

Employee API Documentation

Explanation:

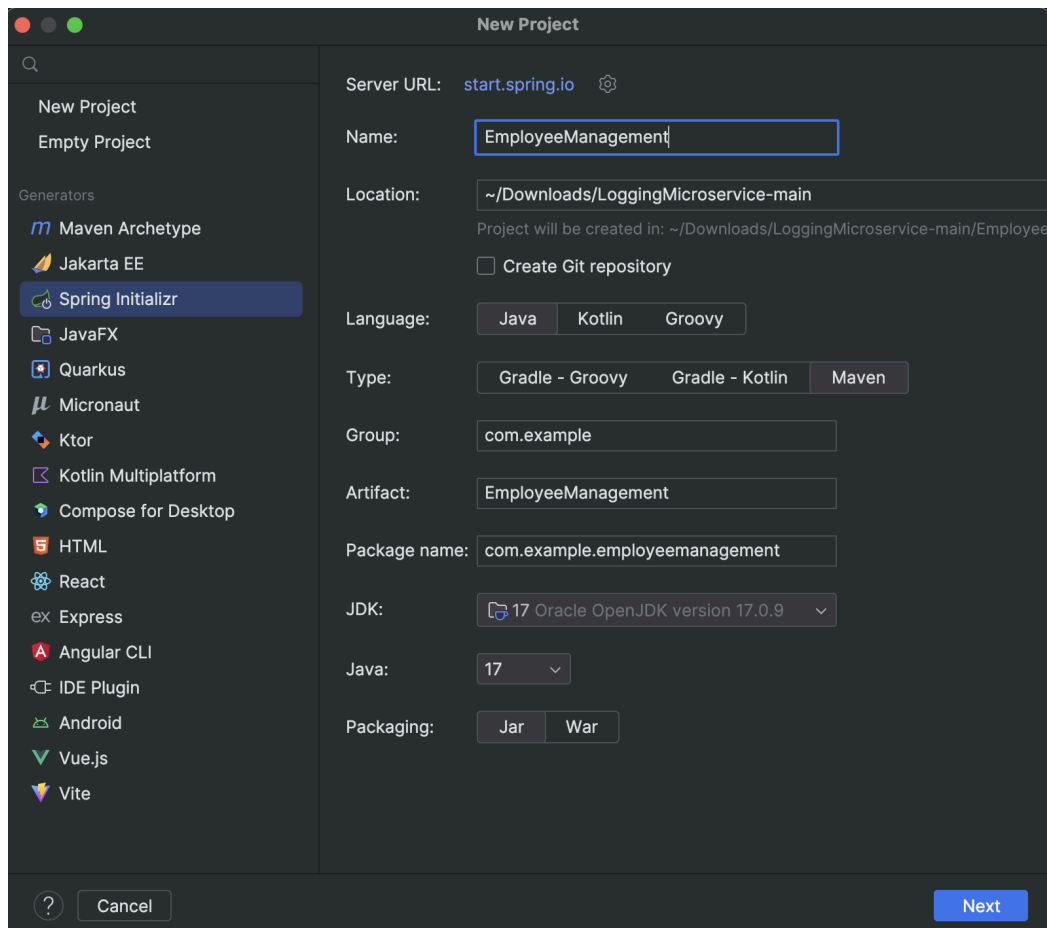
This API allows users to perform CRUD operations on employee data stored in a JSON file. The endpoints provide a straightforward way to add, retrieve, and delete employee records.

Tools Used:

- IntelliJ, or any other such IDE that supports Java and Spring
- JDK version 17.0 - LTS
- Spring Boot (<https://start.spring.io/>)
- Maven - Package Manager
- Postman - API Testing

