Employee Management System — Spring Boot (Java, Spring MVC, Hibernate, MySQL)

This document contains a minimal, working Employee Management System with role-based access (ADMIN / USER). It uses Spring Boot (Spring MVC), Spring Data JPA (Hibernate), MySQL, Spring Security, and Thymeleaf for views.

Project structure

```
employee-management/
├── pom.xml
├── src/main/java/com/example/ems/
│   ├── EmsApplication.java
│   ├── config/SecurityConfig.java
│   ├── controller/EmployeeController.java
│   ├── model/Employee.java
│   ├── repository/EmployeeRepository.java
│   └── service/EmployeeService.java
├── src/main/resources/
│   ├── application.properties
│   └── templates/
│       ├── list.html
│       ├── form.html
│       └── view.html
└── README.md
```

`pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>employee-management</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
```

```
        <java.version>17</java.version>
        <spring.boot.version>3.2.0</spring.boot.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>jakarta.validation</groupId>
            <artifactId>jakarta.validation-api</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

---

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/ems_db?
useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=your_mysql_password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.thymeleaf.cache=false

# Server port (optional)
server.port=8080
```

EmsApplication.java

```
package com.example.ems;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmsApplication {
    public static void main(String[] args) {
        SpringApplication.run(EmsApplication.class, args);
    }
}
```

---

model/Employee.java

```
package com.example.ems.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;

@Entity
@Table(name = "employees")
public class Employee {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "Name is required")
    private String name;

    @NotBlank(message = "Email is required")
    @Email
    @Column(unique = true)
    private String email;

    @NotBlank(message = "Role is required")
    private String role; // e.g., Developer, Manager

    @Min(18)
    @Max(70)
    private Integer age;

    // getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }

    public Integer getAge() { return age; }
    public void setAge(Integer age) { this.age = age; }
}
```

---

repository/EmployeeRepository.java

```
package com.example.ems.repository;

import com.example.ems.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {
```

```
    Optional<Employee> findByEmail(String email);
}
```

---

service/EmployeeService.java

```java
package com.example.ems.service;

import com.example.ems.model.Employee;
import com.example.ems.repository.EmployeeRepository;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

@Service
public class EmployeeService {

    private final EmployeeRepository repository;

    public EmployeeService(EmployeeRepository repository) {
        this.repository = repository;
    }

    public List<Employee> findAll() {
        return repository.findAll();
    }

    public Optional<Employee> findById(Long id) {
        return repository.findById(id);
    }

    public Employee save(Employee employee) {
        return repository.save(employee);
    }

    public void deleteById(Long id) {
        repository.deleteById(id);
    }
}
```

`controller/EmployeeController.java`

```
package com.example.ems.controller;

import com.example.ems.model.Employee;
import com.example.ems.service.EmployeeService;
import jakarta.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/employees")
public class EmployeeController {

    private final EmployeeService service;

    public EmployeeController(EmployeeService service) {
        this.service = service;
    }

    @GetMapping
    public String list(Model model) {
        model.addAttribute("employees", service.findAll());
        return "list";
    }

    @GetMapping("/new")
    public String createForm(Model model) {
        model.addAttribute("employee", new Employee());
        return "form";
    }

    @PostMapping
    public String save(@Valid @ModelAttribute("employee") Employee employee,
BindingResult br) {
        if (br.hasErrors()) {
            return "form";
        }
        service.save(employee);
        return "redirect:/employees";
    }

    @GetMapping("/edit/{id}")
    public String editForm(@PathVariable Long id, Model model) {
```

```
        service.findById(id).ifPresent(e -> model.addAttribute("employee", e));
        return "form";
    }

    @GetMapping("/view/{id}")
    public String view(@PathVariable Long id, Model model) {
        service.findById(id).ifPresent(e -> model.addAttribute("employee", e));
        return "view";
    }

    @GetMapping("/delete/{id}")
    public String delete(@PathVariable Long id) {
        service.deleteById(id);
        return "redirect:/employees";
    }
}
```

---

config/SecurityConfig.java

```
package com.example.ems.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    public InMemoryUserDetailsManager userDetailsService() {
        UserDetails admin = User.withDefaultPasswordEncoder()
                .username("admin")
                .password("adminpass")
                .roles("ADMIN")
                .build();

