion des chambres, des paiements, des incidents, et des étudiants. Elle permettra à l'administration de gagner du temps et d'améliorer l'efficacité des processus. De plus, les étudiants pourront accéder facilement aux informations et effectuer diverses actions (sélection de chambre, paiement, signalement d'incidents) de manière sécurisée et intuitive. Le respect des exigences fonctionnelles et non fonctionnelles assurera une expérience utilisateur optimale et une gestion centralisée et sécurisée des résidences étudiantes.

Chapitre II

Réalisation

Introduction

Ce chapitre détaille la mise en œuvre de notre application, en explorant les technologies utilisées, l'environnement de développement choisi, et la structure du projet mis en place. Nous allons plonger dans les détails techniques de la réalisation, en mettant l'accent sur les choix architecturaux et les bonnes pratiques de développement.

1. Technologies et Environnement de Développement

1.1. Technologies Utilisées

- **Backend : Spring Boot**

Figure 5 Spring boot

  - Spring Boot est le framework central de notre application. Il simplifie grandement la création d'applications Java autonomes, prêtes pour la production. Sa configuration par convention nous permet de nous concentrer sur la logique métier plutôt que sur les configurations complexes.
  - **Avantages :**
    - Configuration automatique et facile.
    - Intégration facile avec de nombreuses bibliothèques et bases de données.
    - Serveur embarqué (Tomcat par défaut) pour un déploiement facile.
    - Large communauté et abondance de ressources.
- **Base de Données : MySQL**

Figure 6 MySql

  - MySQL est notre choix pour la base de données relationnelle. Sa robustesse, sa popularité, et sa compatibilité avec Spring Boot en font un choix naturel.
  - **Avantages :**

        - Base de données open-source populaire et fiable.
        - Bonnes performances pour les applications web.
        - Facile à intégrer avec Spring Data JPA.
        - De nombreux outils de gestion sont disponibles.
- **Outil de Modélisation de Base de Données : MySQL Workbench**
    - MySQL Workbench est utilisé pour la conception et la gestion visuelle de la base de données. Il permet de modéliser les tables, définir les relations, et de générer les scripts SQL.
    - **Avantages :**
        - Interface graphique conviviale pour la conception de schémas de base de données.
        - Facilite la création de scripts SQL.
        - Possibilité de se connecter directement à la base de données pour effectuer des opérations.

1.2. Environnement de Développement

- **IDE : IntelliJ IDEA**

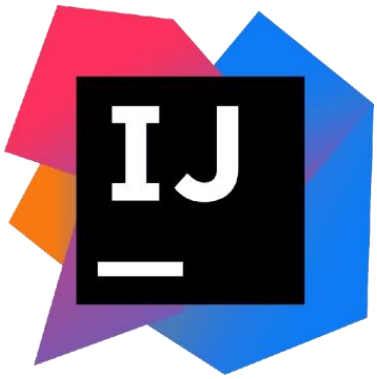

Figure 7 IntelliJ IDEA

    - IntelliJ IDEA est notre environnement de développement intégré. Il offre une multitude de fonctionnalités pour le développement en Java et Spring Boot.
    - **Avantages :**
        - Support complet de Spring Boot.
        - Débogage avancé.
        - Complétion intelligente du code.
        - Intégration avec les systèmes de contrôle de version.
        - Nombreux plugins pour faciliter le développement.

2. Structure du Projet

2.1. Organisation du Projet

Notre projet est organisé de manière à suivre les bonnes pratiques du développement Spring Boot et à faciliter la maintenance et l'évolution de l'application. La structure de base comprend les dossiers suivants :

- **src/main/java :** Contient le code source Java.
- **src/main/resources :** Contient les fichiers de configuration, les ressources statiques, etc.

2.2. Structure des Packages

Dans le dossier src/main/java, nous utilisons une structure de packages bien définie :

- **com.example.prj**: Il s'agit du package de base pour l'ensemble du projet. Il sert d'espace de noms pour organiser le code et éviter les conflits de noms.
- **com.example.prj.classes**: Ce dossier est utilisé pour les configurations et les autres classes d'utilitaires utilisés par le reste de l'application.
- **com.example.prj.config**: Ce dossier contient les classes de configuration de l'application.
- **com.example.prj.controller**: Ce dossier contient les classes de contrôleurs qui gèrent les requêtes web entrantes, les traitent et renvoient les réponses. Il agit comme un point d'entrée pour l'API.
- **com.example.prj.dto**: Ce dossier contient les Data Transfer Objects (DTO), qui sont des objets simplifiés utilisés pour transférer des données.
- **com.example.prj.entity**: Ce dossier contient les classes d'entités JPA qui représentent les tables de la base de données. Chaque entité correspond à une ligne dans la base de données.
- **com.example.prj.repository**: Ce dossier contient les repositories Spring Data JPA qui gèrent les interactions avec la base de données.
- **com.example.prj.security**: Ce dossier contient les classes relatives à la configuration de la sécurité.
- **com.example.prj.service**: Ce dossier contient les classes de services qui contiennent la logique métier de l'application.

