EMPLOYEE MANAGEMENT SYSTEM

**Introduction**…………………………………………………………………………………………………………………………………………2

[**Prerequisites**: 2](#_Toc179141877)

[Getting the code into your local system: 3](#_Toc179141878)

[Opening the project in an IDE: 4](#_Toc179141879)

[Setting the database: 5](#_Toc179141880)

[Starting the application: 6](#_Toc179141881)

[**API DOCUMENTATION:** 8](#_Toc179141882)

[**ENDPOINTS:** 8](#_Toc179141883)

[Api documentation Rest API: 9](#_Toc179141885)  
[Api Documentation For Webcontroller 11](#_Toc179141884)

[**DATA VALIDATION RULES** 13](#_Toc179141886)

[Employee creation & Updation Rules 13](#_Toc179141887)

[**SCHEMAS:** 14](#_Toc179141892)

[Employee table schema(Database): 14](#_Toc179141893)

[User table schema(Spring Boot): 14](#_Toc179141894)

[**FOREIGN KEY REFERENCES** 16](#_Toc179141901)

[Dummy employee data for use: 16](#_Toc179141902)

[conclusion 19](#_Toc179141902)

**INTRODUCTION:**  
 Application designed to facilitate the management of employee records within an organization. It allows authorized administrators to create, view, update, and delete employee profiles through a secure web interface. Key details managed include personal information, contact details, and employment data like hire date, job title, and department.

The application features robust user authentication (database-backed) and role-based authorization, ensuring only users with administrative privileges can perform management operations 1 . It provides both a primary Web interface (built with Thymeleaf) for direct interaction and a RESTful API for programmatic access 2 . Core functionalities include pagination, sorting, and filtering capabilities for efficient data handling, along with data validation to ensure integrity.   
The project is structured with a clear separation of concerns, utilizing DTOs (Data Transfer Objects), controllers, services, and repositories to handle data flow and business logic effectively. The following sections focus on how to set up, configure, and run this application.

Moreover, the application utilizes the following key components:

1. JDK 17
2. Maven
3. Spring boot 3.4.4
4. MySQL
5. Git and Github
6. Thymeleaf template engine (for the Front-end files)
7. HTML
8. Bootstrap-CSS
9. Javascript

The complete code for the application is stored in the git hub repository: <https://github.com/arunprakashxavier/employee-management-system>  
  
**Prerequisites:**

1. An IDE (like, IntelliJ)
2. JDK 17
3. Maven (Build tool)
4. MySQL along with an IDE(like MySQL Workbench)
5. A browser (like Chrome)

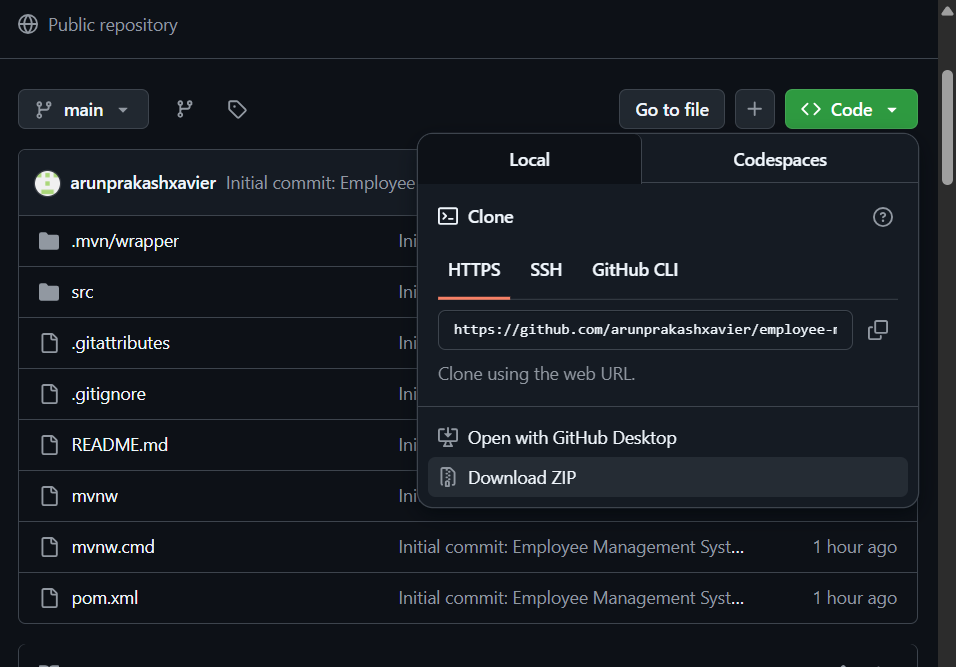
Now, let us proceed to the set-up and configuration of the project.

## 

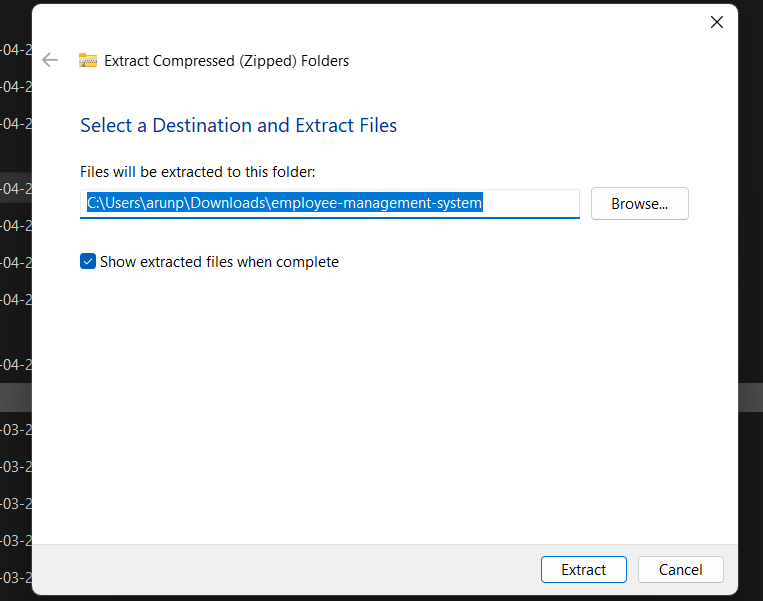
## Getting the code into your local system:

1. Get the code from the github repository <https://github.com/arunprakashxavier/employee-management-system> to your system.

