Vacation Request System

Daniel Evaristo Escalera Bonilla

17/09/2024

1. Introduction

- **Project Overview**:

  As part of an employee management system it is necessary to create a system for the employees to request vacation time for different reasons, to make this process easy the employee should be able to easily submit requests for an Human resources employee to see and approve or decline the request.

- **Technologies Used**:

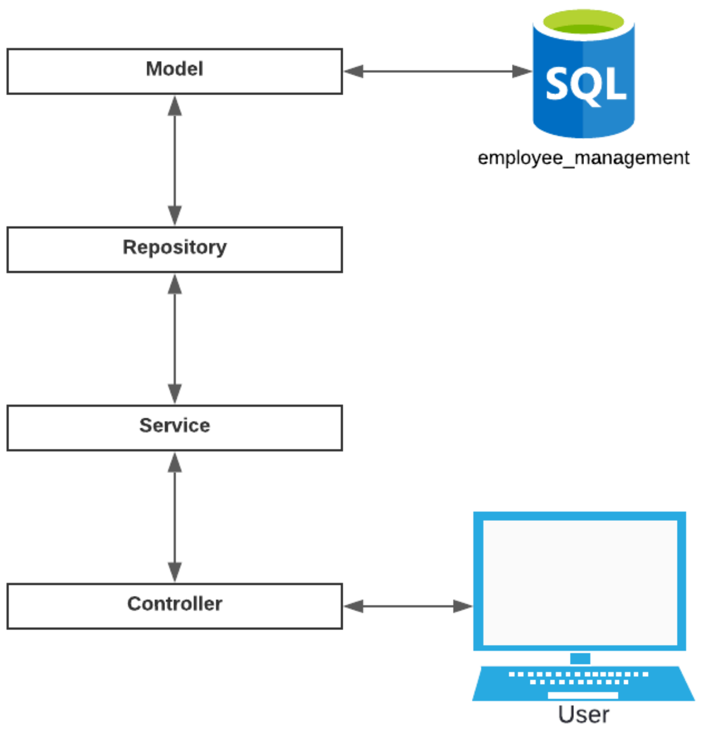
  - **Java**: The application is built with Java.
  - **Spring Boot 3.3.3**: The core framework used for creating the application, including Spring Boot modules like:
    - **Spring Boot Starter Batch**: For batch processing tasks.
    - **Spring Boot Starter Data JPA**: For working with relational databases using JPA.
    - **Spring Boot Starter Security**: For securing the application.
    - **Spring Boot Starter Web**: For building web applications, including RESTful services.
    - **Spring Security:** For endpoint protection
  - **MySQL**: The application uses MySQL as the relational database.
  - **Lombok**: Used to reduce boilerplate code (optional).
  - **Maven**: The build tool used to manage dependencies and the build process.
  - **RabbitMQ**: A message broker used for messaging and communication between components

- **Prerequisites**:

  - Java Development Kit (JDK): Version 17 or higher.
  - Maven: For managing project dependencies and building the application.
  - MySQL Database: MySQL server needs to be installed and configured. The application expects a MySQL database instance running, as well as the correct credentials.
  - Spring Boot DevTools: To enable live reload and hot swapping in the development environment (optional).
  - Lombok Plugin: Ensure that Lombok is installed and configured in your IDE (optional).
  - MySQL Connector: The JDBC driver used to connect to the MySQL database (automatically included via Maven).

2. Architecture Overview

- **System Architecture Diagram**:



- **Key Components**:
  - **Models**: The models were implemented using lombok to create entities that represent the tables on the database, the attributes of the classes represent the columns of the table to access
  - **Repositories**: The repositories created using spring data jpa with the JPArepository interface create and implement common methods for the entities, for example getByAttribute methods depending on the attributes of the object.
  - **Services**: These classes define the business logic, make use of the repositories' methods to get and save elements of the database and their own logic to make the necessary changes or configurations of these elements.
  - **Controllers**: Classes that define the endpoints to access the service methods.

3. Setup and Installation

- **Clone the Repository**: The code repository can be found on the folder "Semana 5" of the following link: https://github.com/Daniel-Ev-Esc/proyectosAcademiaJava.git

  To clone the repository run the command:

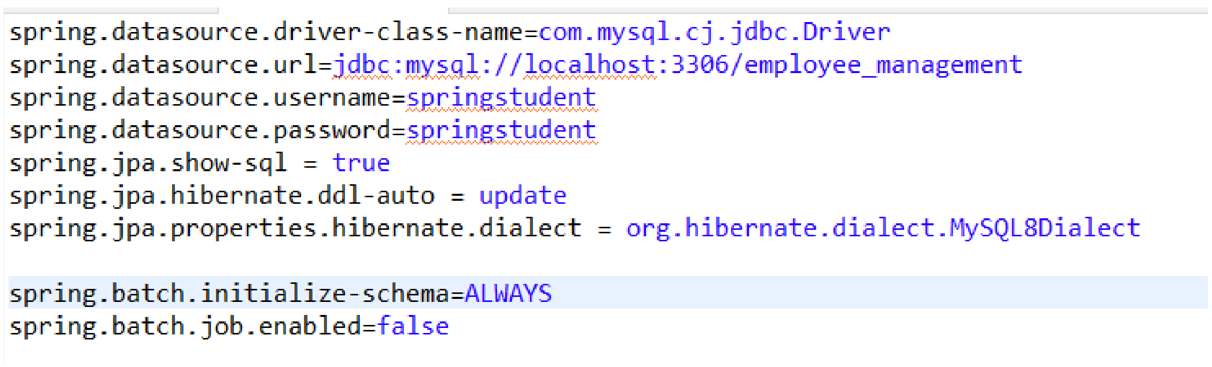
  ```
  git clone "https://github.com/Daniel-Ev-Esc/proyectosAcademiaJava.git"
  ```

- **Installation Steps**:

The projects are maven projects that contain all dependencies on the pom file.