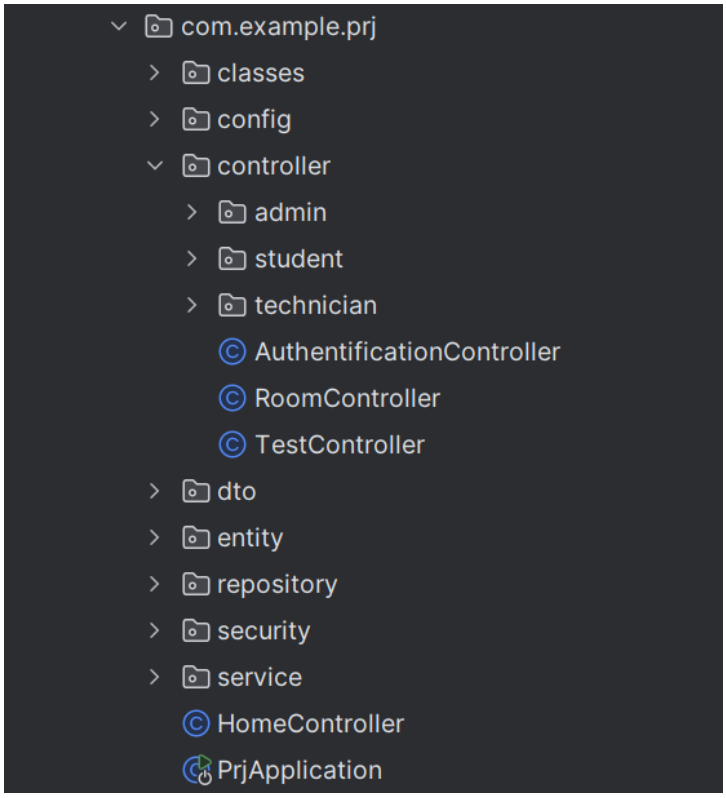

Figure 8 Structure de Projet : src/main/java

Décortiquons maintenant chaque classe que vous avez mentionnée, issues de différents packages.

com.example.prj.entity

- **Student**: Représente une entité étudiant dans la base de données. Elle contient probablement des données spécifiques à l'étudiant comme l'identifiant, le nom, l'email et tous autres attributs liés à l'étudiant.
- **Technician**: Représente une entité technicien dans la base de données. Elle contient probablement des données spécifiques au technicien comme l'identifiant, le nom, la spécialité et tous autres attributs liés au technicien.
- **Room**: Représente une entité chambre dans la base de données. Elle contient des informations sur la chambre, comme son identifiant, son numéro, sa capacité et toutes autres données connexes.
- **RoomStatus**: Un enum qui définit les valeurs d'état possibles pour une Room ("Disponible", "Occupée", "En Maintenance").
- **Maintenance**: Représente une demande ou un enregistrement de maintenance. Elle contient les données relatives à l'activité de maintenance, la chambre, le technicien assigné, les dates, l'état, la description, etc.
- **MaintenanceStatus**: Un enum définissant les états possibles d'un enregistrement de Maintenance (, "En Attente", "En Cours", "Terminée").
- **Payment**: Représente une transaction de paiement. Elle contient les détails du paiement, la date, le montant, l'étudiant associé et le statut du paiement.
- **PaymentStatus**: Un enum définissant les états possibles d'un Payment ("En Attente", "Payé").

com.example.prj.service

- **MaintenanceService**: Contient la logique métier pour la gestion des demandes de maintenance, comme la création, la mise à jour, la récupération et la gestion des transitions d'état de la maintenance.
- **PaymentService**: Contient la logique métier pour les opérations de paiement, telles que le traitement des paiements, la génération d'enregistrements de paiement et la récupération de l'historique des paiements.
- **PaymentReceiptService.java** : Cette classe est utilisée pour générer des rapports au format PDF.
- **RoomService**: Fournit la logique métier pour la gestion des chambres, comme la création, la mise à jour et l'extraction des données de chambre et potentiellement la gestion de l'allocation des chambres.
- **StudentService**: Contient la logique métier pour la gestion des étudiants, comme la création, la mise à jour et l'obtention d'informations sur l'étudiant.
- **TechnicianService**: Contient la logique métier pour la gestion des techniciens, comme l'ajout, la récupération et la mise à jour des données des techniciens.