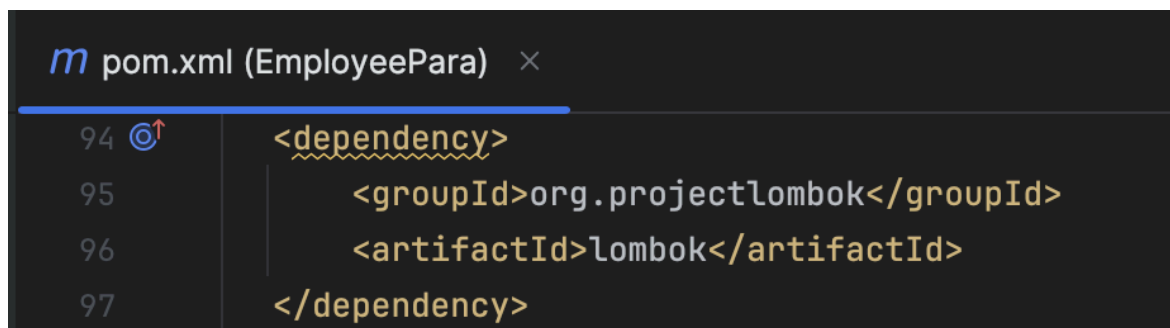
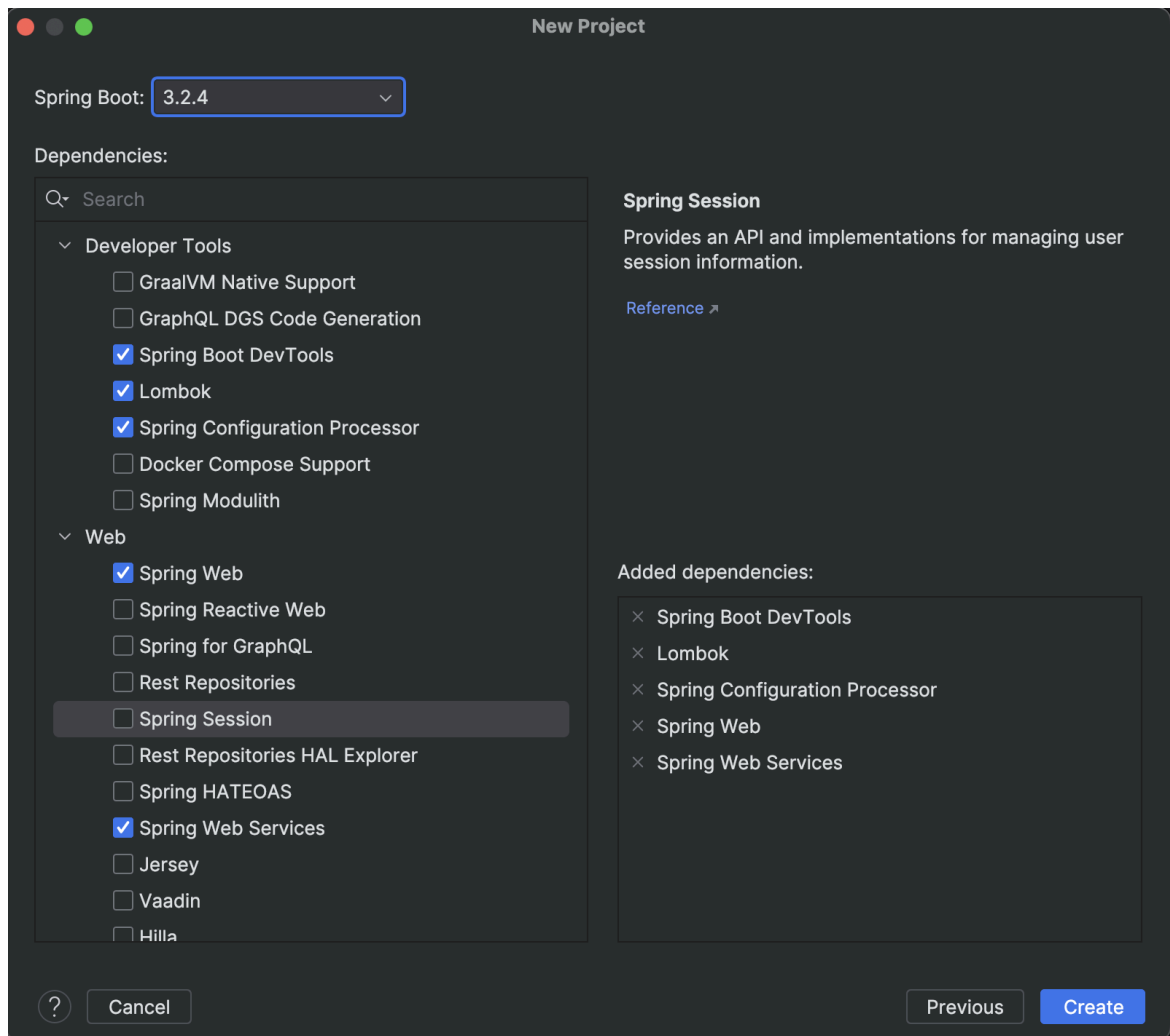
Step-By-Step Instructions:

- Download IntelliJ - <https://www.jetbrains.com/idea/download/> or any other IDE that supports Spring Development. IntelliJ makes the development experience much smoother and easier.
- Click New Project -> Select Spring Initializr -> Rename your project -> Choose Maven as the package manager (under Type) -> Choose JDK 17 if you have it, if not, install it using IntelliJ, or externally.



Employee API Documentation

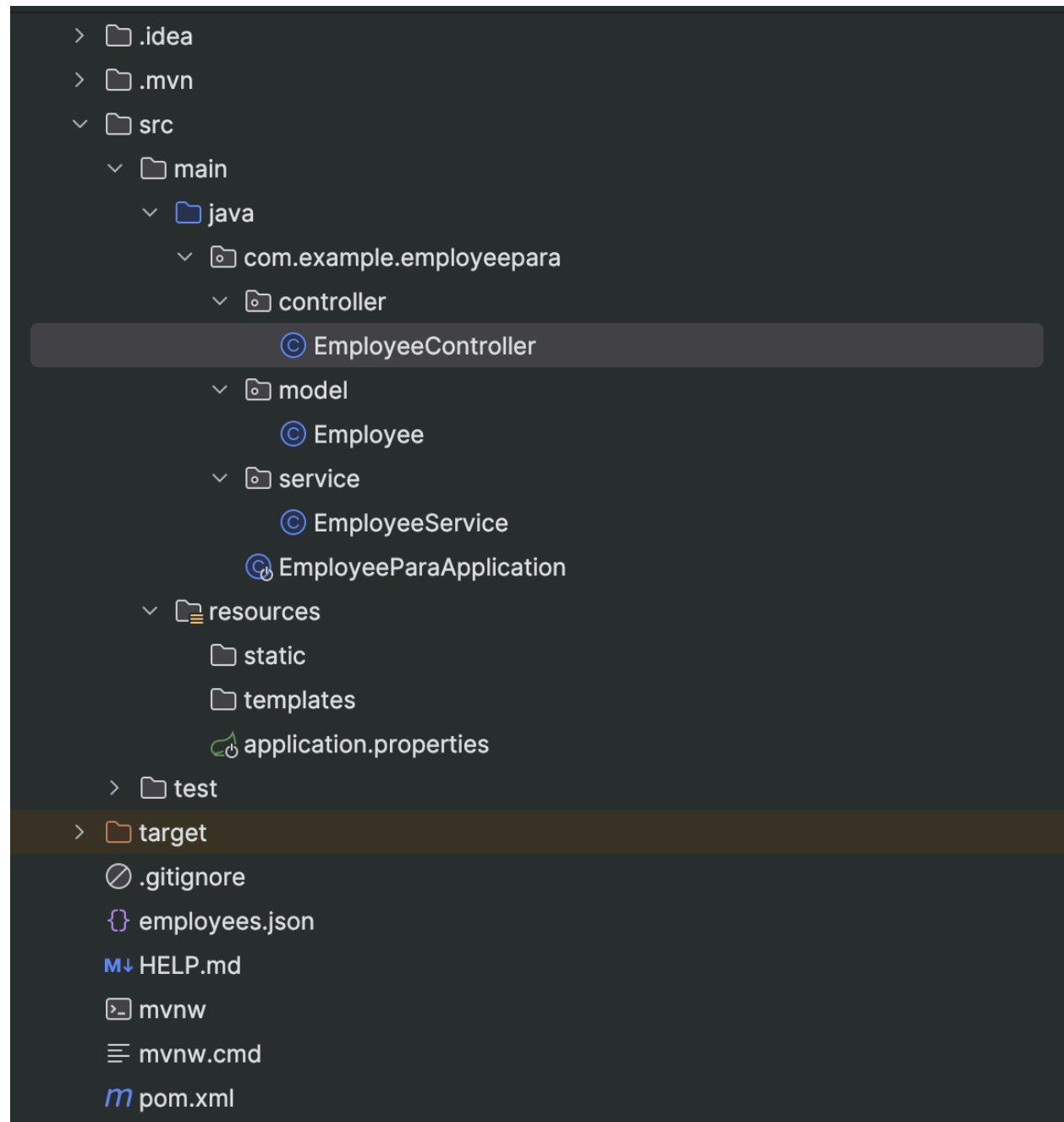
Click Next, and then click the following options, as those libraries will be automatically added to a pom.xml file, which is maven's package manager. It then allows us to do external imports of functions from those packages, such as, for storing in JSON, I have added a dependency for a package titled Lombok, which automatically defines methods such as getName() and setName(), and it's present in the pom.xml file.