- Environment:
    - Database: Define the url of the database, the user and their password
    - Server port: Defines the port in which the springboot application runs, used to call the endpoints.
    - Spring batch: initialize schema to create the necessary tables to run batch jobs and disable the automatic jobs

  Some projects may have less or more variables depending on the tools used.



```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/employee_management
spring.datasource.username=springstudent
spring.datasource.password=springstudent
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

spring.batch.initialize-schema=ALWAYS
spring.batch.job.enabled=false
```

- Database setup (how to initialize the database):
    - The applications create the necessary tables for the endpoints to work, but the database needs to exist already, to create it, run the next command in sql.

      CREATE DATABASE employee_management;

- Running the applications:

  When running the projects, a server will be started by each one of them in the following ports

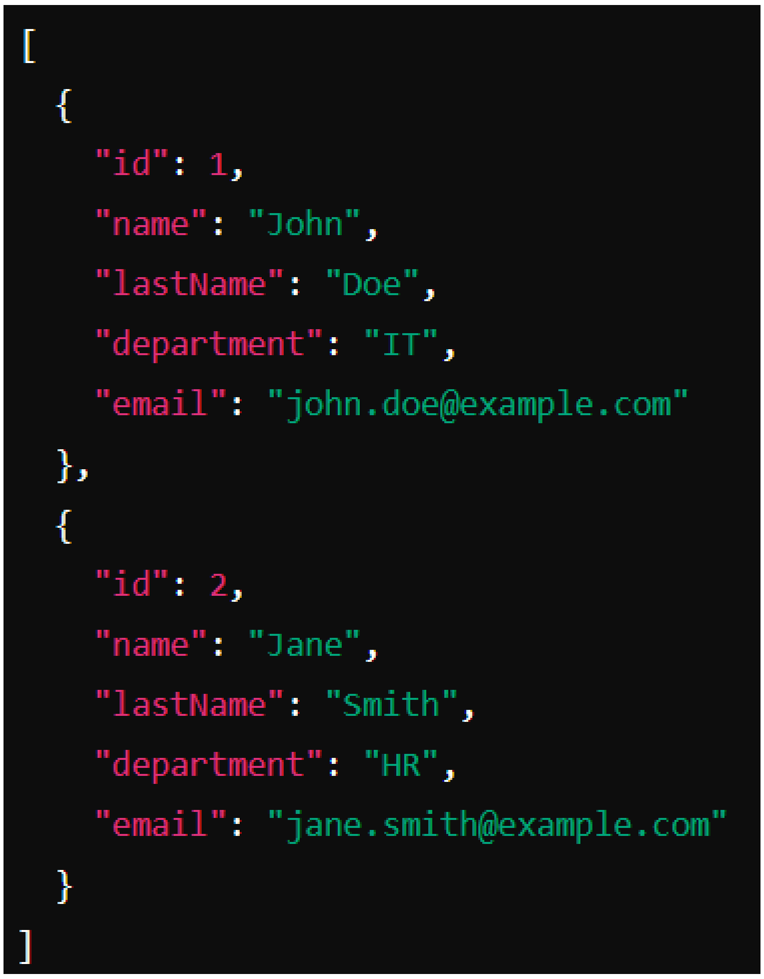
    - Main app (JPA): 8080
    - Batch: 9192
    - Mq-sender: 9000
    - Mq-reciever: 9001

4. API Documentation (If Applicable)

- **Authentication**: The requests are secured by spring security, every endpoint can be accessed with the default username and password using basic authentication, this account can then create users with the role of Employees or HR to activate specific endpoints.

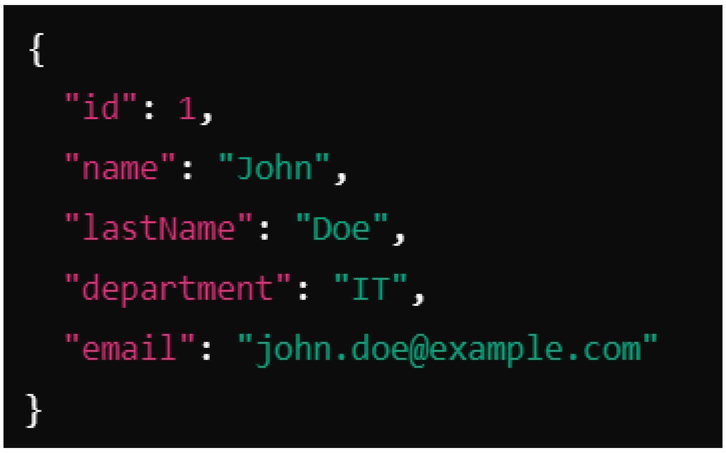
- **Endpoints**:
- **localhost:8080**
    - Endpoint: “/employees”
        - Method: GET
        - Request: No request body or query parameters.