com.example.prj.repository

- **MaintenanceRepository**: Une interface étendant JpaRepository de Spring Data JPA qui permet d'accéder à la base de données en relation avec l'entité Maintenance.
- **PaymentRepository**: Une interface étendant JpaRepository pour accéder aux données relatives à l'entité Payment.
- **RoomRepository**: Une interface étendant JpaRepository pour permettre des interactions avec la base de données pour l'entité Room.

- **StudentRepository**: Une interface étendant JpaRepository pour accéder à la base de données pour l'entité Student.
- **TechnicianRepository**: Une interface étendant JpaRepository pour interagir avec les données de l'entité Technician.

com.example.prj.controller

- **AuthenticationController**: Gère les opérations d'authentification et d'autorisation des utilisateurs comme la connexion et la déconnexion.
- **RoomController**: Gère les requêtes HTTP relatives aux chambres, comme la création, la récupération et la suppression de chambres.
- **TestController**: Il s'agit probablement d'un contrôleur simple pour tester ou expérimenter des fonctionnalités.
- **HomeController**: Le contrôleur principal qui sert généralement les requêtes qui n'entrent pas dans une catégorie de contrôleur spécifique, par exemple, le point d'entrée.
- **admin/AdminDashboardController**: Contrôleur pour gérer les requêtes relatives au tableau de bord principal de l'administrateur et toutes ses informations.
- **admin/AdminMaintenanceController**: Contrôleur pour gérer les requêtes relatives à la maintenance et à la gestion pour la section admin de l'application.
- **admin/AdminPaymentController**: Contrôleur pour gérer les requêtes relatives au paiement et à la gestion pour la section admin de l'application.
- **admin/AdminRoomController**: Contrôleur pour gérer les requêtes relatives à la chambre et à la gestion pour la section admin de l'application.
- **admin/AdminStudentController**: Contrôleur pour gérer les requêtes relatives à l'étudiant et à la gestion pour la section admin de l'application.
- **admin/AdminTechnicianController**: Contrôleur pour gérer les requêtes relatives au technicien et à la gestion pour la section admin de l'application.
- **student/IncidentController**: Contrôleur pour gérer les incidents signalés par les étudiants.
- **student/ResidentsController**: Contrôleur pour gérer les informations sur les résidents.

com.example.prj.dto

- **MaintenanceDTO**: DTO pour transférer les informations de maintenance entre les couches, simplifiant l'exposition des données.
- **MaintenanceReport**: DTO pour signaler les informations de maintenance.
- **PaymentDTO**: DTO pour transférer les informations relatives au paiement entre les couches.
- **RoomDTO**: DTO pour transférer les informations relatives à la chambre entre les couches.
- **StudentDTO**: DTO pour transférer les informations relatives à l'étudiant entre les couches.
- **TechnicianDTO**: DTO pour transférer les informations relatives au technicien entre les couches.

Principes de Conception Clés

- **Architecture en Couches**: Le projet suit une architecture en couches courante avec des contrôleurs, des services, des repositories et des entités, ce qui favorise une meilleure organisation du code.
- **Séparation des Préoccupations**: Chaque couche est responsable d'un aspect spécifique, réduisant le couplage du code.
- **DTOs**: Les DTOs aident au transfert de données et garantissent que seules les données nécessaires sont exposées entre les différentes parties de l'application.

2.3. Fichier pom.xml

Le fichier pom.xml est le descripteur du projet Maven. Il définit les dépendances du projet, les plugins, et les configurations de construction.

Structure Générale du pom.xml

- **<project>**: La racine du fichier pom.xml.
- **<modelVersion>**: Indique la version du modèle du POM utilisé (toujours 4.0.0).
- **<parent>**: Définit le POM parent dont hérite ce projet, ici spring-boot-starter-parent.
- **<groupId>, <artifactId>, <version>**: Identifient de manière unique le projet.
- **<name>, <description>, <url>**: Informations descriptives sur le projet.
- **<properties>**: Définit des variables (ici, la version de Java).
- **<dependencies>**: Liste des librairies et modules requis par le projet.
- **<build>**: Configurations pour le processus de construction du projet.