Employee API Documentation

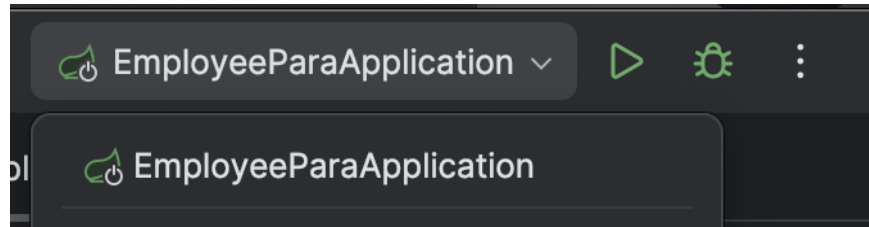
By default, you will get the src -> main -> java -> com.example.employee para out of the hood, but Spring follows an MVC architecture, that is, modules are split according to their use case, and in this, our project is split into 3 files, Controller - which handles the API request and response, model - defining the employee structure and service - with the implementation logic of how data is processed.

The employees are stored, retrieved and deleted from the employee.json file.



Employee API Documentation

To run the application, go to EmployeeParaApplication, and press run.



1. EmployeeService:

- This class manages the CRUD (Create, Read, Update, Delete) operations for employee data.
- It uses a JSON file named employees.json to persist employee records.
- The EmployeeService constructor initializes the service by reading existing employee data from the JSON file.
- The addEmployee method adds a new employee to the collection and writes the updated data back to the JSON file.
- The getEmployeeById method retrieves an employee by their ID from the collection.
- The deleteEmployeeById method deletes an employee by their ID from the collection and updates the JSON file accordingly.
- It also contains helper methods for reading and writing employee data to/from the JSON file.

2. Employee:

- This class represents the structure of an employee object.
- It contains attributes such as id, name, department, and age.
- The constructor initializes these attributes with the provided values.
- It also includes a sanitizeInput method to sanitize user input, removing potential malicious content.

3. EmployeeController:

- This class defines REST endpoints for handling employee-related requests.
- It is annotated with @RestController to indicate that it handles RESTful requests.
- The addEmployee method handles HTTP POST requests to add a new employee.
- The getEmployeeById method handles HTTP GET requests to retrieve an employee by ID.
- The deleteEmployeeById method handles HTTP DELETE requests to delete an employee by ID.
- It utilizes the EmployeeService to perform CRUD operations on employee data.
- The nested Response class is a custom response object containing a status code and a message.

Employee API Documentation

API Documentation:

Base URL: /employees

- Add Employee

- Method: POST
- Endpoint: /
- Description: Add a new employee.
- Request Body: JSON object representing the new employee.
 - Attributes:
 - name (String): Name of the employee.
 - department (String): Department of the employee.
 - age (Integer): Age of the employee.
- Response:
 - Status Code:
 - 201 Created if the employee is successfully added.
 - 500 Internal Server Error if an error occurs while adding the employee.
 - Body: JSON object containing the status code and a message.

