UNIVERSITE ABDELMALEK ESSAADI

FACULTE DES SCIENCES ET TECHNIQUES DE TANGER

Rapport

Exam Management Application Technical Report

MASTER SIT&BIGDATA

Réalisé par : DOUMI SALMA

Pr. AZIZ MAHBOUB

Année universitaire 2024-2025

Exam Management Application Technical Report

1. Introduction

This report provides a detailed overview of the Exam Management Application developed using a modern tech stack with Spring Boot backend and JSP/Ajax/Bootstrap frontend. The application serves as a comprehensive platform for managing educational resources, user accounts, and exam scheduling for an educational institution.

Exam Management App est une application web conçue pour simplifier la gestion des données et optimiser les processus liés au personnel éducatif et à l'organisation des examens. Grâce à l'intégration des frameworks Spring Boot, MVC, JPA et Security, l'application garantit une performance robuste et une interface utilisateur conviviale.

➢ Objectifs du Projet

Ce projet vise à :

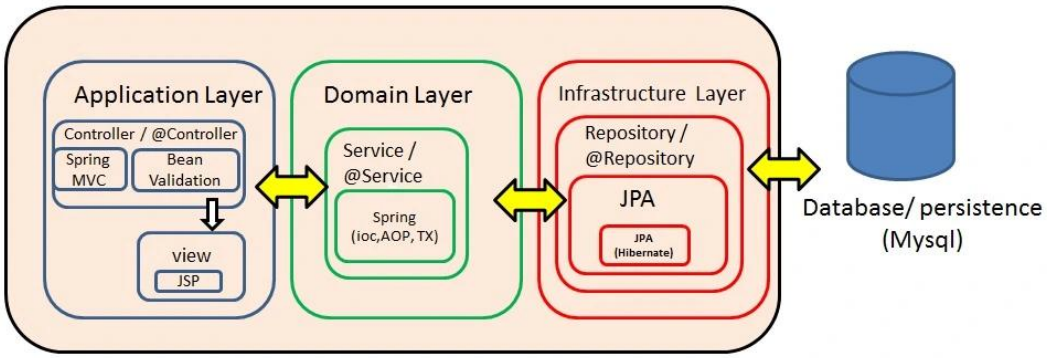
- Gérer le personnel éducatif : ajout, modification, suppression et consultation des enseignants et administrateurs.
- Organiser les examens : création et planification avec affectation des équipes responsables.
- Offrir une expérience utilisateur intuitive avec une navigation fluide et personnalisable.

2. System Architecture

2.1 Overall Architecture

The application follows a classic 3-tier architecture:

- **Presentation Layer**: JSP pages with Bootstrap for responsive design and Ajax for asynchronous communications
- **Business Logic Layer**: Spring Boot with MVC pattern
- **Data Access Layer**: Spring Data JPA for database interactions with PostgreSQL



The architecture adheres to the following pattern:

- **Controller Layer**: Handles HTTP requests and responses
- **Service Layer**: Contains business logic
- **Repository Layer**: Manages data persistence
- **Entity Layer**: Represents database tables as Java objects

2.2 Spring Framework Implementation

- **Spring Boot**: Provides auto-configuration and an embedded server
- **Spring MVC**: Implements the Model-View-Controller pattern
- **Spring Security**: Manages authentication and authorization with JWT
- **Spring Data JPA**: Simplifies data access with Hibernate as the ORM

2.3 Database Design

The PostgreSQL database schema includes the following core entities:

- **Personne**: Base entity for all users
- **Role**: Defines user roles (Admin, CadreAdmin, Professeur, Etudiant)
- **Compte**: User account information
- **Département**: Academic departments
- **ElementPedag**: Educational elements (courses, modules)
- **Examen**: Exam information
- **Filiere**: Study tracks/majors
- **Groupe**: Student groups
- **Niveau**: Academic levels
- **Salle**: Classrooms for exams
- **Surveillance**: Exam supervision details
- **TypeElement**: Types of educational elements

```
postgres=# \c plannig_exam
You are now connected to database "plannig_exam" as user "postgres".
plannig_exam=# \dt
                List of relations
 Schema |        Name         | Type  |  Owner
--------+---------------------+-------+----------
 public | cadreadministrateur | table | postgres
 public | compte              | table | postgres
 public | departement         | table | postgres
 public | elementpedagogique  | table | postgres
 public | enseignant          | table | postgres
 public | etudiant            | table | postgres
 public | examen              | table | postgres
 public | filiere             | table | postgres
 public | groupe              | table | postgres
 public | niveau              | table | postgres
 public | personne            | table | postgres
 public | role                | table | postgres
 public | salle               | table | postgres
 public | surveillance        | table | postgres
 public | typeelement         | table | postgres
(15 rows)
```