Analyse des Dépendances <dependencies>

1. **spring-boot-starter-data-jpa**

    - **groupId** : org.springframework.boot
    - **artifactId** : spring-boot-starter-data-jpa
    - **Explication** : Cette dépendance est cruciale pour la gestion de la persistance des données. Elle intègre Spring Data JPA, un module de Spring qui simplifie l'interaction avec les bases de données relationnelles. Elle fournit une couche d'abstraction pour les opérations CRUD (Create, Read, Update, Delete) sur les entités JPA. En d'autres termes, cette dépendance permet de gérer la base de données avec moins de code que l'écriture de requêtes SQL à la main.

2. **spring-boot-starter-security**

    - **groupId** : org.springframework.boot
    - **artifactId** : spring-boot-starter-security
    - **Explication** : Cette dépendance ajoute la sécurité à votre application. Spring Security est un puissant framework de sécurité qui permet d'authentifier les utilisateurs, d'appliquer des contrôles d'accès (autorisation) et de protéger les ressources de votre application contre les accès non autorisés. Elle est essentielle pour toute application qui gère des données sensibles ou nécessite un accès contrôlé.

3. **spring-boot-starter-validation**

    - **groupId** : org.springframework.boot
    - **artifactId** : spring-boot-starter-validation
    - **Explication** : Cette dépendance permet de valider les données qui entrent dans votre application. Avec cette dépendance, vous pouvez utiliser des annotations pour spécifier les contraintes de vos données (par exemple, un champ ne peut pas être vide, un email doit être valide, etc.). Cela simplifie la validation des données et permet d'éviter des erreurs plus tard dans le processus.

4. **spring-boot-starter-web**

    - **groupId** : org.springframework.boot
    - **artifactId** : spring-boot-starter-web
    - **Explication** : C'est la dépendance fondamentale pour créer des applications web avec Spring MVC. Elle inclut les dépendances nécessaires pour créer des contrôleurs, gérer les requêtes HTTP, utiliser les formats JSON et XML, etc. C'est le point de départ pour toute application Spring Boot qui doit interagir avec un navigateur web.

5. **spring-boot-devtools**

    - **groupId** : org.springframework.boot
    - **artifactId** : spring-boot-devtools
    - **Explication** : Cette dépendance est un ensemble d'outils de développement qui améliorent l'expérience de développement avec Spring Boot. Elle permet notamment le redémarrage automatique de l'application lors de la modification du code, ce qui est très utile lors du développement. Le scope runtime signifie que la dépendance n'est nécessaire qu'au moment de l'exécution. L'option optional indique que cette dépendance est facultative et ne doit pas être transitive dans d'autres modules.

6. **mysql-connector-j**

    - **groupId**: com.mysql
    - **artifactId**: mysql-connector-j
    - **Explication** : Cette dépendance est le driver JDBC qui permet de connecter votre application Spring Boot à une base de données MySQL. Le scope runtime indique qu'il n'est nécessaire que lors de l'exécution et non durant la compilation. Ce driver permet d'exécuter des requêtes SQL sur une base de données MySQL depuis votre code Java.

7. **lombok**

    - **groupId** : org.projectlombok
    - **artifactId** : lombok
    - **Explication** : Lombok est une librairie qui réduit le code répétitif en générant automatiquement des getters, setters, constructeurs et autres méthodes communes lors de la compilation via annotations. Cela rend votre code Java plus concis et facile à lire. L'option optional indique qu'il ne doit pas être transitif vers d'autres modules qui utilisent ce projet.

8. **spring-boot-starter-test**

    - **groupId** : org.springframework.boot
    - **artifactId** : spring-boot-starter-test
    - **Explication** : Cette dépendance fournit un ensemble d'outils pour écrire des tests unitaires et des tests d'intégration pour votre application Spring Boot. Elle inclut JUnit, Mockito, etc., ce qui facilite le processus de test. Le scope test indique qu'elle est utilisée uniquement lors de la phase de test du projet.

9. **spring-security-test**

- **groupId** : org.springframework.security
- **artifactId** : spring-security-test
- **Explication** : Cette dépendance fournit des outils spécifiques pour tester la sécurité de votre application Spring Security. Elle permet de simuler des authentifications et des autorisations pour vos tests. Encore une fois, elle est utilisée uniquement lors de la phase de test.

