

Employee Management – Project Documentation

Overview

This is a Spring Boot application for managing employee records. It supports CRUD operations, validation, logging, exception handling, pagination, sorting, and Swagger API documentation. A basic security layer with role-based access control is also implemented.

Dependencies

- **spring-boot-starter-web** — REST API development
- **spring-boot-starter-data-jpa** — JPA and database access
- **spring-boot-starter-validation** — input validations (@Valid, constraints)
- **lombok** — reduces boilerplate (getters, setters, logs)
- **springdoc-openapi-ui** — Swagger UI
- **mysql-connector-java** — MySQL database support

Key Annotations

| Annotation | Purpose |
|--|-------------------------------------|
| @RestController | REST controller returning JSON |
| @Autowired | Inject dependencies |
| @Entity | Maps class to DB table |
| @Id, @GeneratedValue | Primary key and auto-generation |
| @GetMapping, @PostMapping, @PatchMapping, @DeleteMapping, @PutMapping | Map HTTP methods |
| @RequestBody | Read JSON body |
| @PathVariable, @RequestParam | Read URL path / query values |
| @Valid, @Validated | Enable validation in our controller |
| @NotBlank, @Min, @NotNull, @Null | Field-level validation |
| @Service, @Repository | Service and repository layers |
| @ControllerAdvice | Global exception handler |
| @ExceptionHandler | Handle specific exceptions |
| @Slf4j | Logging |
| @Tag | Swagger controller tag |

API Endpoints

| Method | Endpoint | Description | Input | Output |
|--------|--------------|--|-----------------|-----------------------|
| GET | /getemployee | Get all employees | — | List of employees |
| GET | //{id} | Get employee by ID | id | JSON employee |
| POST | /create | Create employee | JSON body | Status |
| PUT | //{id} | Update employee | id, JSON | Updated employee |
| PATCH | //{id} | Update name only | id, name | String |
| DELETE | //{id} | Delete one employee | id | String |
| DELETE | /deleteall | Delete all employees | — | String |
| GET | /get/{id} | Get formatted employee info | id | String |
| GET | /welcome | Welcome message | — | String |
| GET | /sal | Test arithmetic exception | — | Custom error response |
| POST | /signup | Register new user | JSON | Status (Admin only) |
| GET | /getalluser | Get users with USER role | — | List of users |
| GET | /getall | Return the employee with No paginations and sorting | No neccesary | Employee list pagable |

Application Flow

Client sends request

- Controller receives request ? validates input.
- Service performs logic (create, fetch, update, delete).
- Repository communicates with database.
- Response is returned as JSON.
- If error then handled by Globalhandler and sends custom error response.

Exception Handling

Custom global exception handling using `@ControllerAdvice` + `@ExceptionHandler`.

Handled exceptions:

- `EmployeeNotFoundException`
- `ArithmeticException` (demo divide-by-zero case)
- `Exception` —will return the what it get then return this exception
- Returns custom JSON error responses

Validation

Applied at db level:

- `@NotBlank` — name required
- `@Min(10000)` — minimum salary

- `@Min(0)` — minimum experience
- `@Null / @NotNull` — based on create vs update scenario

validated automatically through `@Valid` in controller.

If validation fails, it will automatically returns 400 Bad Request.

Logging

Using `@Slf4j`:

```
log.info("Fetching employees");
log.warn("Deleting employee");
log.error("Test error");
```

we can track without using print statements.

Swagger

URL:

`http://localhost:8080/swagger-ui/index.html`

Swagger shows:

- All endpoints
- Shows all endpoints with input/output formats.

Swagger Security

- Added support for Basic Auth inside Swagger UI, so I can test secure endpoints without Postman login every time.
- Creates and customizes the main OpenAPI documentation object.
- using the `addsecurity` method i tell all swagger ui to all endpoint require the basic authentication then the Authorize button will comes.
- Then i define the details of the scheme

Security

Features

- Role-based access (ADMIN, USER)
- Basic authentication
- Password encoding (BCrypt)
- **Admin auto-creation on startup**
 - Username: `admin`
 - Password: `123`

Access Rules

| Endpoint | Access |
|-------------------------------|---------------|
| /swagger-ui/*, /v3/api-docs/* | Public |
| /signup | ADMIN only |
| /emp/** | ADMIN + USER |
| Others | Auth required |

Implementation

- Disabled CSRF
- SecurityConfig configures rules for the role field
- UserDetailsService implemented to load user from DB
- only the Encoded passwords stored in DB
- Default role = USER if not provided

Password Encoding

- Used here BCryptPasswordEncoder() so passwords are encoded before saving.
- In db also encoded password is saved

User Entity

- Created a UserEntity table with fields (id, username, password, role)

Repository

- Repository with queries to filter by username & role and also list all the users

UserService

- I implemented UserDetailsService and give my custom implementation for the loadUserByUsername
- This is the method Spring Security calls during the login process.
- Here i try to find the user by username if found store it in user if not then throws a userNotFoundException
- then finally i converted the userentity to spring security userDetails obj which has contain all info about users like username, password, role.

AuthController

- Implemented UserDetailsService to tell Spring Security how to load users from DB
- **/signup** endpoint to register users
- If no role given then defaults to USER as a role
- **/getalluser** lists all regular users

- Only ADMIN can create users with /signup

Pagination & Sorting

In controller:

- Accept page, size, dir, sortBy as request params

In service:

- Create Sort object
- Create Pageable using PageRequest.of()
- Execute JPA query: findAll(Pageable)
- Spring Data JPA internally use the LIMIT and OFFSET.

Response contains:

- Current page
- Page size
- Total pag
- Total records
- Paged employee list