The screenshot displays a REST client interface with the following details:

- URL:** localhost:8080/employees
- Method:** POST
- Request Body (JSON):**

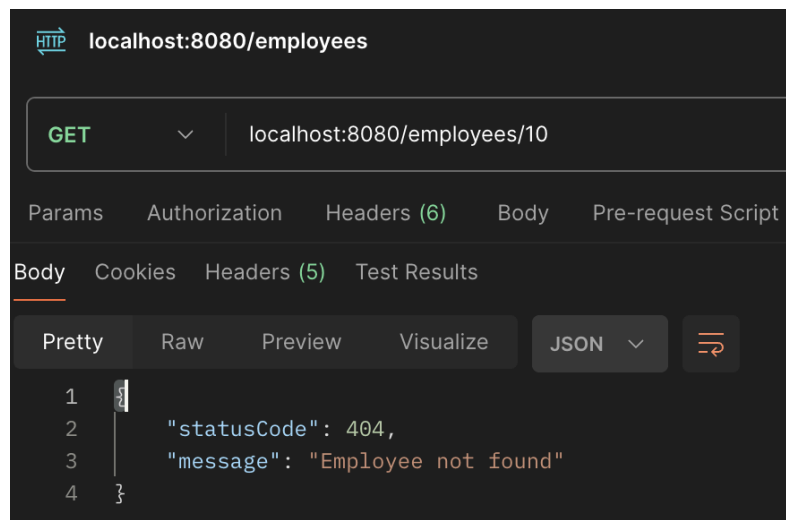
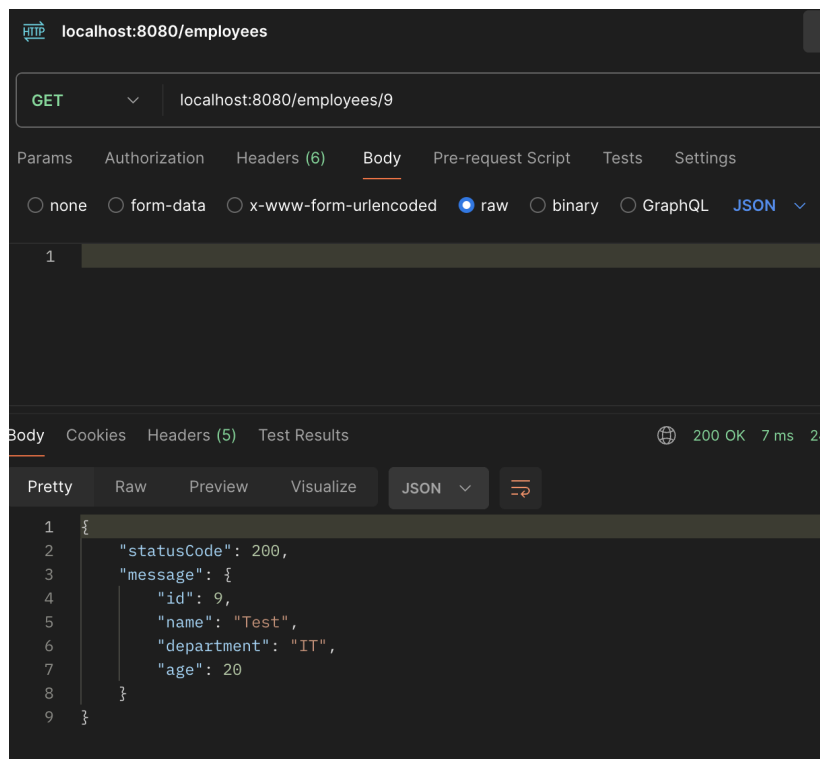
```
{
  "name": "Test",
  "department": "IT",
  "age": 20
}
```
- Response Status:** 201 Created (6 ms)
- Response Body (JSON):**

```
{
  "statusCode": 201,
  "message": {
    "id": 9,
    "name": "Test",
    "department": "IT",
    "age": 20
  }
}
```