10. **jjwt-api, jjwt-impl, jjwt-jackson**

- **groupId** : io.jsonwebtoken
- **artifactId** : jjwt-api, jjwt-impl, jjwt-jackson
- **Explication** : Ces trois dépendances sont celles qui permettent de gérer les JSON Web Tokens (JWT). Elles sont utilisées pour la création, la signature et la vérification des tokens JWT, qui sont souvent utilisés pour l'authentification et l'autorisation dans les applications web.

11. **spring-boot-starter-thymeleaf**

- **groupId** : org.springframework.boot
- **artifactId** : spring-boot-starter-thymeleaf
- **Explication** : Cette dépendance permet d'utiliser le moteur de templates Thymeleaf pour générer des vues web. Elle est utile si vous avez besoin de créer des pages HTML dynamiquement.

12. **javax.servlet-api**

- **groupId** : javax.servlet
- **artifactId** : javax.servlet-api
- **Explication** : fournit une API pour travailler avec des servlets, composant essentiel du développement web en Java, la spécification Java EE, mais dans les projets Spring boot elle est souvent déjà fournis dans le starter spring-boot-starter-web et elle est mis provided ici, pour dire que le serveur d'application ou le conteneur servlet l'a déjà mis a disposition.

2.4. Vue Générale

L'architecture de notre application est une architecture multicouche, où chaque couche a une responsabilité bien définie :

- **Couche Contrôleur :** Reçoit les requêtes HTTP, valide les données, et délègue le traitement aux services.
- **Couche Service :** Encapsule la logique métier, fait l'appel aux repositories, et renvoie les données au contrôleur.
- **Couche Repository :** Interagit avec la base de données en utilisant Spring Data JPA pour effectuer les opérations CRUD (Create, Read, Update, Delete).
- **Couche Entité :** Représente les tables de la base de données sous forme de classes Java.
- **Couche DTO :** Transfère les données entre le frontend et le backend.

Conclusion

Ce chapitre a présenté les aspects techniques clés de la mise en œuvre de l'application "ResidenceManager". Le choix des technologies, notamment Spring Boot, MySQL, et IntelliJ IDEA, a permis de créer un environnement de développement robuste et efficace. La structure du projet, organisée en packages clairs et respectant les bonnes pratiques de Spring Boot, facilite la maintenance et l'évolution de l'application.

L'analyse des dépendances du fichier pom.xml a révélé les composants essentiels du projet, tels que Spring Data JPA pour l'accès aux données, Spring Security pour la sécurité, et les bibliothèques pour la gestion des JWT et du web. L'architecture multicouche adoptée permet de séparer les préoccupations et de garantir une meilleure organisation du code.

Chapitre III

Vue de l’application

Introduction

Ce chapitre documente l'interface utilisateur de l'application "ResidenceManager", conçue pour la gestion de résidences étudiantes. L'objectif est de présenter les fonctionnalités clés mises en œuvre pour optimiser la gestion des résidences et faciliter l'expérience des utilisateurs, tant les étudiants que les administrateurs. L'application offre un ensemble d'outils couvrant différents aspects, depuis la consultation des disponibilités de chambres jusqu'au suivi des paiements, en passant par la gestion des incidents et la gestion des différentes entités du système. Les pages suivantes détaillent ces fonctionnalités.