        - Response:
        - Success: Returns a list of all employees.



```
[
  {
    "id": 1,
    "name": "John",
    "lastName": "Doe",
    "department": "IT",
    "email": "john.doe@example.com"
  },
  {
    "id": 2,
    "name": "Jane",
    "lastName": "Smith",
    "department": "HR",
    "email": "jane.smith@example.com"
  }
]
```

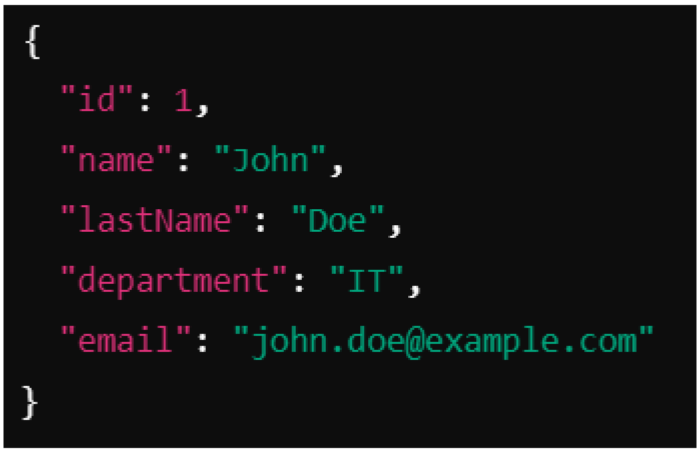
    - Endpoint: /employees/{id}
        - Method: GET
        - Request:
            - Path Variable: id - The ID of the employee to retrieve.
        - Response:
            - Success: Returns the employee with the specified ID.



```
{
  "id": 1,
  "name": "John",
  "lastName": "Doe",
  "department": "IT",
  "email": "john.doe@example.com"
}
```

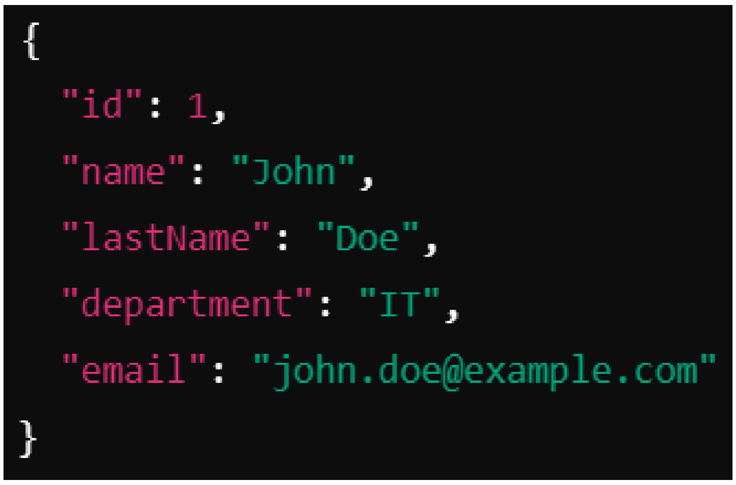
            - Error: 404 Not Found: If no employee is found with the specified ID.

- Endpoint: /employees/get-profile/{email}
    - Method: GET
    - Request:
        - Path Variable: email - The email of the employee to retrieve.
    - Response:
        - Success: Returns the employee with the specified email.



```json
{
  "id": 1,
  "name": "John",
  "lastName": "Doe",
  "department": "IT",
  "email": "john.doe@example.com"
}
```

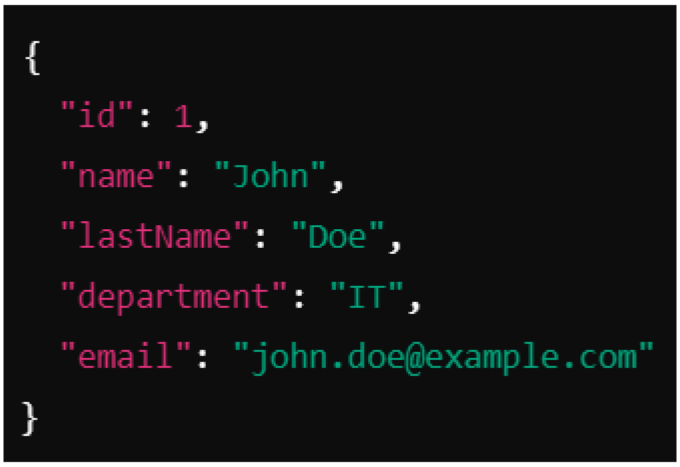
        - Error: 404 Not Found: If no employee is found with the specified ID.
- Endpoint: /employees
    - Method: POST
    - Request:
        - Request Body:



```json
{
  "id": 1,
  "name": "John",
  "lastName": "Doe",
  "department": "IT",
  "email": "john.doe@example.com"
}
```

    - Response:
        - Success: Returns the created employee with a generated ID.
- Endpoint: /employees
    - Method: PUT
    - Request:

            - Request Body:



```
{
  "id": 1,
  "name": "John",
  "lastName": "Doe",
  "department": "IT",
  "email": "john.doe@example.com"
}
```

        - Response:
            - Success: Returns the updated employee.
            - Error: **404 Not Found**: If no employee is found with the specified ID.
    - Endpoint: /employees/{id}
        - Method: DELETE
        - Request:
            - Path Variable: id - The ID of the employee to delete.
        - Response:
            - Success: No content is returned; status code 204.
            - Error: 404 Not Found: If no employee is found with the specified ID
    - Endpoint: /hr/employees
        - Method: GET
        - Request: No request body or query parameters.
        - Response:

            - Success: Returns a list of all HR employees.