- **Fonctionnalités**

A. Gestion du Personnel Éducatif

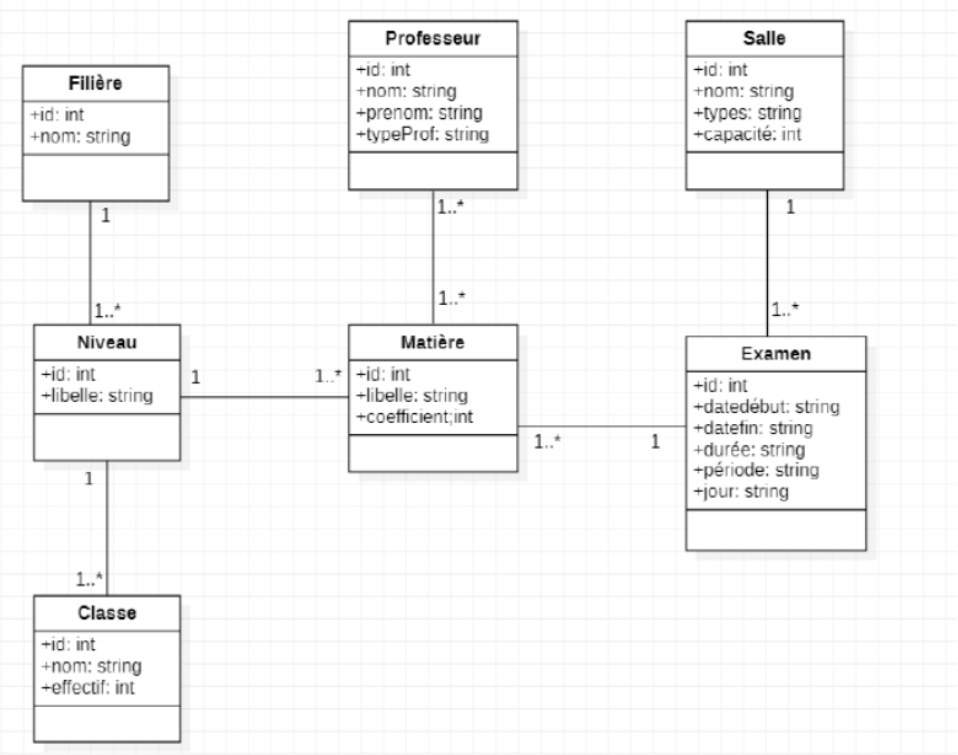
- **Ajout de personnel** : informations essentielles (nom, ID, département, etc.).
- **Modification et suppression** : gestion des membres existants.
- **Affichage personnalisé** : tri et filtrage des informations.

B. Gestion des Examens

- **Création d'examens** : détails (matière, date, heure, durée).
- **Affectation** : assignation des classes et du personnel.
- **Vue d'ensemble** : planification et récapitulatif des examens.

