

# TASK-MANAGEMENT-APPLICATION

The Task Management Application is a web-based platform designed to facilitate task assignment, tracking, and management within an organization. It provides functionalities for employers to assign tasks to employees, track task progress, and manage task details. Employees can receive assigned tasks, update task status, and provide completion details.

## **PROJECT STRUCTURE:**

**Spring Boot Application:** This will serve as the main entry point for our application.

**Controllers:** Define REST endpoints for handling requests.

**Services:** Implement business logic for task management and user authentication.

**Repositories:** Interface with the database for CRUD operations.

**Entities:** Define the structure of our data models.

**Security Configuration:** Implement JWT authentication and authorization.

**Auditing Interceptor:** Track changes made to tasks.

## **DETAILS:**

### **Task Assignment:**

Employers can create tasks and assign them to specific employees. Each task is assigned a unique ID for identification. Employers can specify the planned duration, time range, and other details for each task.

### **Task Tracking:**

Employees can view their assigned tasks and track their progress. Task status (e.g., pending, in progress, completed) is updated as employees work on tasks.


### **Task Details Management:**

Employers can manage task details such as descriptions, priorities, attachments, etc. Employees can provide task completion details, update task descriptions, and attach relevant files.

## User Management:

The application supports user management functionalities for both employers and employees. Employers can create and manage employee accounts.

```
project-root/
|
├─ src/
|   └─ main/
|       └─ java/
|           └─ com.example/
|               └─ config/
|                   ├── SecurityConfig.java      # Spring Security configuration
|                   └─ SwaggerConfig.java       # Swagger configuration
|               └─ controller/
|                   ├── AuthController.java     # Controller for authentication endpoints
|                   └─ TaskController.java      # Controller for task endpoints
|               └─ model/
|                   ├── Task.java              # Task entity
|                   ├── User.java              # User entity
|                   └─ ...                      # Other entity classes
|               └─ repository/
|                   ├── TaskRepository.java     # Repository interfaces
|                   └─ UserRepository.java      # Repository interfaces
|               └─ service/
|                   ├── AuthService.java        # Service for authentication
|                   ├── TaskService.java        # Service for task operations
|                   └─ ...                      # Other service classes
|               └─ Application.java             # Main Spring Boot application class
|           └─ resources/
|               ├── static/
|               ├── templates/
|               └─ application.properties       # Application properties
|       └─ test/                               # Test classes (unit & integration tests)
|
├─ README.md                                  # Project documentation
└─ pom.xml                                    # Maven project configuration
```



## **COMPONENTS:**

### **Model:**

Defines the structure of entities such as Task, Employee, Employer, etc. Includes JPA annotations for mapping entities to database tables.

### **View:**

Frontend interface for interacting with the application. Implemented using HTML, CSS, and JavaScript frameworks (not covered in this document).

### **Controller:**

Handles HTTP requests and orchestrates the flow of data between the model and view. Implements RESTful APIs for task management operations.

### **Service:**

Contains business logic and interacts with the repository layer. Manages tasks, users, and other application functionalities.

### **Repository:**

Interface for interacting with the database. Implements CRUD operations and data access logic using Spring Data JPA.

## **API DEVELOPMENT :**

### **1.JWT Authentication for Role-Based Access:**

Implement JWT token generation upon successful login. Use roles (Employer and Employee) to restrict access to certain endpoints.

### **2. Auditing for Task Changes:**

Utilize Spring Data JPA's auditing features to automatically track changes to entities. Implement custom auditing for specific entities like tasks.

### **3. Documenting API Endpoints:**

Create detailed documentation specifying each API endpoint, HTTP method, request/response format, and functionality. Include examples for request/response payloads. Use tools like Swagger or Spring Rest Docs for automated API documentation.

#### **4. User Authentication APIs:**

- Login Endpoint: Authenticates users and generates JWT tokens.
- Logout Endpoint: Clears the user's authentication token.
- User Profile Endpoint: Retrieves user details based on the authenticated token.

#### **5. Task CRUD Operations:**

- Create Task Endpoint: Allows authenticated users to create tasks.
- Read Task Endpoint: Retrieves task details based on task ID.
- Update Task Endpoint: Allows authorized users to update task details.
- Delete Task Endpoint: Allows authorized users to delete tasks.