Descriptions des Pages

1. Page d'Accueil

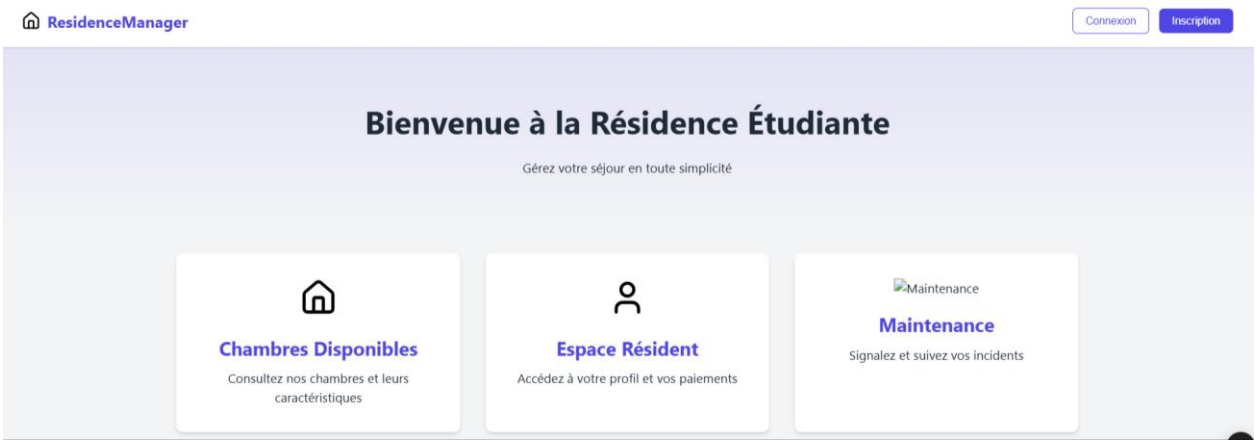

Figure 9 Page d'Accueil

La page d'accueil de "ResidenceManager" constitue le point d'entrée de l'application. Elle présente les principales catégories de fonctionnalités sous forme de liens : consultation des chambres disponibles, accès à l'espace personnel de l'étudiant, et signalement d'incidents. Cette page offre également les options de connexion et d'inscription pour accéder aux fonctionnalités personnalisées de l'application. L'objectif de cette page est de fournir un point d'accès simple et direct aux différentes sections de l'application.

2. Page "Chambres Disponibles"

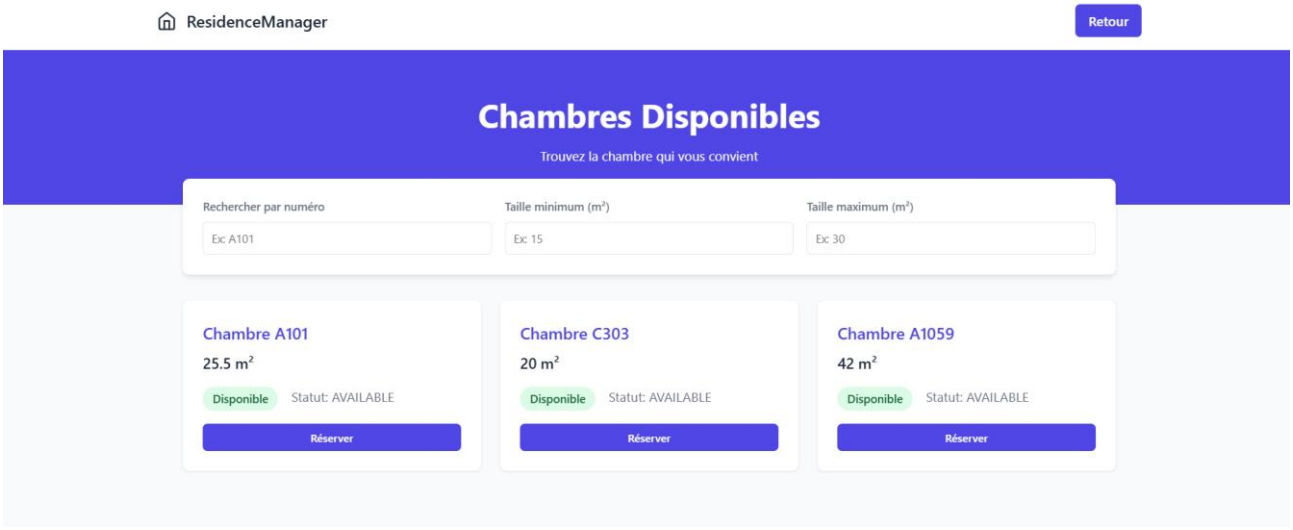

Figure 10 Page des chambres disponibles

La page "Chambres Disponibles" offre une interface de recherche pour la consultation des chambres. Les étudiants peuvent filtrer les chambres par numéro et par taille. Chaque résultat affiche le numéro de la chambre, sa taille et son statut de disponibilité. Un bouton de réservation est disponible pour chaque chambre afin de permettre aux utilisateurs de choisir et sélectionner une chambre rapidement.

