**Employee Management Application**

**(Backend)**

**1. Introduction**

This application serves as a centralised platform for managing employee information within an organisation. By leveraging modern web technologies and best practices, our goal is to streamline employee data management, enhance security control through JWT Token and ensure scalability for future growth.

**2. Login using Spring Security with JWT Token Authentication**

This application ensures secure authentication using Spring Security integrated with JWT (JSON Web Token). JWT provides a stateless authentication mechanism where the server generates a token upon successful login, which is then included in subsequent requests to authenticate and authorise access.

#### **2.1. Functionality Highlights:**

* **Authentication Flow:**
  + Users submit credentials via a “/authenticate” API.
  + Spring Security validates the credentials and generates a JWT upon successful authentication.
* **Authorization:**
  + Each JWT contains user roles and permissions, allowing fine-grained access control.
  + JWTs are validated on each protected endpoint to ensure authorised access.
* **Token Expiry:**
  + JWTs have a configurable expiry to enhance security.
  + Once Expired, new tokens can be provided with re-authentication.

#### **2.2. Implementation Details:**

* **Security Configuration (SecurityConfiguration.java):**

**SecurityFilterChain:** Configures security settings for HTTP requests, including disabling CSRF and CORS protection since JWT-based authentication is stateless.

**jwtAuthenticationFilter:** Handles authentication requests (/authenticate endpoint) by extracting credentials, authenticating them using AuthenticationManager, and generating a JWT upon successful authentication.

**jwtAuthorizationFilter:** Validates incoming JWTs on every request (/\*\* path pattern) to authorise access based on the token's validity

##### 

##### **Authentication Filter (AuthenticationFilter.java):**

**doFilterInternal:** Handles incoming authentication requests (POST /authenticate) by extracting username and password from headers, authenticating them using AuthenticationManager, and generating a JWT upon successful authentication.

**generateToken:** Creates a JWT with the subject (username), issued date, expiration date, and signs it using the provided secret key.

* **Authorization Filter (JwtAuthorizationFilter.java):**

**doFilterInternal:** Intercept incoming requests (/\*\* path pattern) to validate JWTs. It extracts the JWT from the Authorization header, parses and verifies its validity using the secretKey, and loads the associated user details from the database using userService.

**validateToken:** Checks if the extracted token is valid (not expired) and matches the provided userDetails.

#### **2.3. Benefits:**

* Enhanced Security: JWT-based authentication eliminates the need for server-side session management, reducing security vulnerabilities associated with traditional session-based authentication.
* Scalability: Stateless nature of JWTs allows our application to handle authentication requests efficiently and scale seamlessly with increasing user load.
* Flexibility: JWTs can carry custom claims and are interoperable across different platforms, facilitating integration with external services and microservices architectures.

### **3. Employee Management**

#### **3.1. List of Employees**

The heart of our application lies in the management of employee records. Through a user-friendly interface, administrators and authorised personnel can view, add, edit, and delete employee information. Each employee record includes essential details such as name, department, contact information, and role within the organisation.

##### **Functionality Highlights:**

* **View Employees:** Displaying a list of all employees with associated departments.
* **Add Employee:** Creating new employee record.
* **Edit Employee:** Ability to modify existing employee details with validation checks.
* **Delete Employee:** Secure deletion process to remove outdated or redundant records.

#### **3.2. Employee CRUD Operations**

To achieve seamless data management, I have Utilised RESTful services for performing CRUD (Create, Read, Update, Delete) operations on employee entities. Each operation corresponds to an HTTP method:

* **Create (POST):** Adding a new employee record to the database.
* **Read (GET):** Retrieving employee details for display purposes.
* **Update (PUT):** Modifying existing employee information based on user input.
* **Delete (DELETE):** Removing employee records securely while maintaining data integrity.

These operations are facilitated by Spring Data JPA, which simplifies database interactions and ensures efficient handling of entity relationships within our H2 database.

### **4. Department Management**

#### **4.1. List of Departments**

In addition to managing employees, our application also supports the management of departments. Each department is associated with one or more employees and has attributes such as name, address, organisation and head of the department.

##### **Functionality Highlights:**

* **View Departments:** Listing all departments
* **Add Department:**Creating new departments.
* **Edit Department:** Ability to modify department details, including updating the details.
* **Delete Department:** Deleting departments

**4.2. Department CRUD Operations**

Similar to employee management, department operations are also exposed as RESTful services:

* **Create (POST):** Adding a new department.
* **Read (GET):** Fetching department details**.**
* **Update (PUT):** Modifying department attributes.
* **Delete (DELETE):** Removing departments.

### **5. RESTful Services**

Our application adheres to RESTful principles, which provide a standardised approach for building scalable and interoperable web services. Each endpoint follows URI conventions and utilises appropriate HTTP methods for data manipulation. This architecture ensures flexibility, allowing our application to integrate seamlessly with other systems and support various client interfaces.

### **6. Data Management with JPA and H2 Database**

For persistent data storage, I have leveraged the Java Persistence API (JPA) in conjunction with an H2 in-memory database. JPA simplifies the mapping between Java objects (entities) and relational database tables, enabling efficient CRUD operations and complex query executions. The H2 database, being lightweight and embedded, facilitates rapid development and testing without the need for a separate database server.

### 

### **7. Logging with SLF4J**

Effective logging is crucial for monitoring application behaviour, debugging issues, and auditing user actions. Our application utilises SLF4J (Simple Logging Facade for Java) to record relevant events and messages at different logging levels (e.g., DEBUG, INFO, ERROR). This ensures transparency in application operations and aids in diagnosing potential issues during development, testing.

**8. Code Coverage**

To ensure the reliability and maintainability of our application codebase, I have used JUnit for writing and executing unit tests and Mockito for mocking dependencies. This combination allows us to isolate components and verify their behaviour in controlled environments.

**9. Conclusion**

In conclusion, Employee Management Application represents a robust solution for organisations seeking efficient employee and department management. By leveraging Spring Boot, Spring Security with JWT token, JPA, and RESTful services, I have delivered a secure, scalable, and user-friendly platform. Future enhancements may include advanced reporting capabilities, integration with external systems, and support for additional authentication mechanisms.

.