1. Introduction

Dans le cadre de ce projet de gestion du bureau de courrier, j’ai été en charge du développement de la partie back-end de l’application en utilisant **Spring Boot**, un Framework Java moderne et léger qui permet de créer rapidement des services REST performants, tout en assurant la configuration automatique des composants nécessaires. Il facilite le développement d'applications web complètes en Java, avec une architecture claire basée sur les couches Controller, Service et Repository.

1. Objectives

Assurer la communication fluide avec le Frontend en fournissant des Endpoints bien définis et structurés.

Organisation du projet en couches (Controller, Service, Repository).

Production des REST APIs sécurisée pour gérer les différents requêtes HTTP (GET / POST / DELETE…) via des endpoints REST.

Implémenter les opérations CRUD sur les entités principales et surtout Courrier

Gestion des employées, les roles, les services, les courriers…

Création et gestion de la base de données

1. Technologies utilisées

Spring Boot : Framework principal pour créer l'API REST

Spring Data JPA : Abstraction pour manipuler la BDD avec des interfaces simples

Hibernate : ORM qui mappe les entités Java aux tables de la BDD

MySQL : Système de gestion de base de données relationnelle

Postman / Swagger : Test et documentation des endpoints

Lombok : Génère automatiquement les getters/setters/constructeurs

1. Modelisation

Premièrement pour le diagramme de classe après analyse du projet et de cahier de charge, le diagramme résultats et le suivant

\*

\*

\*

\*\*

\*

\*

\*

\*

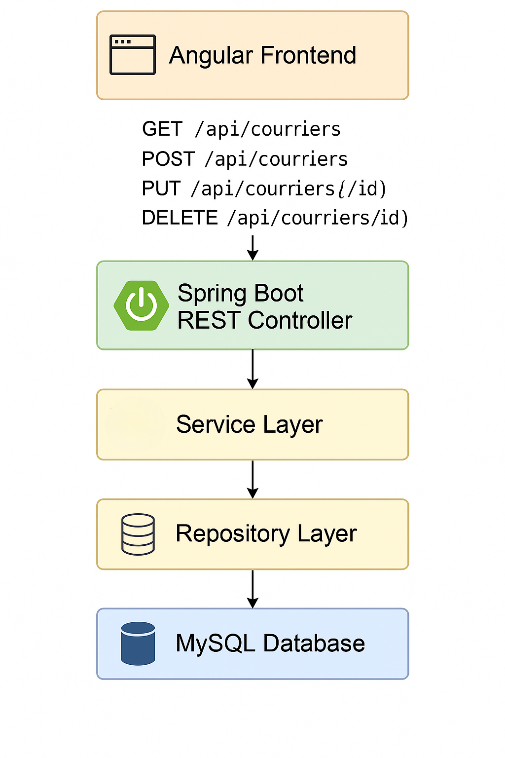
Avec la table principale ce qui est table courrier avec la modélisation du processus et objectif du projet

Le modèle de données repose sur plusieurs entités principales, notamment :

* Courrier : représente les documents envoyés ou reçus.
* Utilisateur : représente un employé connecté à un service.
* Service : représente une direction ou unité administrative.
* TypeCourrier : catégorise les courriers.

1. Implémentation des fonctionnalités

Apres être documenté a propos du Springboot et son algorithmes de travail donc J’ai créé les endpoints nécessaires pour la gestion complète des courriers (ajout, suppression, édition, affichage). Par exemple, le contrôleur CourrierController contient des méthodes comme @PostMapping("/courriers") pour l’ajout et @GetMapping("/courriers") pour l’affichage. Chaque appel est délégué à un service métier CourrierService, qui traite la logique puis interagit avec la base via CourrierRepository.



1. Test et Validation

Pour la partie de test on a décide d’aller avec swagger donc une fois les endpoints définis, leur bon fonctionnement a été validé via Swagger. Chaque route a été testée avec différents cas (données valides, erreurs, etc.). Par exemple, la requête GET pour avoir les données qui seront affichées dans le tableau du bord.

A screenshot of a computer program

AI-generated content may be incorrect.

**Problèmes rencontrés et solutions apportées**

Lors du développement back-end avec Spring Boot, plusieurs problèmes ont été rencontrés et résolus :

* **Erreur CORS entre Angular et Spring Boot** : empêchant les requêtes HTTP du front-end.  
  ➤ *Solution* : ajout de @CrossOrigin dans les contrôleurs et configuration du fichier de sécurité.
* **Échec de connexion à la base de données** : causé par des erreurs dans application.properties.  
  ➤ *Solution* : vérification des identifiants et ajout de spring.jpa.hibernate.ddl-auto=update.
* **Réponses d’erreur non explicites (500 au lieu de 400/404)**.  
  ➤ *Solution* : création de gestionnaires d’erreurs avec @ControllerAdvice et @ExceptionHandler