One of the methods you can do is download the file as a zip folder, shown below:



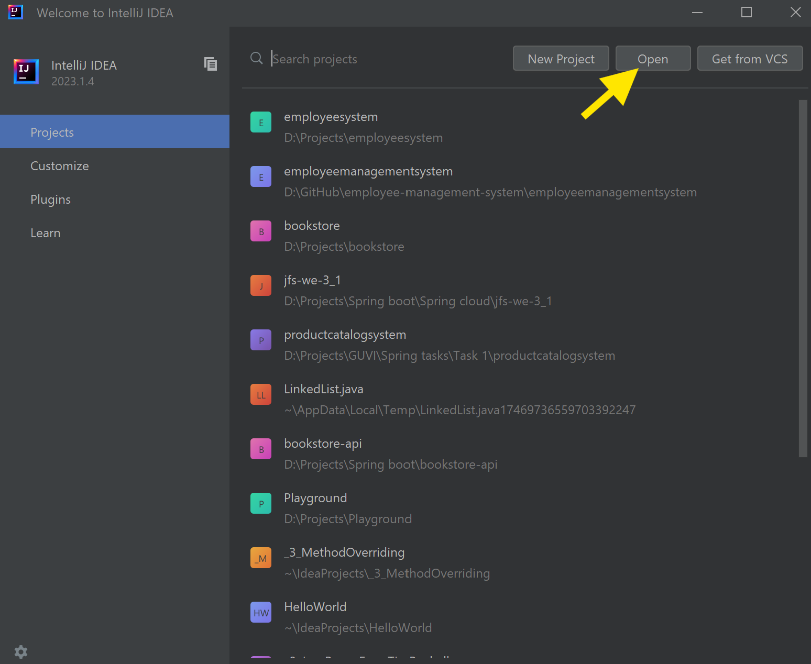
1. Then extract the downloaded zip folder into a folder as per your interest.



## Opening the project in an IDE:

In this manual and my project, I will be using IntelliJ as my preferred IDE. Whereas using Eclipse or any other IDE shouldn’t be a problem as the procedure is almost the same in any IDE.

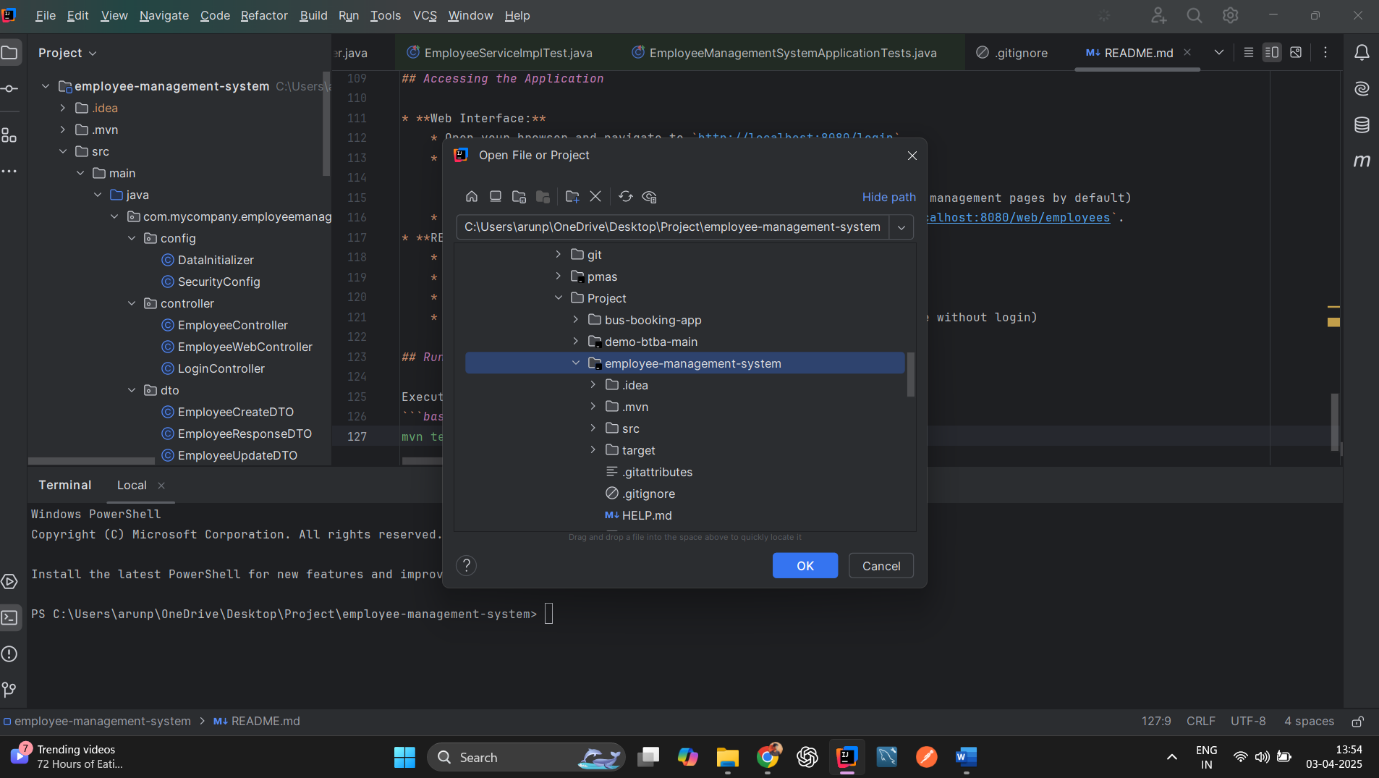
1. Open IntelliJ. Select *Open*.