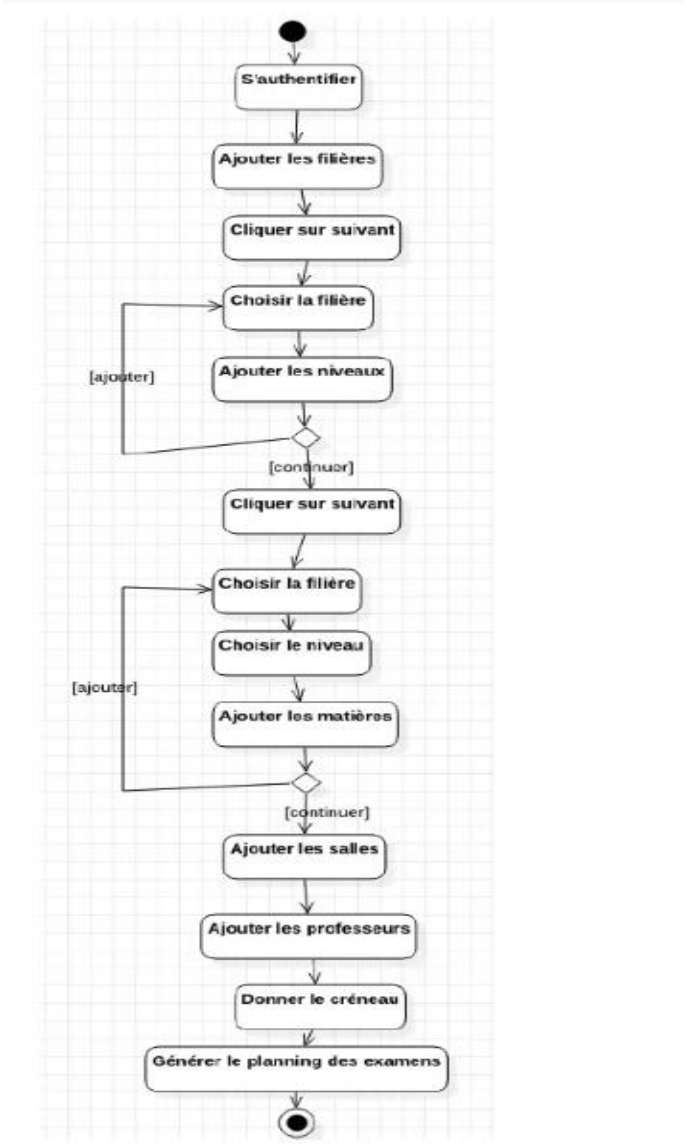
C. Authentification et Autorisation

- Contrôle d'accès pour les rôles : "Admin", "Cadre Admin", "Professeur" et "Étudiant".



Cas d'utilisation

- **Acteurs principaux :** Administrateur, Enseignant.
- **Scénarios :**
  - ✓ Ajouter un membre du personnel.(CRUD complet)
  - ✓ Créer un examen.
  - ✓ Consulter les examens planifiés.
  - ✓ Gérer les classes et leurs affectations.



3. User Roles and Functionalities

The application implements four distinct user roles:

3.1 Administrator (Administrateur)

- Complete system management
- User account creation and management
- Department and curriculum management
- System-wide configuration

3.2 Administrative Staff (Cadre Admin)

- Student management
- Limited user account management
- Reporting capabilities
- Support for administrative processes

3.3 Professor (Professeur)

- Course and educational element management
- Exam creation and scheduling
- Student progress tracking
- Grade management

3.4 Student (Etudiant)

- Exam schedule viewing
- Personal information management
- Course enrollment
- Grade viewing

4. Key Features

4.1 User Management

- Account creation and management
- Role-based access control
- Password encryption with BCrypt
- JWT token authentication

4.2 Educational Element Management

- Course and module creation
- Assignment of professors to courses
- Level and curriculum organization
- Educational resource management

4.3 Exam Management

- Comprehensive exam scheduling
- Multiple exam types support
- Exam room allocation
- Supervision management

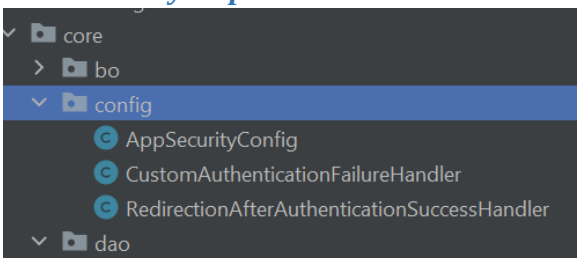
4.4 Dashboard and Analytics

- User statistics visualization
- Student performance tracking
- User activity monitoring
- Data-driven insights

5. Technical Implementation Details

5.1 Backend Implementation

5.1.1 Security Implementation



@Configuration

```
no usages
@Configuration
@EnableWebSecurity // Car c'est notre classe de gestion de sécurité donc on active Spring Security
public class AppSecurityConfig {
    1 usage
    Logger LOGGER = LoggerFactory.getLogger(getClass().getName());
    3 usages
    @Autowired // injection du gestionnaire CustomAuthentificationService
    private CustomAuthentificationService userService;
    1 usage
    @Bean
    public AuthenticationSuccessHandler uthenticationSuccessHandler() {
        return new RedirectionAfterAuthenticationSuccessHandler();
    }
    no usages
    public AppSecurityConfig(CustomAuthentificationService userService) {
        this.userService = userService;

        LOGGER.debug("AppSecurityConfig Initialisé");
    }
    no usages
    @Bean
    public DaoAuthenticationProvider authProvider() {
```

```
    @Bean                                                                    7
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        // TODO : Configurer la securité de votre application
        http
        //.csrf(csrf -> csrf.disable())  // Disable CSRF
        .authorizeHttpRequests(
                authz -> authz.requestMatchers( ...patterns: "/student/**") AuthorizedUrl
                .hasRole("STUDENT") //les mappings /student/** sont accessibles uniquementpour ROLE STUDENT
                .requestMatchers( ...patterns: "/cadreadmin/**")
                .hasRole("CADRE_ADMIN") AuthorizationManagerRequestMat...
                .requestMatchers( ...patterns: "/prof/**") AuthorizedUrl
                .hasRole("PROF") AuthorizationManagerRequestMat...
                .requestMatchers( ...patterns: "/admin/**") AuthorizedUrl
                .hasRole("ADMIN") AuthorizationManagerRequestMat...
                .anyRequest().permitAll()

        ).formLogin(form -> form.loginPage("/showMyLoginPage")
                .loginProcessingUrl("/authenticateTheUser")
                .failureHandler(authenticationFailureHandler())
                .successHandler(uthenticationSuccessHandler())
                .permitAll()

        ).logout(logout -> logout.logoutUrl("/logout") // Endpoint de déconnexion
```