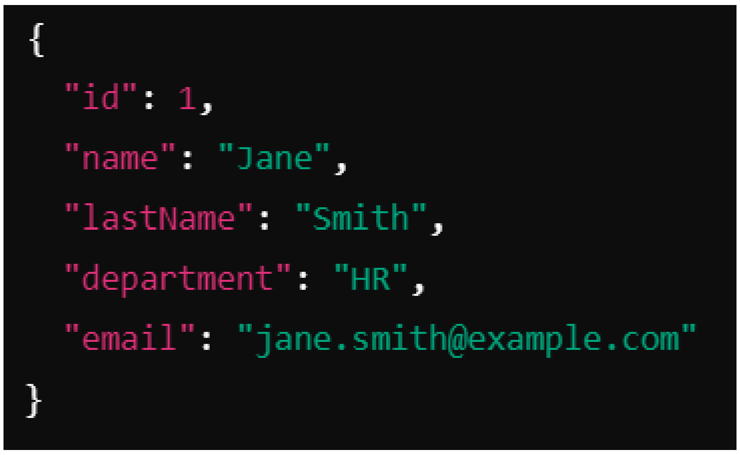
```
[
  {
    "id": 1,
    "name": "Jane",
    "lastName": "Smith",
    "department": "HR",
    "email": "jane.smith@example.com"
  },
  {
    "id": 2,
    "name": "John",
    "lastName": "Doe",
    "department": "HR",
    "email": "john.doe@example.com"
  }
]
```

    - Endpoint: /hr/employees/{id}
        - Method: GET
        - Request:
            - Path Variable: id - The ID of the HR employee to retrieve.
        - Response:
            - Success: Returns the HR employee with the specified ID.

```
{
  "id": 1,
  "name": "Jane",
  "lastName": "Smith",
  "department": "HR",
  "email": "jane.smith@example.com"
}
```

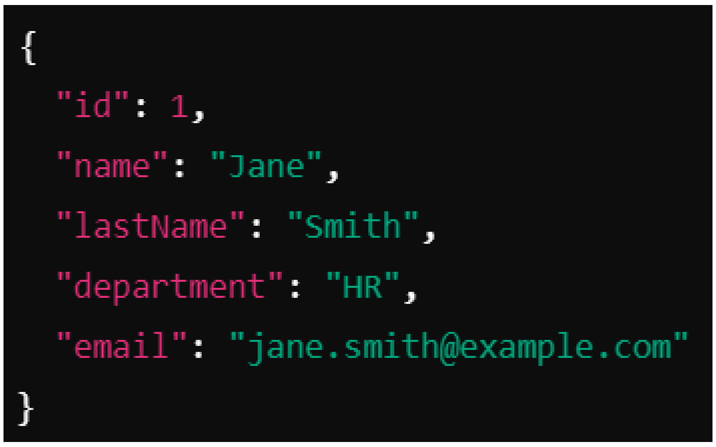
            - Error: 404 Not Found: If no HR employee is found with the specified ID.
    - Endpoint: /hr/employees/get-profile/{email}

        - Method: GET
        - Request:
            - Path Variable: email - The email of the HR employee to retrieve.
        - Response:
            - Success: Returns the HR employee with the specified email.



```json
{
  "id": 1,
  "name": "Jane",
  "lastName": "Smith",
  "department": "HR",
  "email": "jane.smith@example.com"
}
```

            - Error: 404 Not Found: If no HR employee is found with the specified email.
    - Endpoint: /hr/employees
        - Method: POST
        - Request:
            - Request Body:



```json
{
  "id": 1,
  "name": "Jane",
  "lastName": "Smith",
  "department": "HR",
  "email": "jane.smith@example.com"
}
```

        - Response:
            - Success: Returns the created HR employee with a generated ID
    - Endpoint: /hr/employees
        - Method: PUT
        - Request:

            - Request Body:

```
{
  "id": 1,
  "name": "Jane",
  "lastName": "Smith",
  "department": "HR",
  "email": "jane.smith@example.com"
}
```

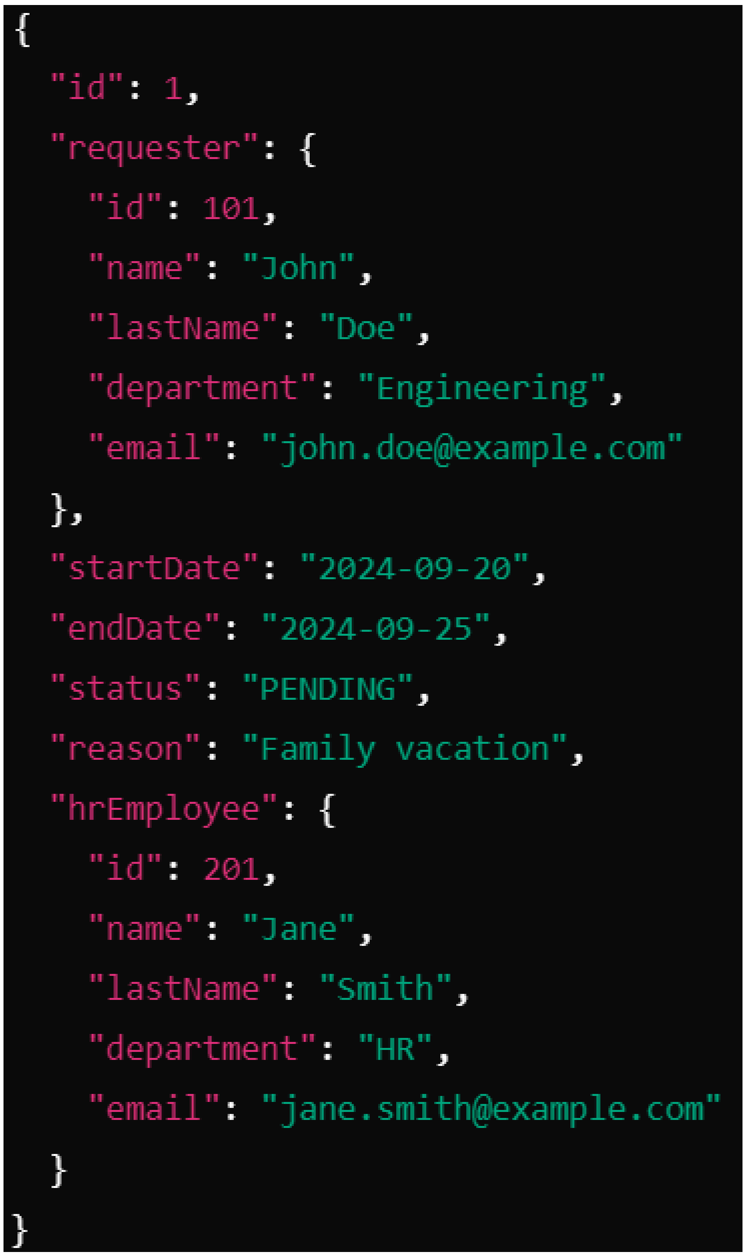
        - Response:
            - Success: Returns the updated HR employee.
    - Endpoint: /hr/employees/{id}
        - Method: DELETE
        - Request:
            - Path Variable: id - The ID of the HR employee to delete.
        - Response:
            - Success: No content is returned; status code 204.
            - Error: 404 Not Found: If no HR employee is found with the specified ID.
    - Endpoint: /messages
        - Method: GET
        - Request: No request body or query parameters.
        - Response:
            - Success: Returns a list of all custom messages

```
[
  {
    "id": "msg1",
    "content": "Welcome to the system!",
    "dismissed": false,
    "date": "2024-09-17T10:00:00"
  },
  {
    "id": "msg2",
    "content": "Your request has been approved.",
    "dismissed": true,
    "date": "2024-09-16T14:30:00"
  }
]
```