3. Page "Espace Résident"

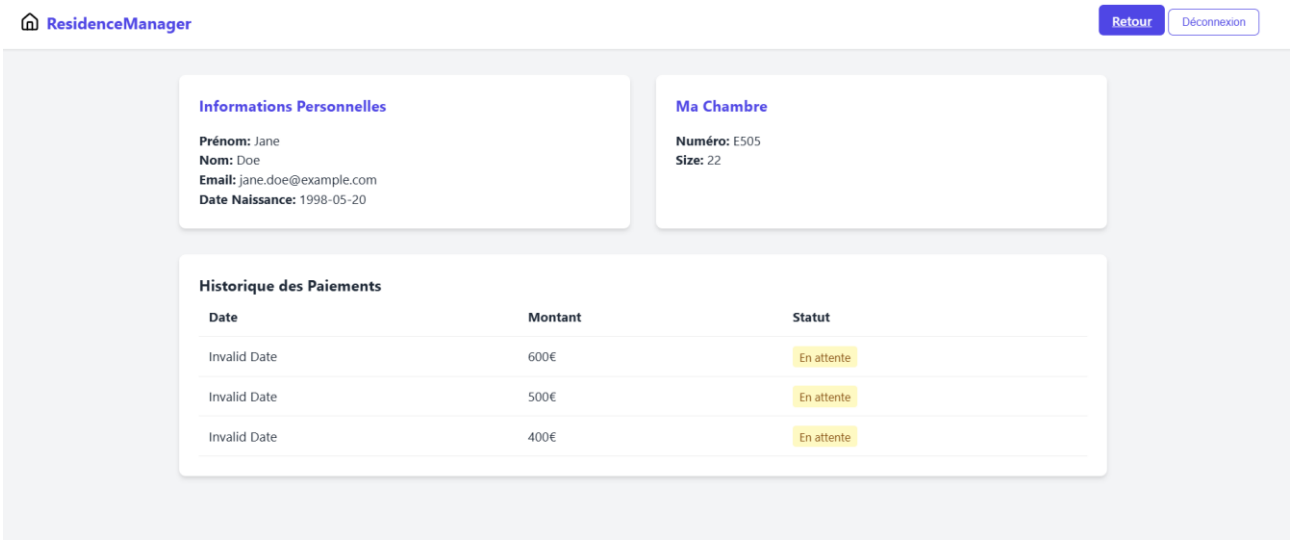

Figure 11 Page de l'espace résident

La page "Espace Résident" fournit un accès aux informations personnelles de l'étudiant, incluant nom, adresse email et date de naissance, ainsi qu'aux informations sur la chambre assignée, telles que son numéro et sa taille. Un tableau récapitulatif des paiements effectués, incluant la date, le montant et le statut, est également présenté dans cette section. Cette page a pour fonction de consolider les informations propres à l'étudiant. Des options de navigation sont disponibles en haut de la page pour un retour à l'accueil et la déconnexion.

4. Page "Maintenance"

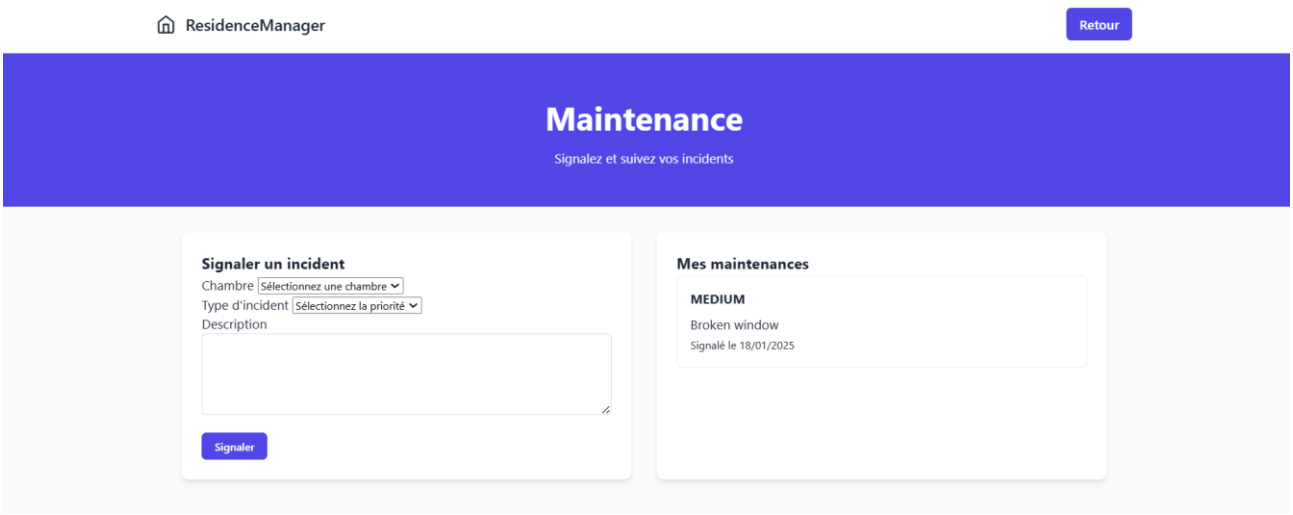

Figure 12 Page de maintenance

La page "Maintenance" permet aux étudiants de signaler des incidents concernant leur chambre ou les espaces communs. Un formulaire est disponible pour renseigner la chambre concernée, le type d'incident, et une description détaillée de l'incident. Un historique des incidents signalés est également visible pour l'utilisateur, affichant l'état de chaque signalement ainsi que la date. La fonction de cette page est de centraliser et de faciliter la gestion des incidents pour l'étudiant.