5.1.2 Entity Relationships

```java
Personne.java   CustomAuthenticationFailureHandler.java   RedirectionAfterAuthent
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import java.util.*;
import jakarta.persistence.*;
3 inheritors
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Personne {
    3 usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idPersonne;
    3 usages
    private String nom;
    3 usages
    private String prenom;
    3 usages
    @Column(unique = true)
    private String cin;
    3 usages
    private String email;
```

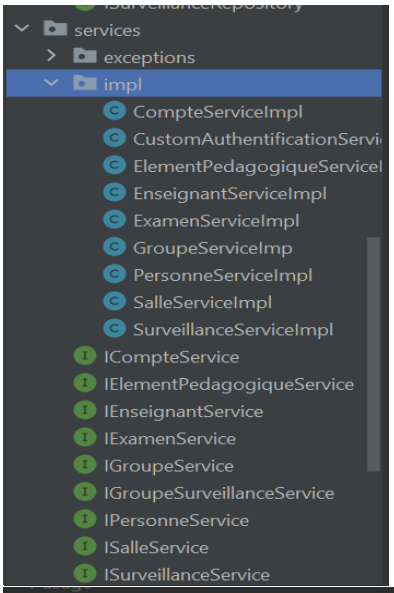
```java
@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "personnes")
private Groupe groupe ;
2 usages
@OneToMany(mappedBy = "proprietaire", cascade = CascadeType.ALL, targetEntity = Compte.class)
private Set<Compte> comptes;

public Long getIdPersonne() { return idPersonne; }

public void setIdPersonne(Long idPerson) { this.idPersonne = idPerson; }

public String getNom() { return nom; }

public void setNom(String nom) { this.nom = nom; }

public String getPrenom() { return prenom; }

public void setPrenom(String prenom) { this.prenom = prenom; }

public String getCin() { return cin; }

public void setCin(String cin) { this.cin = cin; }
```

5.1.3 Repository Layer

```
dao
  impl
  DepartementRepository
  ElementPedagogiqueRepository
  EnseignantRepository
  FiliereRepository
  GroupeRepository
  ICompteRepository
  IExamenRepository
  IPersonneRepository
  IRepositoryDao
  ISalleRepository
  ISurveillanceRepository
services
```

```java
public interface ICompteRepository extends JpaRepository<Compte, Long> {
    4 usages
    public Compte getCompteByLogin(String username);

    no usages
    boolean existsByProprietaireIdPersonne(Long personId);
    no usages
    Compte findByLogin(String login);
}
```

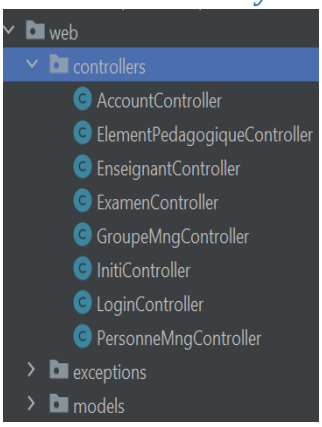
5.1.4 Service Layer




```java
@Service
@Transactional
public class CompteServiceImpl implements ICompteService {
    6 usages
    @Autowired
    private ICompteRepository userDao;
    2 usages
    @Autowired
    private IRepositoryDao roleDao;
    1 usage
    @Autowired
    private IPersonneRepository personDao;
    1 usage
    @Autowired
    private PasswordEncoder passwordEncoder;
    1 usage
    public List<Role> getAllRoles() { return roleDao.findAll(); }
    4 usages
    public List<Compte> getAllAccounts() { return userDao.findAll(); }
    1 usage
    public String createUser(Long idRole, Long idPerson) {Personne person = personD
        Compte userAccount = new Compte();
        userAccount.setProprietaire(person);
```

```java
public String generatePassayPassword() {
    CharacterRule digits = new CharacterRule(EnglishCharacterData.Digit);

    PasswordGenerator passwordGenerator = new PasswordGenerator();
    String password = passwordGenerator.generatePassword( length: 10, digits);

    return password;
```