    - Endpoint: /messages/dismiss/{id}
        - Method: PUT
        - Request:

            - Path Variable: id - The ID of the message to be marked as dismissed.
        - Response:
            - Success: Returns the updated custom message with its dismissed status set to true
            - Error: 404 Not Found: If no message is found with the specified ID.
    - Endpoint: /vacation-requests
        - Method: GET
        - Request: No request body or query parameters.
        - Response:
            - Success: Returns a list of all vacation requests.
    - Endpoint: /vacation-requests/{id}
        - Method: GET
        - Request:
            - Path Variable: id - The ID of the vacation request to retrieve.
        - Response:

            - Success: Returns the vacation request with the specified ID



```
{
  "id": 1,
  "requester": {
    "id": 101,
    "name": "John",
    "lastName": "Doe",
    "department": "Engineering",
    "email": "john.doe@example.com"
  },
  "startDate": "2024-09-20",
  "endDate": "2024-09-25",
  "status": "PENDING",
  "reason": "Family vacation",
  "hrEmployee": {
    "id": 201,
    "name": "Jane",
    "lastName": "Smith",
    "department": "HR",
    "email": "jane.smith@example.com"
  }
}
```

            - Error: 404 Not Found: If no vacation request is found with the specified ID.
    - Endpoint: /vacation-requests/create
        - Method: POST
        - Request:

            - Body: VacationRequestDTO

```
{
  "employeeId": 101,
  "startDate": "2024-09-20",
  "endDate": "2024-09-25",
  "status": "PENDING",
  "reason": "Family vacation"
}
```

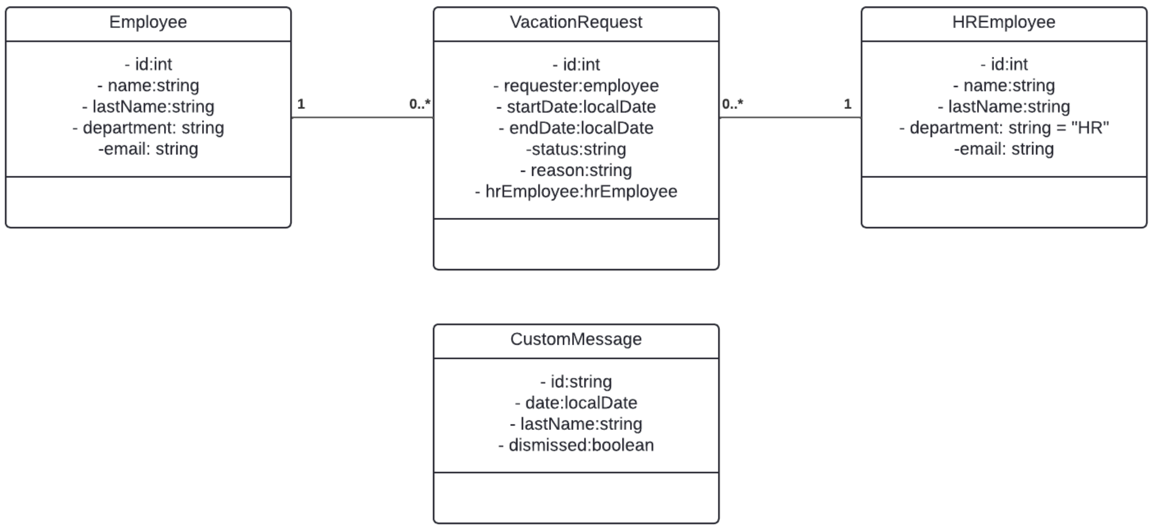
        - Response:
            - Success: Returns the created vacation request.

```
{
  "id": 1,
  "requester": {
    "id": 101,
    "name": "John",
    "lastName": "Doe",
    "department": "Engineering",
    "email": "john.doe@example.com"
  },
  "startDate": "2024-09-20",
  "endDate": "2024-09-25",
  "status": "PENDING",
  "reason": "Family vacation",
  "hrEmployee": null
}
```

    - Endpoint: /vacation-requests/accept/{vrId}
        - Method: PUT
        - Request:
            - Path Variable: vrId - The ID of the vacation request to be accepted.
        - Response:
            - Success: Returns the updated vacation request with status set to ACCEPTED
    - Endpoint: /vacation-requests/decline/{vrId}