Employee API Documentation

- Get Employee by ID

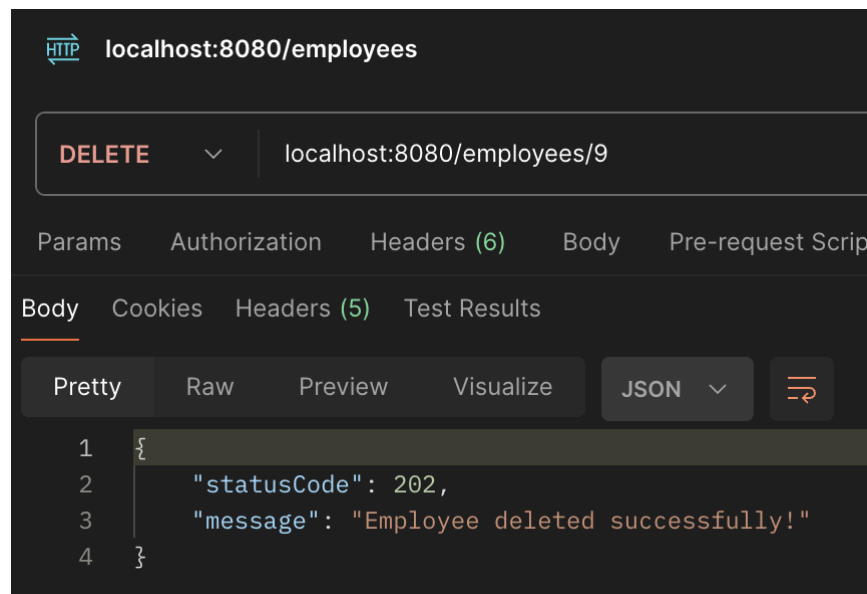
- Method: GET
- Endpoint: `/id`
- Description: Retrieve an employee by their ID.
- Path Variable:
 - id (Integer): ID of the employee to retrieve.
- Response:
 - Status Code:
 - 200 OK if the employee is found.
 - 404 Not Found if the employee is not found.
 - Body: JSON object containing the status code and the retrieved employee information.



Employee API Documentation

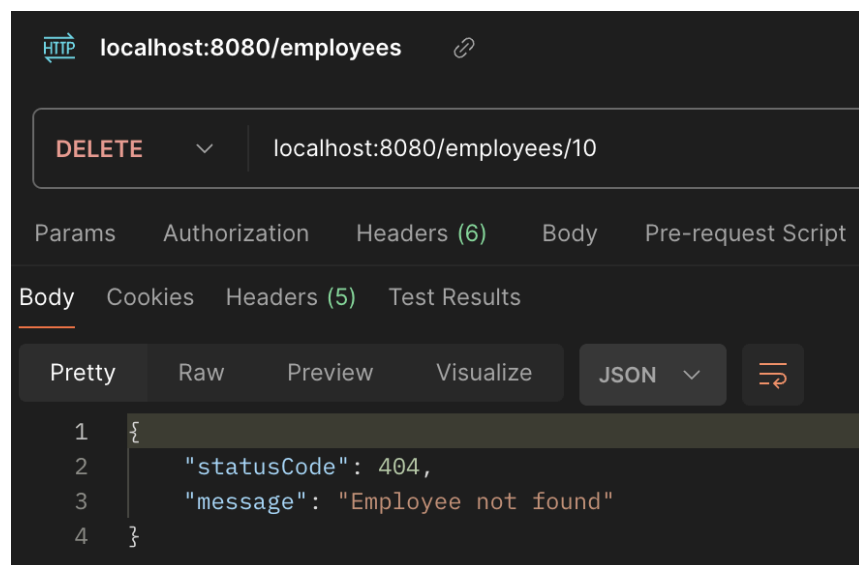
- Delete Employee by ID

- Method: DELETE
- Endpoint: `/id`
- Description: Delete an employee by their ID.
- Path Variable:
 - id (Integer): ID of the employee to delete.
- Response:
 - Status Code:
 - 202 Accepted if the employee is successfully deleted.
 - 404 Not Found if the employee is not found.
 - Body: JSON object containing the status code and a message indicating success or failure.



The screenshot shows a REST client interface with the URL `localhost:8080/employees` and the method `DELETE`. The path variable `/9` is entered. The response body is displayed in JSON format:

```
1 {  
2   "statusCode": 202,  
3   "message": "Employee deleted successfully!"  
4 }
```



The screenshot shows a REST client interface with the URL `localhost:8080/employees` and the method `DELETE`. The path variable `/10` is entered. The response body is displayed in JSON format:

```
1 {  
2   "statusCode": 404,  
3   "message": "Employee not found"  
4 }
```