5. Page "Gestion des Étudiants" (Admin)

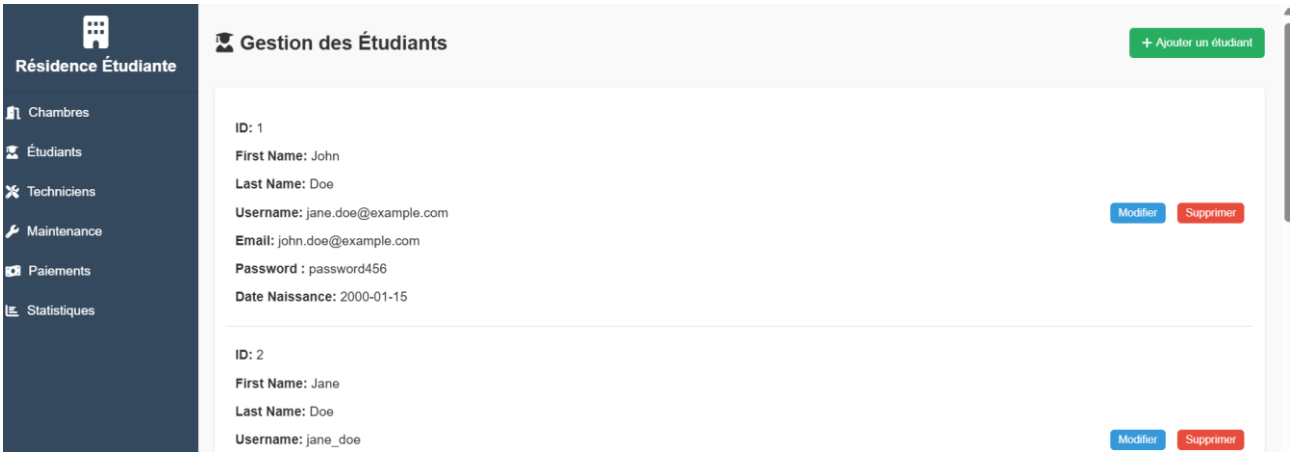

Figure 13 Interface admin

La page "Gestion des Étudiants", accessible aux administrateurs, permet la gestion de différentes tables de données au sein de l'application. Elle affiche une liste des étudiants avec leurs ID, nom, prénom, nom d'utilisateur, email, mot de passe et date de naissance. Elle propose des actions de modification et de suppression pour chaque utilisateur, ainsi qu'un bouton d'ajout de nouveaux étudiants. Cette page est la première d'une série d'outils visant à fournir un accès centralisé pour la gestion administrative des différentes entités telles que les étudiants, les techniciens, les paiements, etc.

Conclusion

Ce projet a permis le développement d'une application web complète pour la gestion de résidences étudiantes, répondant aux besoins des étudiants et de l'administration. L'application "ResidenceManager" centralise et automatise les processus de gestion des chambres, des inscriptions d'étudiants, des paiements et des signalements d'incidents, tout en offrant une interface intuitive et accessible.

Les choix technologiques, notamment l'utilisation de Spring Boot pour le backend, MySQL pour la base de données, ainsi que l'architecture en couches, ont permis de créer une application robuste, évolutive et facile à maintenir. Les diagrammes UML ont servi de guide pour structurer le développement, tandis que le cahier des charges a permis de définir précisément les exigences et les objectifs du projet.

L'application présente une interface conviviale pour les étudiants, facilitant la consultation des chambres, la gestion de leurs profils et de leurs paiements, ainsi que le signalement d'incidents. Côté administration, "ResidenceManager" offre des outils performants pour la gestion des chambres, des étudiants, des paiements, des incidents, et pour la production de statistiques, permettant une gestion plus efficace des résidences.

En conclusion, ce projet a permis de mettre en œuvre une solution répondant aux problématiques de la gestion de résidences étudiantes, en combinant des technologies modernes, une conception rigoureuse et une approche centrée sur l'utilisateur. Il constitue une base solide pour des développements ultérieurs.