            - Method: PUT
            - Request:
                - Path Variable: vrId - The ID of the vacation request to be declined.
            - Response:
                - Success: Returns the updated vacation request with status set to DECLINED
- localhost:9192
    - Endpoint: /jobs/completeRequest
        - Method: POST
        - Description: This endpoint triggers a Spring Batch job designed to process and update vacation requests in the database.
        - Request:
            - Body: No request body is required.
        - Response:
            - Success: The job is executed, and the vacation requests are processed and updated in the database. The endpoint returns a status of 200 OK if the job is successfully started.
- localhost:9000
    - Endpoint: /publish
        - Method: POST
        - Request:
            - Body: CustomMessage object
        - Response:
            - Success: Returns a message indicating the status of the operation

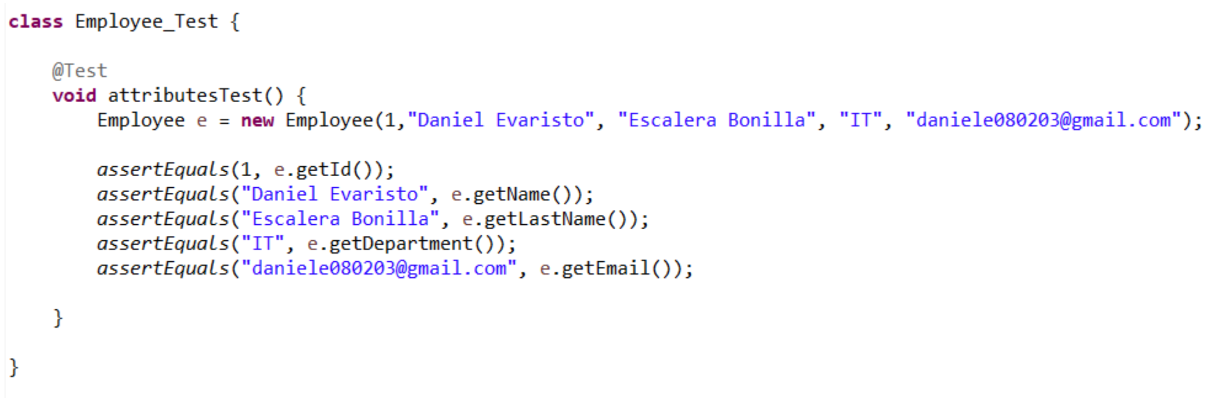
5. Database Structure



- **Tables/Collections**: An employee can create many vacation request, which can be accepted or denied by an HR Employee, the handling of the request generates a message separated from the flow of the data

9. Testing

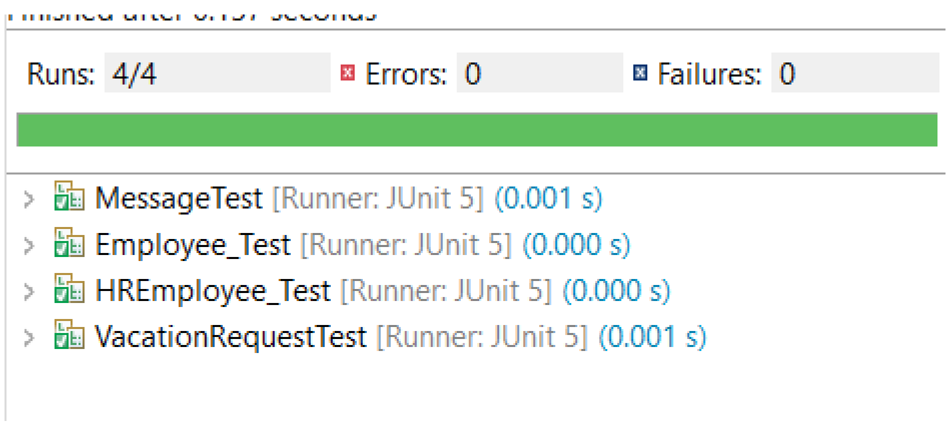
- **Test Strategy**: Overview of how the application is tested.
- **Unit Tests**: Instructions on running unit tests.
  - Entities tests: Each entity has a test to assert the correct creation of the entity using its constructor here is an example with the employee class:



```java
class Employee_Test {

    @Test
    void attributesTest() {
        Employee e = new Employee(1,"Daniel Evaristo", "Escalera Bonilla", "IT", "daniele080203@gmail.com");

        assertEquals(1, e.getId());
        assertEquals("Daniel Evaristo", e.getName());
        assertEquals("Escalera Bonilla", e.getLastName());
        assertEquals("IT", e.getDepartment());
        assertEquals("daniele080203@gmail.com", e.getEmail());

    }

}
```