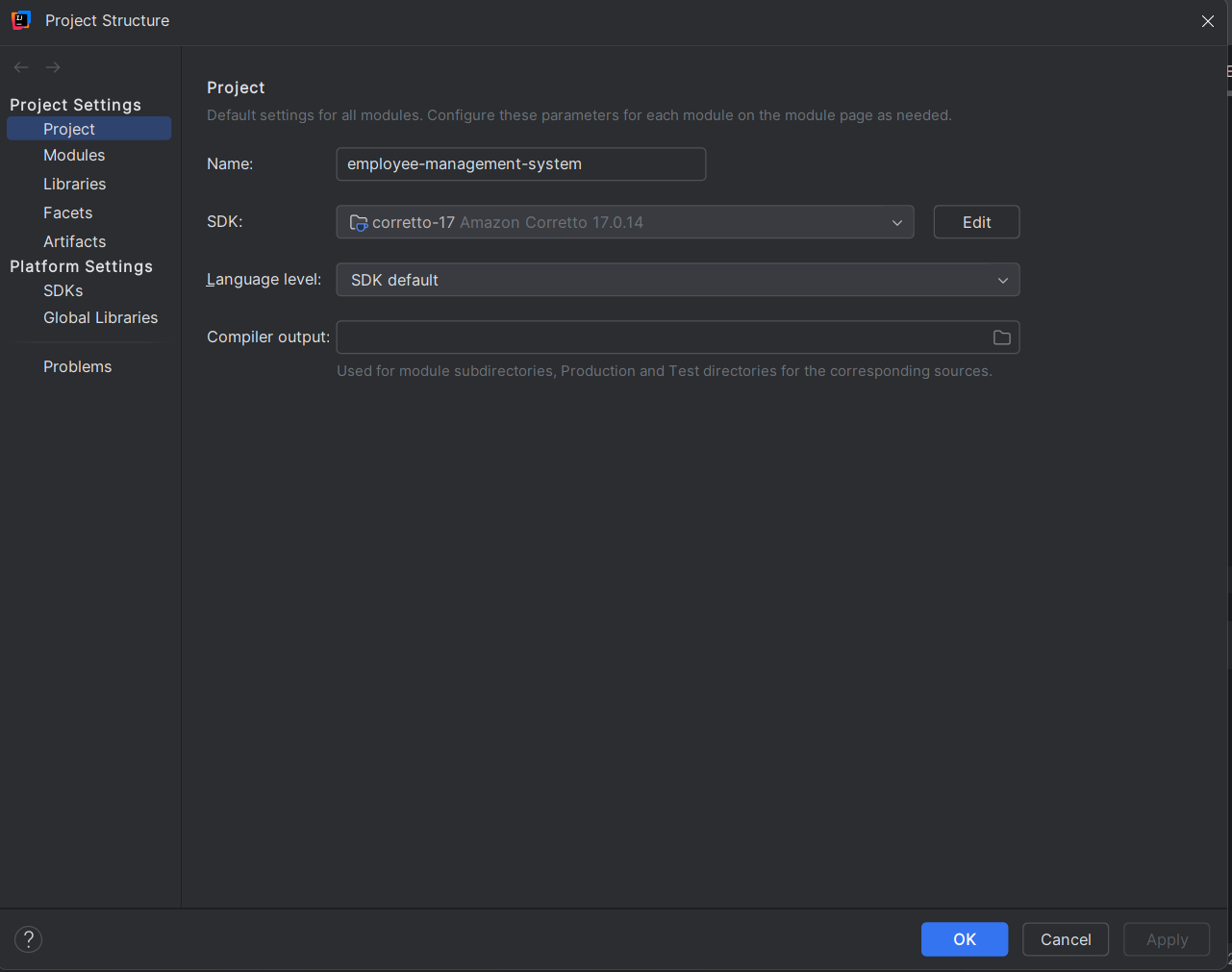
1. Browse to the path where we extracted our project and select *OK*.

Wait for a few seconds as the project will automatically load. During this time, the maven dependencies must also load on its own. You can also manually do this by clicking