        UserDetails user = User.withDefaultPasswordEncoder()
```

```
                .username("user")
                .password("userpass")
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(admin, user);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/css/**", "/js/**").permitAll()
                .requestMatchers("/employees/new", "/employees/delete/**", "/
employees/edit/**").hasRole("ADMIN")
                .requestMatchers("/employees/**").hasAnyRole("ADMIN", "USER")
                .anyRequest().authenticated()
            )
            .formLogin(form -> form.defaultSuccessUrl("/employees", true))
            .logout(logout -> logout.logoutSuccessUrl("/login"));
        return http.build();
    }
}
```

Note: `withDefaultPasswordEncoder()` is used for simplicity in this demo. For production, use a stronger password encoder and persistent user store (JDBC or LDAP).

Thymeleaf templates (placed in `src/main/resources/templates`)

`list.html`

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Employees</title>
</head>
<body>
<h1>Employees</h1>
<p><a th:href="@{/employees/new}">Add Employee</a> | <a th:href="@{/
logout}">Logout</a></p>
<table border="1">
    <thead>
    <tr><th>ID</th><th>Name</th><th>Email</th><th>Role</th><th>Age</
th><th>Actions</th></tr>
```

```
    </thead>
    <tbody>
    <tr th:each="emp : ${employees}">
        <td th:text="${emp.id}"></td>
        <td th:text="${emp.name}"></td>
        <td th:text="${emp.email}"></td>
        <td th:text="${emp.role}"></td>
        <td th:text="${emp.age}"></td>
        <td>
            <a th:href="@{'/employees/view/' + ${emp.id}}">View</a>
            <span th:if="${#authentication.principal?.authorities.?
[0]?.authority == 'ROLE_ADMIN'}"> |
                <a th:href="@{'/employees/edit/' + ${emp.id}}">Edit</a> |
                <a th:href="@{'/employees/delete/' + ${emp.id}}"
onclick="return confirm('Delete?')">Delete</a>
            </span>
        </td>
    </tr>
    </tbody>
</table>
</body>
</html>
```

`form.html`

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title th:text="${employee.id} != null ? 'Edit' : 'New'">Employee Form</
title>
</head>
<body>
<h1 th:text="${employee.id} != null ? 'Edit Employee' : 'New Employee'"></h1>
<form th:action="@{/employees}" th:object="${employee}" method="post">
    <input type="hidden" th:field="*{id}"/>
    <div>
        <label>Name</label>
        <input th:field="*{name}" />
        <div th:if="${#fields.hasErrors('name')}" th:errors="*{name}"></div>
    </div>
    <div>
        <label>Email</label>
        <input th:field="*{email}" />
        <div th:if="${#fields.hasErrors('email')}" th:errors="*{email}"></div>
    </div>
```

```
    <div>
        <label>Role</label>
        <input th:field="*{role}" />
    </div>
    <div>
        <label>Age</label>
        <input th:field="*{age}" type="number" />
    </div>
    <div>
        <button type="submit">Save</button>
        <a th:href="@{/employees}">Cancel</a>
    </div>
</form>
</body>
</html>
```

`view.html`

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>View Employee</title>
</head>
<body>
<h1>Employee Details</h1>
<div>
    <p><strong>ID:</strong> <span th:text="${employee.id}"></span></p>
    <p><strong>Name:</strong> <span th:text="${employee.name}"></span></p>
    <p><strong>Email:</strong> <span th:text="${employee.email}"></span></p>
    <p><strong>Role:</strong> <span th:text="${employee.role}"></span></p>
    <p><strong>Age:</strong> <span th:text="${employee.age}"></span></p>
</div>
<p><a th:href="@{/employees}">Back</a></p>
</body>
</html>
```

Database

Run this SQL to create the database (or create it via MySQL Workbench):

```
CREATE DATABASE ems_db;
```

When the app runs with `spring.jpa.hibernate.ddl-auto=update` it will create the `employees` table automatically.

Run / Setup steps

1. Ensure MySQL is running and create `ems_db` database.
2. Update `application.properties` with your MySQL username/password.
3. Build and run:
4. `mvn clean package`
5. `mvn spring-boot:run`
6. Open `http://localhost:8080` — you'll be redirected to login.
7. Admin: `admin` / `adminpass` (can add/edit/delete)
8. User: `user` / `userpass` (can view list & details)

Notes & Next steps

- For production, move from in-memory users to JDBC-backed users and encode passwords with `BCryptPasswordEncoder`.
- Add DTOs and validation error handling to improve UX.
- Add REST API endpoints if you want a SPA or mobile client.
- Add pagination, search, and sorting for large datasets.

If you want, I can also: - Convert the views to a React frontend with a REST backend. - Use JDBC-based authentication and role management in the DB. - Provide Dockerfile and docker-compose for MySQL + app.