    i. Create the object
    ii. Assert each attribute

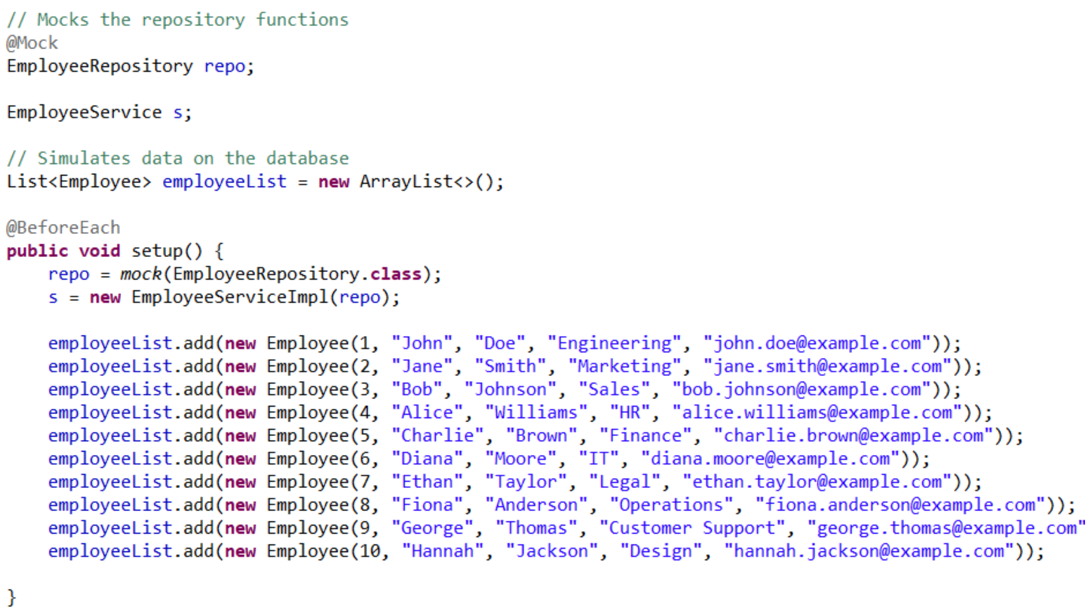
  - Results:



```
Runs: 4/4        Errors: 0        Failures: 0

> MessageTest [Runner: JUnit 5] (0.001 s)
> Employee_Test [Runner: JUnit 5] (0.000 s)
> HREmployee_Test [Runner: JUnit 5] (0.000 s)
> VacationRequestTest [Runner: JUnit 5] (0.001 s)
```

  - Service tests (This section explains the universal service tests that use the CRUD methods of the service):

i. Setup:

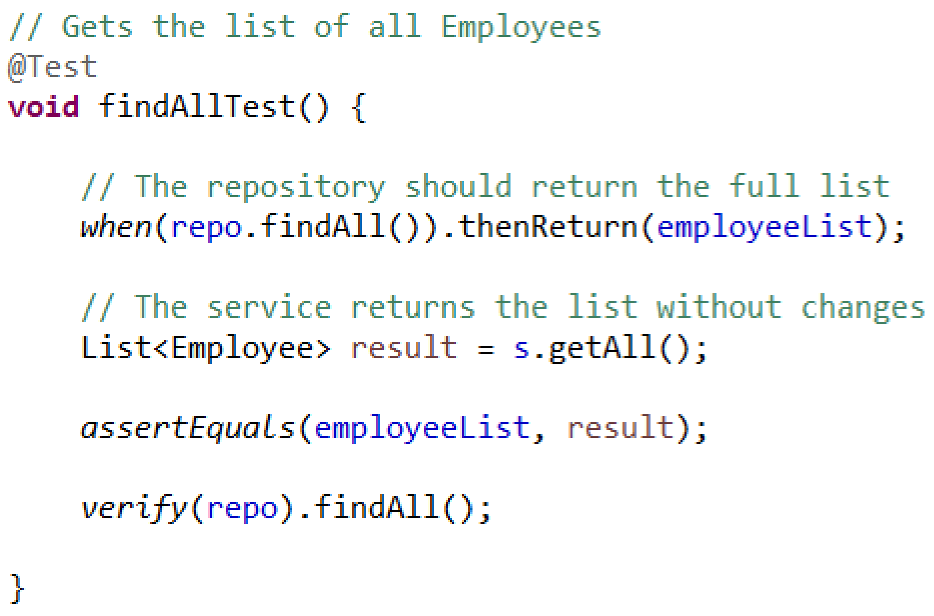


```
// Mocks the repository functions
@Mock
EmployeeRepository repo;

EmployeeService s;

// Simulates data on the database
List<Employee> employeeList = new ArrayList<>();

@BeforeEach
public void setup() {
    repo = mock(EmployeeRepository.class);
    s = new EmployeeServiceImpl(repo);

    employeeList.add(new Employee(1, "John", "Doe", "Engineering", "john.doe@example.com"));
    employeeList.add(new Employee(2, "Jane", "Smith", "Marketing", "jane.smith@example.com"));
    employeeList.add(new Employee(3, "Bob", "Johnson", "Sales", "bob.johnson@example.com"));
    employeeList.add(new Employee(4, "Alice", "Williams", "HR", "alice.williams@example.com"));
    employeeList.add(new Employee(5, "Charlie", "Brown", "Finance", "charlie.brown@example.com"));
    employeeList.add(new Employee(6, "Diana", "Moore", "IT", "diana.moore@example.com"));
    employeeList.add(new Employee(7, "Ethan", "Taylor", "Legal", "ethan.taylor@example.com"));
    employeeList.add(new Employee(8, "Fiona", "Anderson", "Operations", "fiona.anderson@example.com"));
    employeeList.add(new Employee(9, "George", "Thomas", "Customer Support", "george.thomas@example.com"
    employeeList.add(new Employee(10, "Hannah", "Jackson", "Design", "hannah.jackson@example.com"));

}
```

The repository is mocked using mockito and a service is created, a list is used to act as a database for the tests

ii. Test each service method:

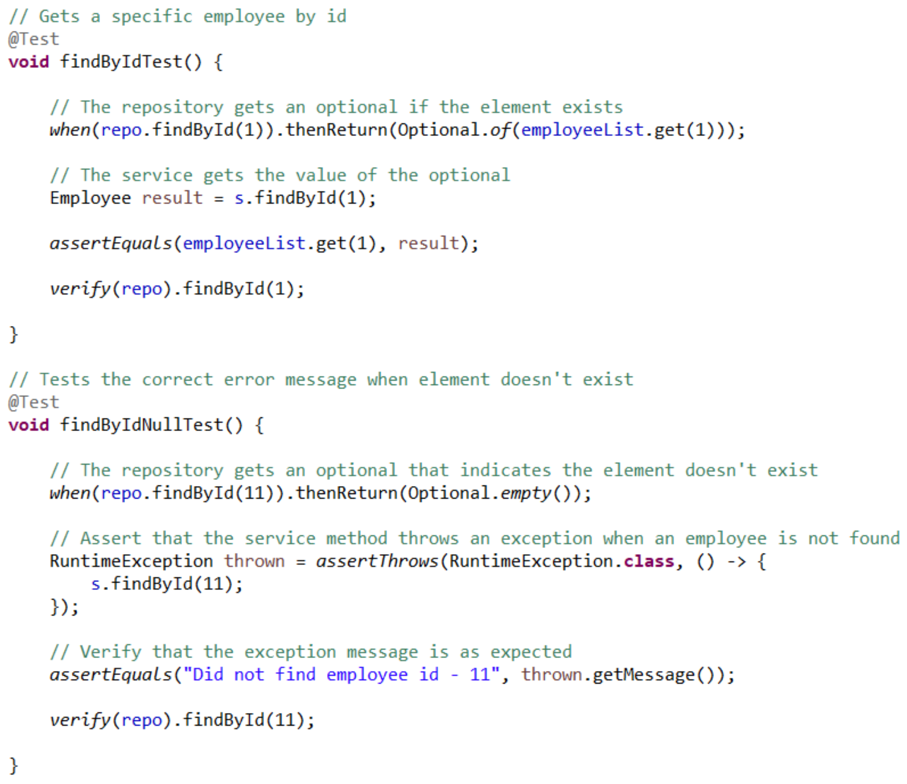
- Find all



```
// Gets the list of all Employees
@Test
void findAllTest() {

    // The repository should return the full list
    when(repo.findAll()).thenReturn(employeeList);

    // The service returns the list without changes
    List<Employee> result = s.getAll();

    assertEquals(employeeList, result);

    verify(repo).findAll();

}
```

Should return the list without modifications

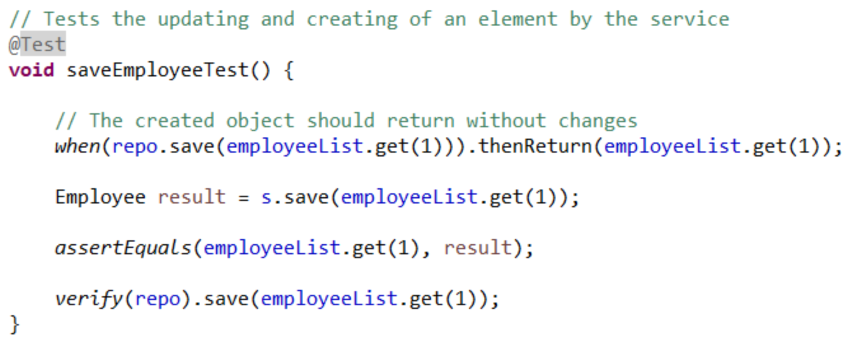
- Find by id



```java
// Gets a specific employee by id
@Test
void findByIdTest() {

    // The repository gets an optional if the element exists
    when(repo.findById(1)).thenReturn(Optional.of(employeeList.get(1)));

    // The service gets the value of the optional
    Employee result = s.findById(1);

    assertEquals(employeeList.get(1), result);

    verify(repo).findById(1);

}

// Tests the correct error message when element doesn't exist
@Test
void findByIdNullTest() {

    // The repository gets an optional that indicates the element doesn't exist
    when(repo.findById(11)).thenReturn(Optional.empty());

    // Assert that the service method throws an exception when an employee is not found
    RuntimeException thrown = assertThrows(RuntimeException.class, () -> {
        s.findById(11);
    });

    // Verify that the exception message is as expected
    assertEquals("Did not find employee id - 11", thrown.getMessage());

    verify(repo).findById(11);

}
```

Uses an element of the list to assert that the method returns said element or an error if not found

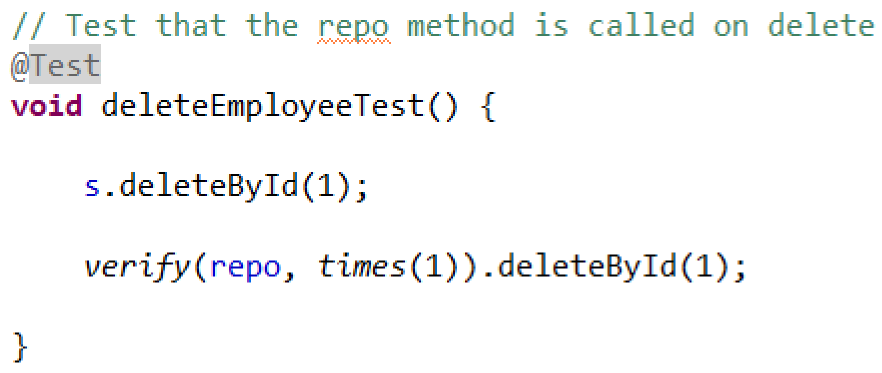
- Save



```java
// Tests the updating and creating of an element by the service
@Test
void saveEmployeeTest() {

    // The created object should return without changes
    when(repo.save(employeeList.get(1))).thenReturn(employeeList.get(1));

    Employee result = s.save(employeeList.get(1));

    assertEquals(employeeList.get(1), result);

    verify(repo).save(employeeList.get(1));
}
```

Verifies the method returns the same element

- Delete



```
// Test that the repo method is called on delete
@Test
void deleteEmployeeTest() {

    s.deleteById(1);

    verify(repo, times(1)).deleteById(1);

}
```

Asserts that the delete method is called one time.