*maven -> Reload all maven projects.*

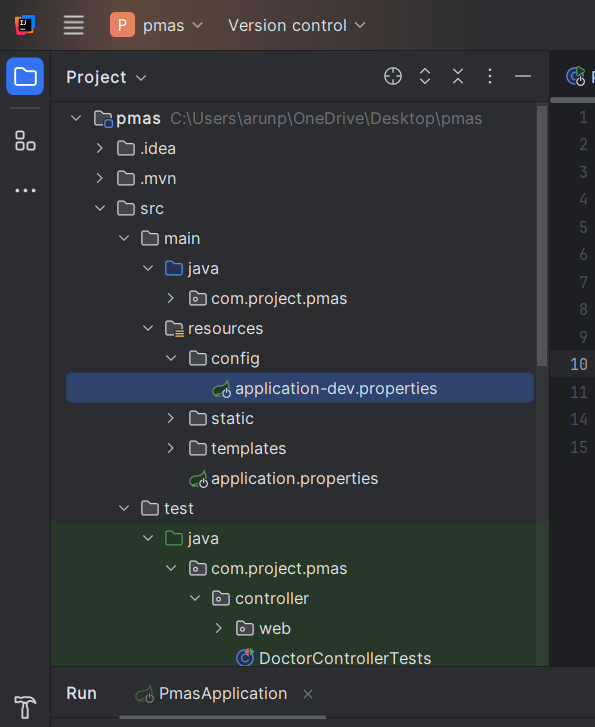


1. Go to *File -> Project Structure -> Project* and ensure that *JDK 17* is selected as the SDK



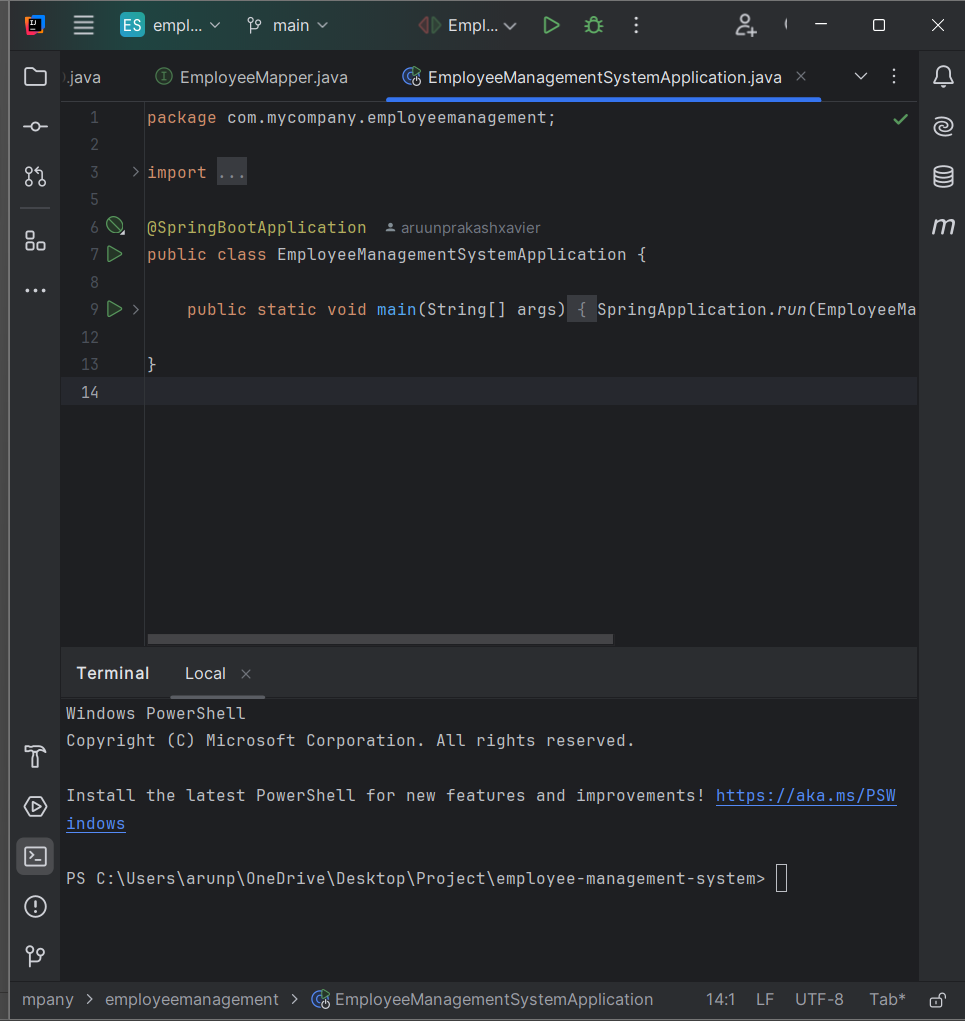
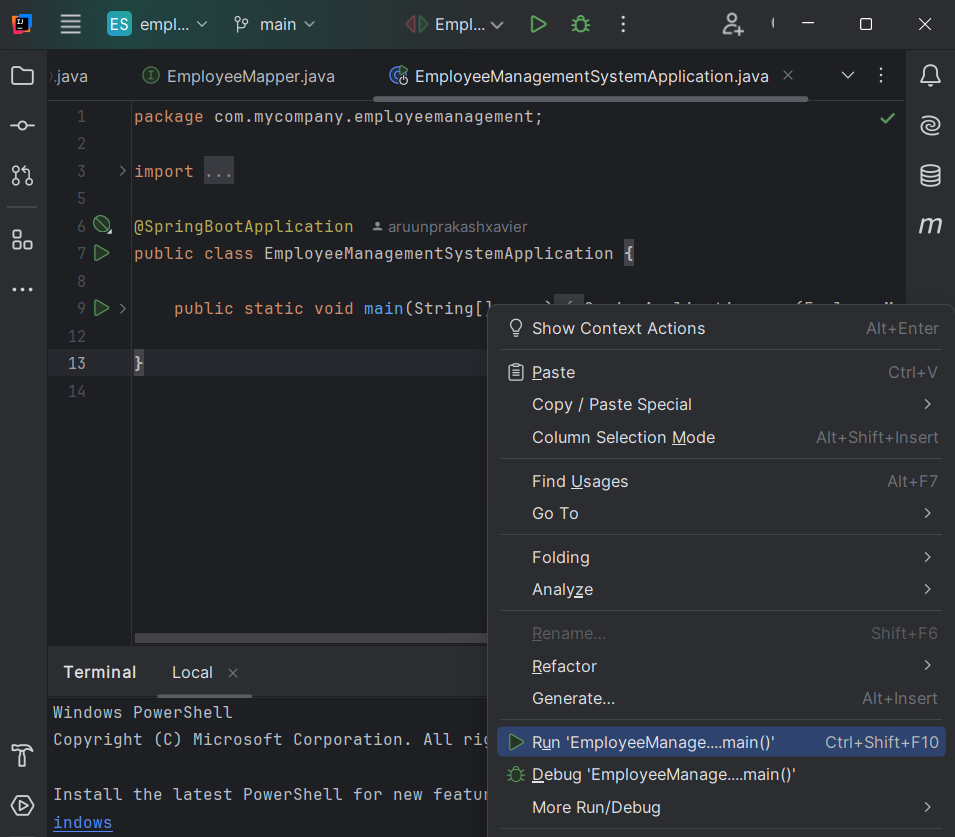
## Setting the database:

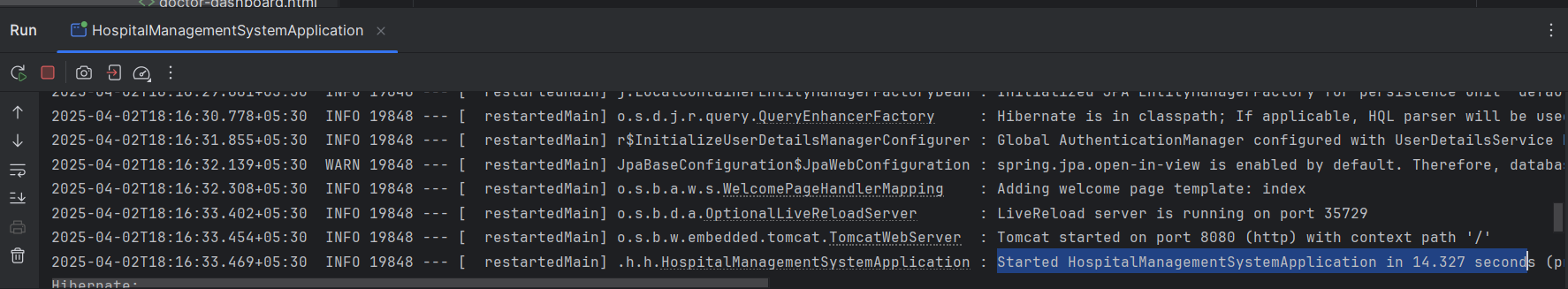
The application is created having MySQL as its database for storing its data. The main configurations of the application are written inside the application-dev.properties file inside

***src->main->resources->config->application-dev.properties***

* As shown, change the *url, username* and *password* properties as per your system configurations.
* Create a database in MySQL for this application.
* **Do ensure that the database name and the name present in the url (in this case: employee\_db) are the exact same.** Any difference in the name will cause error while running the application.

## Starting the application:

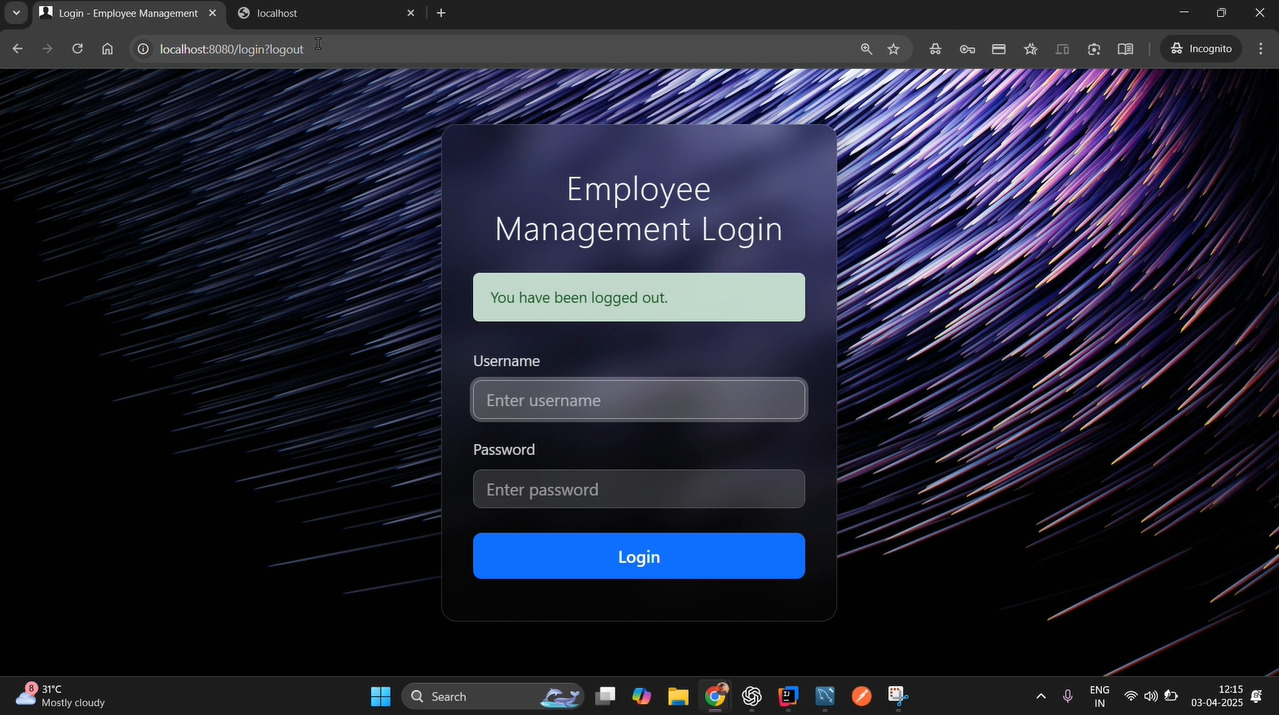
1. Before starting the application, ensure MySQL service is up and running.
2. Go to s*rc/main/java/com/mycompany/employeemanagement/EmployeeManagementSystemApplication.java*
3. 
4. Right-click inside the *EmployeeManagementSystemApplication.java f*ile and click EmployeeManagementSystemApplication.main()
5. 
6. On clicking Run, the application will start and you will see the below screen:



If the database name in the *application-dev.properties* and in the MySQL database match, Hibernate will automatically create a table as required and the system will get started.