5.1.5 Controller Layer




```java
@Controller
public class LoginController {
    2 usages
    @Autowired
    private HttpSession httpSession;
    5 usages
    private UserAndAccountInfos getUserAccount() {
        UserAndAccountInfos userInfo = (UserAndAccountInfos) httpSession.getAttribute( s: "userInfo");
        if (userInfo == null) {
            Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            Compte userAccount = ((UserPrincipal) principal).getUser();
            Personne u = userAccount.getProprietaire();
            String roleName = userAccount.getRole().getNomRole();
            userInfo = new UserAndAccountInfos(u.getIdPersonne(), userAccount.getIdCompte(),
                    userAccount.getLogin(),
                    u.getNom(), u.getPrenom(), u.getEmail(), roleName);
            httpSession.setAttribute( s: "userInfo", userInfo);
        }
        return userInfo;

    }
```

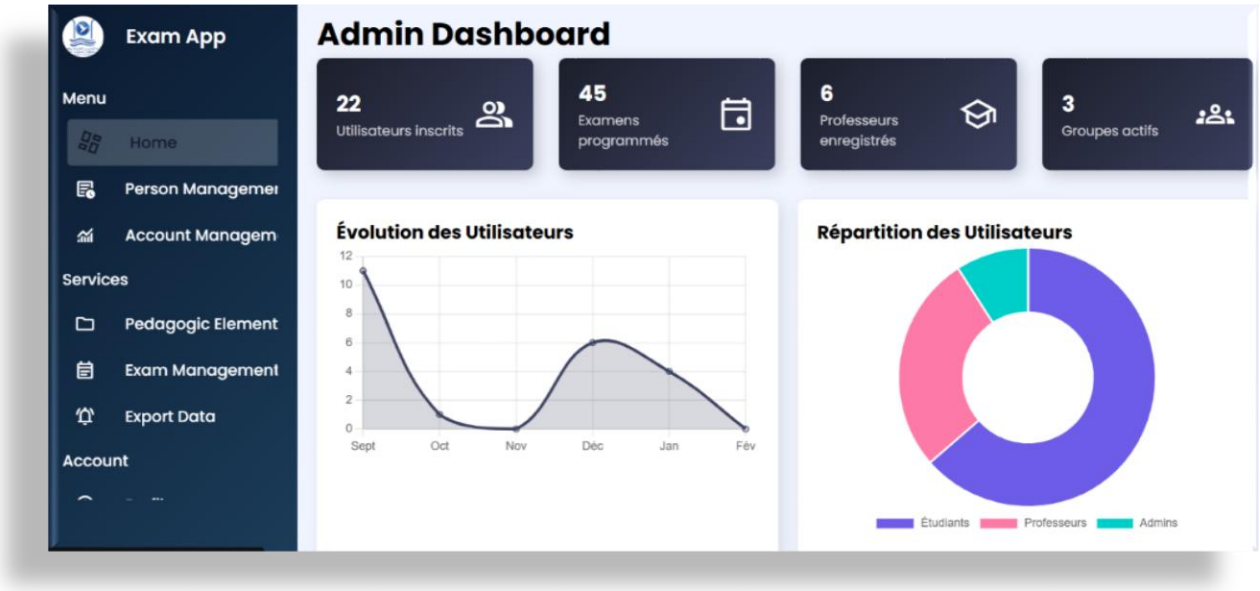
```java
@Controller
@RequestMapping("/admin")
public class AccountController {
    7 usages
    @Autowired
    private ICompteService userService; // On obtient par injection automatique le service
    1 usage
    @Autowired
    private IPersonneService personService; // On obtient par injection automatique le service
    no usages
    @RequestMapping(value = "createAccountForm/{idPerson}", method = RequestMethod.GET)
    public String createAccountForm(@PathVariable int idPerson, Model model) {
        AccountModel accountModel = new AccountModel(Long.valueOf(idPerson));
        model.addAttribute( attributeName: "accountModel", accountModel);
        model.addAttribute( attributeName: "roleList", userService.getAllRoles());
        model.addAttribute( attributeName: "accountList", userService.getAllAccounts());
        return "admin/formAccount";}
    no usages
    @GetMapping("manageAccounts")
    public String manageAccounts(@ModelAttribute("accountModel") AccountModel accountModel, Model model) {

        model.addAttribute( attributeName: "accountList", userService.getAllAccounts());
```

