

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 handling

all the dependencies are added in pom.xml.

Annotations

Annotation

@RestController
@Autowired
@Entity
@Id, @GeneratedValue
@GetMapping, @PostMapping,
@PatchMapping, @DeleteMapping, @PutMapping
@RequestBody
@PathVariable, @RequestParam
@Valid, @Validated
@NotBlank, @Min, @NotNull, @Null
@Service, @Repository
@ControllerAdvice
@ExceptionHandler
@Slf4j
@Tag
@Hidden

Purpose

REST controller returning JSON
Inject dependencies
Maps class to DB table
Primary key and auto-generation
Map HTTP methods
Read JSON body
Read URL path / query values
Enable validation in our controller
Field-level validation
Service and repository layers
Global exception handler
Handle specific exceptions
Logging
Swagger controller tag
Hidde this controler in the swagger ui

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 |
| PATCH | / change {id} / {sal} | Change the employee's sal by ge thier id | Id,new sal | Employee updated salary |

Application.Properties

Here i did the following things

- Sets the application's name.
- This applicatio will run on 8082 port .
- I printed all generated SQL statements to the console.
- I Specifies the connection path to the MySQL database named employeeDev
- The database details like login username, and password
- Specifies the MySQL JDBC driver.

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 feild
- 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 feilds (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()
- Executed the 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

MySql Validations

- Checked tables in employeeDev schema.
- Confirmed the EmployeeEntity and UserEntity tables were created
- run the query `SELECT * FROM user_entity, SELECT * FROM employee_entity` in Workbench.
- Verified that in DataConfig has method `initialuser` executed successfully, inserting the `admin` user record.
- Confirmed the password field contains the expected BCrypt-encoded values instead of pure passwords.

API Testing using the Postman

- Run and test all the api endpoint using the postman
- Validates the security implementation.
- Tested all endpoints return expected status codes.
- and collected it as a json file

Version control using github

GithubLink:[[project link](#)]

- Initialized a new local Git repository in your my project directory.
- Moved files from the working directory to the staging area.
- Confirmed the current state of the repository before committing.
- Created the commit message before committed.
- configured my identity in github.
- then i connected my local repo with remote repo.
- And i managed code by branches like create ,viewed ,switched .
- If any conflict happens when two branches have modified the same lines of code in the same file then comes in my eclipse it shows where the conflict occurs
- then chose which one i wanted .
- Then saved it and committed to git.
- If i want to merge only a specific commit from one branch onto your current branch, i used `git cherry-pick` command.