1. In *application-dev.properties*, *8080* is the port number that is configured for this application. But ensure that the no other service is using that port and it is correctly mentioned in *server.port* in *application-dev.properties*.
2. Now, go to your browser and hit [http://localhost:<port-number>/ login](http://localhost:%3cport-number%3e/patients/login). For port 8080: <http://localhost:8080/login> . This should render the login page that after logging in will lead to the homepage of the application.

This is how the login page will look like:



Once logged in, you will be able to see the home page of the application:

From here, you can communicate with the application through the browser for add new employees

**Do checkout the video present in the repository for a project walk-through.**

## API DOCUMENTATION

**Overview:** The REST API provides endpoints for programmatic management of employee records. Access requires authentication (Basic Auth) and the ADMIN role.

## API DOCUMENTATION (REST API)

**Overview:** The REST API provides endpoints for programmatic management of employee records. Access requires authentication (Basic Auth) and the ADMIN role.

**Endpoints:**

**1. Create Employee**

* **Endpoint:** /api/v1/employees
* **Method:** POST
* **Controller:** EmployeeController.java
* **Description:** Creates a new employee record. Requires authentication and ADMIN role. Accepts employee data in the request body.
* **Authentication:** Basic Auth (admin/password123) required.
* **Request Body:** JSON representing EmployeeCreateDTO. Example:

JSON

{

"firstName": "New",

"lastName": "Employee",

"email": "new.emp@example.com",

"department": "HR",

"hireDate": "2025-04-03",

"jobTitle": "Recruiter",

"phoneNumber": "555111222"

}

* **Response:**
  + 201 Created: Success, returns created EmployeeResponseDTO in JSON body.
  + 400 Bad Request: Validation errors (response body contains error details).
  + 401 Unauthorized: Authentication missing or invalid.
  + 403 Forbidden: Authenticated user lacks ADMIN role.

**2. Get All Employees**

* **Endpoint:** /api/v1/employees
* **Method:** GET
* **Controller:** EmployeeController.java
* **Description:** Retrieves a paginated, sortable, and optionally filtered list of employees. Requires authentication and ADMIN role.
* **Authentication:** Basic Auth (admin/password123) required.
* **Query Parameters (Optional):**
  + page (e.g., 0, 1, ... - zero-indexed)
  + size (e.g., 5, 10, 20)
  + sort (e.g., lastName,asc, hireDate,desc) - can be repeated for multi-sort.
  + department (e.g., IT, Sales) - case-insensitive filter.
* **Response:**
  + 200 OK: Success, returns a Page object containing EmployeeResponseDTO list in JSON body.
  + 401 Unauthorized: Authentication missing or invalid.
  + 403 Forbidden: Authenticated user lacks ADMIN role.

**3. Get Employee by ID**

* **Endpoint:** /api/v1/employees/{id} (e.g., /api/v1/employees/1)
* **Method:** GET
* **Controller:** EmployeeController.java
* **Description:** Retrieves details for a single employee by their ID. Requires authentication and ADMIN role.
* **Authentication:** Basic Auth (admin/password123) required.
* **Path Variable:** Replace {id} with the actual employee ID.
* **Response:**
  + 200 OK: Success, returns EmployeeResponseDTO in JSON body.
  + 404 Not Found: Employee with the given ID does not exist.
  + 401 Unauthorized: Authentication missing or invalid.
  + 403 Forbidden: Authenticated user lacks ADMIN role.

**4. Update Employee**

* **Endpoint:** /api/v1/employees/{id} (e.g., /api/v1/employees/1)
* **Method:** PUT
* **Controller:** EmployeeController.java
* **Description:** Updates details for an existing employee by their ID. Requires authentication and ADMIN role.
* **Authentication:** Basic Auth (admin/password123) required.
* **Path Variable:** Replace {id} with the actual employee ID.
* **Request Body:** JSON representing EmployeeUpdateDTO with the fields to be updated. Example:

JSON

{

"firstName": "Updated Name",

"lastName": "Employee",

"email": "updated.emp@example.com",

"department": "Marketing",

"hireDate": "2025-04-01",

"jobTitle": "Marketing Lead",

"phoneNumber": "555333444"

}

* **Response:**
  + 200 OK: Success, returns updated EmployeeResponseDTO in JSON body.
  + 404 Not Found: Employee with the given ID does not exist.
  + 400 Bad Request: Validation errors (response body contains error details).
  + 401 Unauthorized: Authentication missing or invalid.
  + 403 Forbidden: Authenticated user lacks ADMIN role.

**5. Delete Employee**

* **Endpoint:** /api/v1/employees/{id} (e.g., /api/v1/employees/1)
* **Method:** DELETE
* **Controller:** EmployeeController.java
* **Description:** Deletes an employee record by their ID. Requires authentication and ADMIN role.
* **Authentication:** Basic Auth (admin/password123) required.
* **Path Variable:** Replace {id} with the actual employee ID.
* **Response:**
  + 204 No Content: Success, no response body.
  + 404 Not Found: Employee with the given ID does not exist.
  + 401 Unauthorized: Authentication missing or invalid.
  + 403 Forbidden: Authenticated user lacks ADMIN role.