5.2 Frontend Implementation

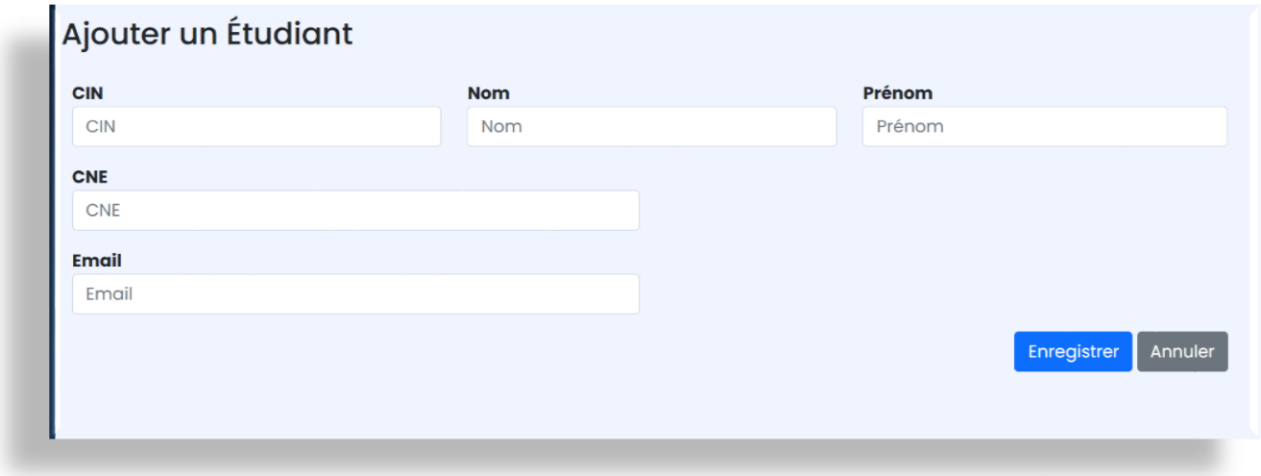
5.2.1 JSP and Bootstrap

The frontend utilizes JSP pages enhanced with Bootstrap for responsive design. Key components include:

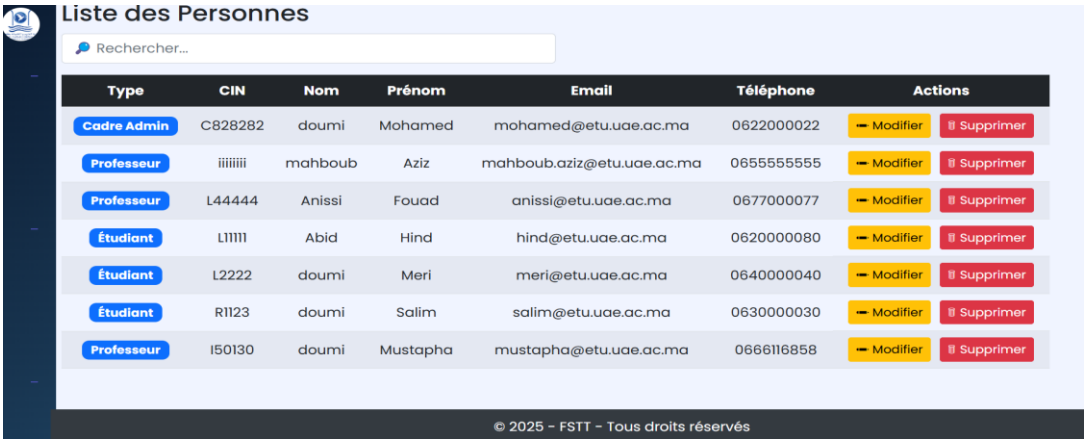
- Admin dashboard with statistics visualization