#### **6. Task Assignment and Status Updates:**

- Assign Task Endpoint: Allows employers to assign tasks to employees.
- Update Task Status Endpoint: Enables employees to update task status (e.g., from Pending to Completed).

#### **7. Auditing APIs:**

- Log Viewing Endpoint: Retrieves audit logs for tasks or other audited entities.
- History Tracking Endpoint: Provides historical changes to tasks or other audited entities.

### **ARCHITECTURE:**

#### **1. Presentation Layer:**

Responsible for handling user interactions. Includes the API controllers for handling HTTP requests. Implements authentication and authorization using JWT tokens.

#### **2. Service Layer:**

Contains business logic and orchestrates operations. Validates requests, processes data, and interacts with repositories. Manages user authentication, task assignment, status updates, etc.

### **3. Repository Layer:**

Handles data access and persistence. Provides interfaces for CRUD operations on entities. Implements auditing features for tracking changes to tasks.

### **4. Domain Model:**

Represents the core business entities such as Task, User, etc. Contains the attributes and behaviors of these entities. Ensures data integrity and consistency.

### **5. Security Layer:**

Implements security features such as JWT authentication and authorization. Controls access to resources based on user roles (Employer and Employee).

### **6. Documentation and Testing:**

Provides comprehensive documentation for API endpoints and functionalities. Includes detailed code comments for better understanding and maintainability. Implements unit tests and integration tests to ensure correctness and reliability.

## **TECHNOLOGIES:**

**Spring Boot:** For rapid development and easy configuration of Spring-based applications.

**Spring Security:** For implementing JWT-based authentication and role-based access control.

**Spring Data JPA:** For simplifying data access and persistence.

**Hibernate Envers:** For auditing and versioning of entities.

**Swagger/OpenAPI:** For automated API documentation.

**JUnit/Mockito:** For unit testing and mocking dependencies.

## **ANGULAR :**

## **ARCHITECTURE**

## **COMPONENT STRUCTURE:**

Organize components into separate folders based on functionality:

- **task-list:** Component for displaying a list of tasks.
- **task-details:** Component for displaying details of a selected task.
- **user-auth:** Components related to user authentication, such as login and user profile.

## **ROUTING:**

Define routing configuration in the `app-routing.module.ts` file to navigate between different components based on user actions and URLs.

## **SERVICES :**

Create Angular services for managing API calls and sharing data between components:

- **task.service.ts:** Service for interacting with task-related backend APIs.
- **auth.service.ts:** Service for managing user authentication.

## **FEATURES :**

### **User Authentication :**

Implement user authentication features such as login, logout, and user profile management using JWT tokens. Use the `auth.service.ts` for handling authentication-related logic and API calls.

### **Task Management :**

Display Tasks Display tasks in a user-friendly interface with filtering and sorting options. Use the `task-list` component to list tasks retrieved from the backend.

### **1.Task Assignment and Tracking :**

- Allow employers to assign tasks using the `task-details` component.
- Enable tracking of task progress and status updates.

### **2.Employee Task Handling:**

- Allow employees to view assigned tasks in the `task-list` component.
- Provide options for updating task status and providing completion details.

## **INTEGRATION WITH BACKEND APIS :**

Connect the Angular application to backend APIs for seamless data exchange. Use Angular's HttpClient module to make HTTP requests to the backend endpoints defined in the backend documentation.

## **CONCLUSION :**

In conclusion, the task management application project has successfully achieved its objectives of providing a robust and user-friendly platform for managing tasks efficiently. Throughout the development process, we have implemented various features and functionalities to meet the needs of both employers and employees, ensuring seamless task assignment, tracking, and completion.

**Role-Based Access Control:** Implemented JWT authentication to ensure secure access control based on user roles, allowing employers to assign tasks and track progress while enabling employees to view and update their assigned tasks.

**Backend API Development:** Developed a set of RESTful APIs using Spring Boot, providing CRUD operations for tasks, user authentication, and auditing functionalities, along with clear documentation for each API endpoint.

**Frontend Development:** Created an intuitive and responsive user interface using Angular, connecting seamlessly with the backend APIs to provide a seamless user experience.