**6. API Documentation UI**

* **Endpoint:** /swagger-ui.html
* **Method:** GET
* **Handler:** SpringDoc Library
* **Description:** Renders the interactive Swagger UI page for testing API endpoints.
* **Authentication:** Not required.
* **Response:** HTML page.

## WEB CONTROLLER ENDPOINTS (Thymeleaf UI)

**Overview:** These endpoints handle requests from the browser to render HTML pages using Thymeleaf templates. Access generally requires authentication and the ADMIN role, except for the login page.

**Endpoints:**

**1. Render Login Page**

* **Endpoint:** /login
* **Method:** GET
* **Controller:** LoginController.java
* **Description:** Renders the custom login page (login.html). Accessible without authentication.
* **Response:** HTML page (login.html).

**2. Process Login Submission**

* **Endpoint:** /login
* **Method:** POST
* **Handler:** Spring Security Filter Chain
* **Description:** Processes the username and password submitted from the login form. Handled internally by Spring Security.
* **Response:** Redirect (302) to /web/employees on success or /login?error on failure.

**3. Process Logout**

* **Endpoint:** /logout
* **Method:** POST (default configuration)
* **Handler:** Spring Security Filter Chain
* **Description:** Logs the current user out. Triggered by the logout form/button.
* **Response:** Redirect (302) to /login?logout.

**4. Render Employee List Page**

* **Endpoint:** /web/employees
* **Method:** GET
* **Controller:** EmployeeWebController.java
* **Description:** Renders the main employee list page (employees-list.html) with data (paginated, sortable, filterable). Requires authentication and ADMIN role. Accepts optional query parameters: page, size, sort, department.
* **Response:** HTML page (employees-list.html). Redirects to /login if not authenticated/authorized.

**5. Render 'Add New Employee' Form Page**

* **Endpoint:** /web/employees/new
* **Method:** GET
* **Controller:** EmployeeWebController.java
* **Description:** Renders the empty form (employee-form.html) for adding a new employee. Requires authentication and ADMIN role.
* **Response:** HTML page (employee-form.html). Redirects to /login if not authenticated/authorized.

**6. Process 'Add New Employee' Form Submission**

* **Endpoint:** /web/employees/save
* **Method:** POST
* **Controller:** EmployeeWebController.java
* **Description:** Handles the submission of the 'Add New Employee' form. Validates data and creates the employee via the service. Requires authentication and ADMIN role. Expects form data corresponding to EmployeeCreateDTO.
* **Response:** Redirect (302) to /web/employees on success (with flash message), or re-renders employee-form.html on validation error (displaying errors). Redirects to /login if not authenticated/authorized.

**7. Render 'Edit Employee' Form Page**

* **Endpoint:** /web/employees/edit/{id} (e.g., /web/employees/edit/5)
* **Method:** GET
* **Controller:** EmployeeWebController.java
* **Description:** Renders the employee form (employee-form.html) pre-populated with data for the employee specified by {id}. Requires authentication and ADMIN role.
* **Response:** HTML page (employee-form.html) with data. Redirects to /web/employees if ID not found (with flash message). Redirects to /login if not authenticated/authorized.

**8. Process 'Update Employee' Form Submission**

* **Endpoint:** /web/employees/update/{id} (e.g., /web/employees/update/5)
* **Method:** POST
* **Controller:** EmployeeWebController.java
* **Description:** Handles the submission of the 'Edit Employee' form. Validates data and updates the specified employee via the service. Requires authentication and ADMIN role. Expects form data corresponding to EmployeeUpdateDTO.
* **Response:** Redirect (302) to /web/employees on success (with flash message), or re-renders employee-form.html on validation error (displaying errors), or redirects to /web/employees if employee not found during update (with flash message). Redirects to /login if not authenticated/authorized.

**9. Process Employee Deletion**

* **Endpoint:** /web/employees/delete/{id} (e.g., /web/employees/delete/5)
* **Method:** GET (based on current link implementation)
* **Controller:** EmployeeWebController.java
* **Description:** Deletes the employee specified by {id}. Triggered by the 'Delete' link in the list (includes JS confirmation). Requires authentication and ADMIN role.
* **Response:** Redirect (302) to /web/employees (with success or error flash message). Redirects to /login if not authenticated/authorized.

**10. Render Employee View Page**

* **Endpoint:** /web/employees/view/{id} (e.g., /web/employees/view/5)
* **Method:** GET
* **Controller:** EmployeeWebController.java
* **Description:** Renders the read-only details page (employee-view.html) for the specified employee. Requires authentication and ADMIN role.
* **Response:** HTML page (employee-view.html). Redirects to /web/employees if ID not found (with flash message). Redirects to /login if not authenticated/authorized.

## Data Validation Rules

In this application, data validation ensures that employee information submitted via the API or the web forms meets specific criteria before being processed or saved. These rules are primarily defined in the EmployeeCreateDTO and EmployeeUpdateDTO classes using Jakarta Bean Validation annotations.