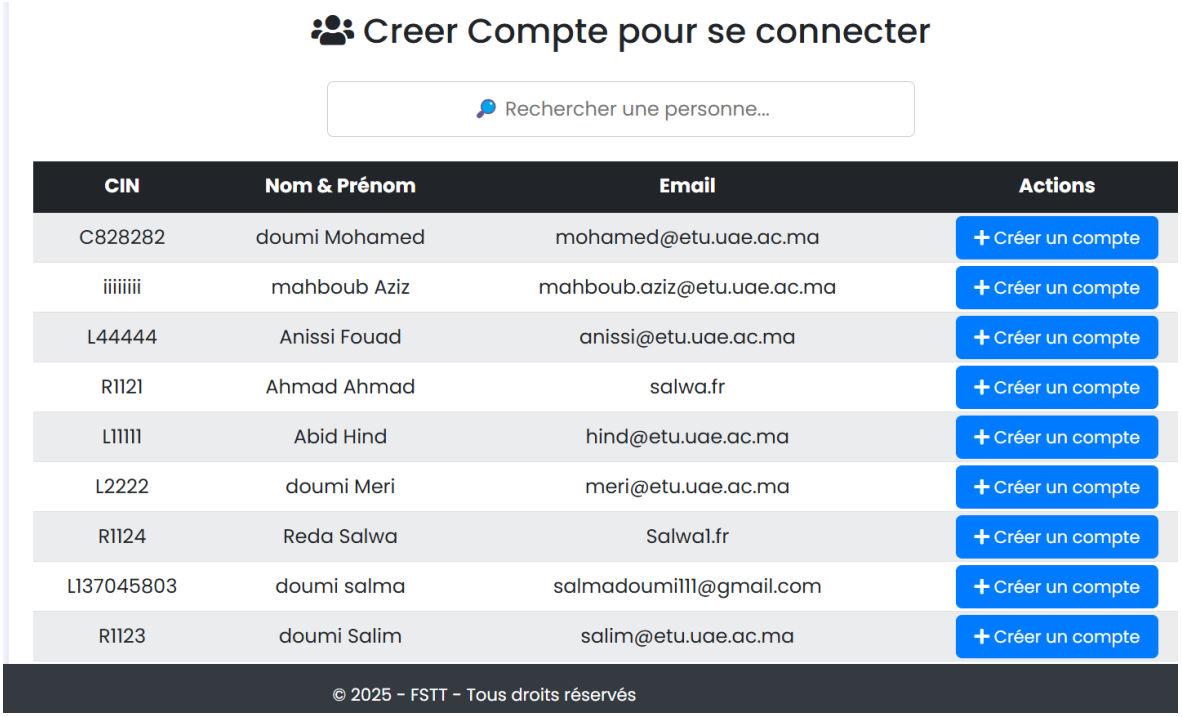


- User management interfaces

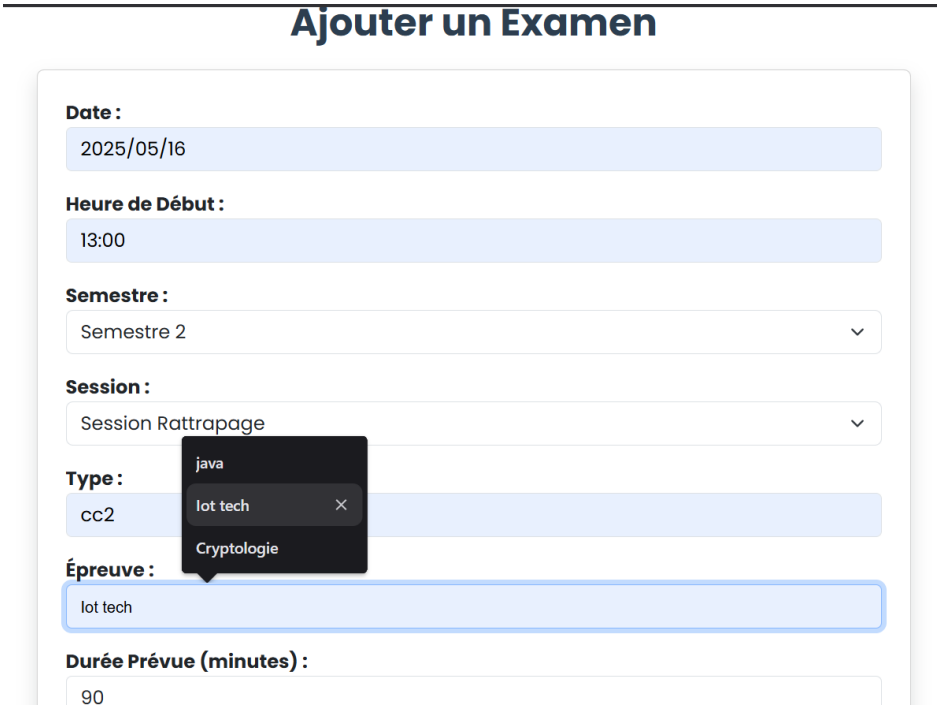


- Educational element management screens

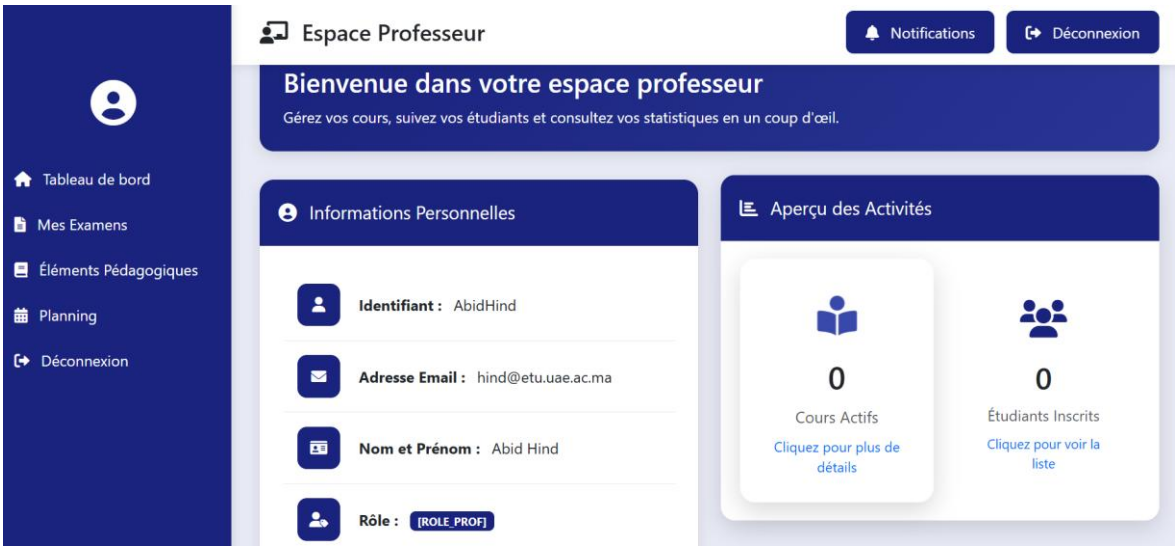




- Exam scheduling and management forms



- Prof dashboard with statistics vizualiation



5.2.2 Ajax for Asynchronous Communication

```
function loadUserData() {
    $.ajax({
        url: '/api/personnes',
        type: 'GET',
        dataType: 'json',
        headers: {
            'Authorization': 'Bearer ' + localStorage.getItem('token')
        },
        success: function(data) {
            populateUserTable(data);
        },
        error: function(xhr, status, error) {
            console.error('Error loading user data:', error);
            showErrorMessage('Failed to load user data');
```

```
    }
  });
}
```

```
<!-- Script pour filtrer les enseignants -->
<script>
    document.getElementById("searchInput").addEventListener("keyup", function () {
        let input = this.value.toLowerCase();
        let options = document.querySelectorAll("#enseignantSelect option");

        options.forEach(option => {
            let text = option.textContent.toLowerCase();
            option.style.display = text.includes(input) || option.value === "" ? "" : "none";
        });
    });

    function confirmUpdate() {
        return confirm("Êtes-vous sûr de vouloir mettre à jour cet élément pédagogique ?");
    }
</script>
```

5.2.3 JWT Authentication Integration

```
function login(credentials) {
    $.ajax({
        url: '/api/auth/login',
        type: 'POST',
        contentType: 'application/json',
        data: JSON.stringify(credentials),
        success: function(response) {
            localStorage.setItem('token', response.token);
            localStorage.setItem('role', response.role);
            redirectBasedOnRole(response.role);
        },
        error: function(xhr, status, error) {
            showErrorMessage('Invalid credentials');
        }
    });
}

function setupAuthInterceptor() {
    $(document).ajaxSend(function(e, xhr, options) {
        const token = localStorage.getItem('token');
        if (token) {
            xhr.setRequestHeader('Authorization', 'Bearer ' + token);
        }
    });}
```

6. Deployment and Configuration

6.1 Application Properties

```
spring.application.name=examPlanApp
gsabs.app.mode=DEV
# Logs
logging.level.org.springframework=INFO
logging.level.org.hibernate=INFO
logging.level.com=ERROR
## View Resolver
spring.mvc.view.prefix=/WEB-INF/jsp/view/
spring.mvc.view.suffix=.jsp
## DATABASE
spring.datasource.url=jdbc:postgresql://localhost:5432/plannig_exam
spring.datasource.username=postgres
spring.datasource.password=1234
spring.datasource.driver-class-name=org.postgresql.Driver
# Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.
logging.level.root=DEBUG
server.port=8081
```

6.2 Deployment Options

- Containerized deployment with Docker
- Cloud deployment on services-AWS

7. Future Enhancements

7.1 Technical Improvements

- Migrate from JSP to a modern frontend framework like React or Angular
- Implement microservices architecture for better scalability
- Add comprehensive API documentation with Swagger
- Enhance testing coverage with unit and integration tests

7.2 Functional Improvements

- Advanced reporting and analytics
- Mobile application for students
- Integration with external calendaring systems
- Automated email notifications

8. Conclusion

The Exam Management Application successfully implements a comprehensive solution for educational institutions to manage their administrative processes, particularly focusing on exam scheduling and user management. The system's architecture provides a solid foundation for future enhancements and scaling.

The implementation of Spring Boot on the backend and JSP/Ajax/Bootstrap on the frontend offers a robust, secure, and user-friendly platform that meets the needs of different user roles within the educational ecosystem.