### Employee Creation & Update Validation Rules

The following rules apply when adding a new employee or updating an existing one:

* **First Name**
  + **Required:** Yes
  + **Validation:** Must not be blank (null or empty). Must be between 2 and 50 characters long.
  + **Constraints:**
    - @NotEmpty(message = "First name cannot be empty")
    - @Size(min = 2, max = 50, message = "First name must be between 2 and 50 characters")
* **Last Name**
  + **Required:** Yes
  + **Validation:** Must not be blank. Must be between 2 and 50 characters long.
  + **Constraints:**
    - @NotEmpty(message = "Last name cannot be empty")
    - @Size(min = 2, max = 50, message = "Last name must be between 2 and 50 characters")
  + (Note: Does not currently restrict characters to only letters/spaces)
* **Email**
  + **Required:** Yes
  + **Validation:** Must not be blank. Must be a well-formed email address. Must not exceed 100 characters.
  + **Constraints:**
    - @NotEmpty(message = "Email cannot be empty")
    - @Email(message = "Email should be valid")
    - @Size(max = 100, message = "Email must not exceed 100 characters")
  + **Format Example:** example@domain.com
* **Phone Number**
  + **Required:** No
  + **Validation:** If provided, must not exceed 20 characters.
  + **Constraints:**
    - @Size(max = 20, message = "Phone number must not exceed 20 characters")
* **Hire Date**
  + **Required:** No
  + **Validation:** If provided, must be a date in the past or the present day. Cannot be a future date.
  + **Constraints:**
    - @PastOrPresent(message = "Hire date cannot be in the future")
* **Job Title**
  + **Required:** No
  + **Validation:** If provided, must not exceed 100 characters.
  + **Constraints:**
    - @Size(max = 100, message = "Job title must not exceed 100 characters")
* **Department**
  + **Required:** No
  + **Validation:** If provided, must not exceed 100 characters.
  + **Constraints:**
    - @Size(max = 100, message = "Department must not exceed 100 characters")

.

# **SCHEMAS:**

### employees table schema (Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** | **Remarks** |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| first\_name | varchar(50) | NOT NULL |
| last\_name | varchar(50) | NOT NULL |
| email | varchar(100) | NOT NULL, UNIQUE |
| phone\_number | varchar(20) | DEFAULT NULL |
| hire\_date | date | DEFAULT NULL |
| job\_title | varchar(100) | DEFAULT NULL |
| department | varchar(100) | DEFAULT NULL |

### users table schema (Database):

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Datatype** |  |
| id | bigint | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT |
| username | varchar(50) | NOT NULL, UNIQUE |
| password | varchar(60) | NOT NULL |
| roles | varchar(255) | NOT NULL |
| enabled | boolean | NOT NULL |

### **1. Admin Login Details**

### Our current application uses database-backed users with Spring Security's Form Login for the web UI and supports HTTP Basic Auth for the API. There isn't a specific JSON-based API login endpoint like /api/auth/admin/login configured. Here's how the admin logs in:

### **Method A: Web UI Login (Using Browser)**

### **Navigate To:** http://localhost:8080/login

### **Action:** User interacts with the HTML login form (login.html).

### **Credentials (entered into form fields):**

### username: admin

### password: password123

### **Submission:** The form performs a POST request to /login with the credentials as form data (application/x-www-form-urlencoded). Spring Security handles this submission.

### **Result:** On success, the user is redirected to /web/employees with an active session.

### **Method B: API Access (Using Postman, Curl, etc.)**

### **Mechanism:** HTTP Basic Authentication

### **When:** Applied when making requests to protected API endpoints (e.g., GET /api/v1/employees, POST /api/v1/employees, etc.).

### **Credentials:**

### Username: admin

### Password: password123

### **How (Example in Postman):**

### Select the HTTP method (e.g., GET) and enter the API endpoint URL (e.g., http://localhost:8080/api/v1/employees).

### Go to the "Authorization" tab.

### Select "Basic Auth" from the "Type" dropdown.

### Enter admin in the "Username" field.

### Enter password123 in the "Password" field.

### Click "Send". Postman adds the required Authorization: Basic <encoded\_credentials> header to the request.

**2. Dummy Employee Data**  
  
 [

{

"firstName": "Arun",

"lastName": "Kumar",

"email": "arun.kumar@example.com",

"phoneNumber": "9876543210",

"hireDate": "2023-05-15",

"jobTitle": "Software Engineer",

"department": "Technology"

},

{

"firstName": "Priya",

"lastName": "Sharma",

"email": "priya.sharma@example.com",

"phoneNumber": "9123456789",

"hireDate": "2022-11-01",

"jobTitle": "HR Manager",

"department": "Human Resources"

},

{

"firstName": "Vikram",

"lastName": "Singh",

"email": "vikram.singh@example.com",

"phoneNumber": "9988776655",

"hireDate": "2024-01-10",

"jobTitle": "Sales Associate",

"department": "Sales"

},

{

"firstName": "Sunita",

"lastName": "Gupta",

"email": "sunita.gupta@example.com",

"phoneNumber": "9001122334",

"hireDate": "2021-08-20",

"jobTitle": "Senior Accountant",

"department": "Finance"

},

{

"firstName": "Rajesh",

"lastName": "Patel",

"email": "rajesh.patel@example.com",

"phoneNumber": "9777888999",

"hireDate": "2023-09-05",

"jobTitle": "Marketing Specialist",

"department": "Marketing"

},

{

"firstName": "Meena",

"lastName": "Iyer",

"email": "meena.iyer@example.com",

"phoneNumber": "9666555444",

"hireDate": "2024-03-01",

"jobTitle": "Software Tester",

"department": "Technology"

}

]

**Conclusion**

The **Employee Management System** successfully streamlines the process of managing employee records, improving efficiency and reducing manual workload. By integrating features such as **employee registration, role-based access control, attendance tracking, payroll management, and report generation**, the system enhances organizational productivity.

With a **futuristic UI** featuring translucent elements, soft glow effects, and a modern layout, the system provides a visually appealing and user-friendly experience. The integration of **secure authentication mechanisms, database management, and automated processes** ensures data integrity and operational reliability.

Future enhancements could include **AI-driven analytics, advanced reporting, and mobile compatibility** to further optimize workforce management. Overall, this system lays a strong foundation for effective and scalable employee administration.

Project done by  
 Arun Prakash.M