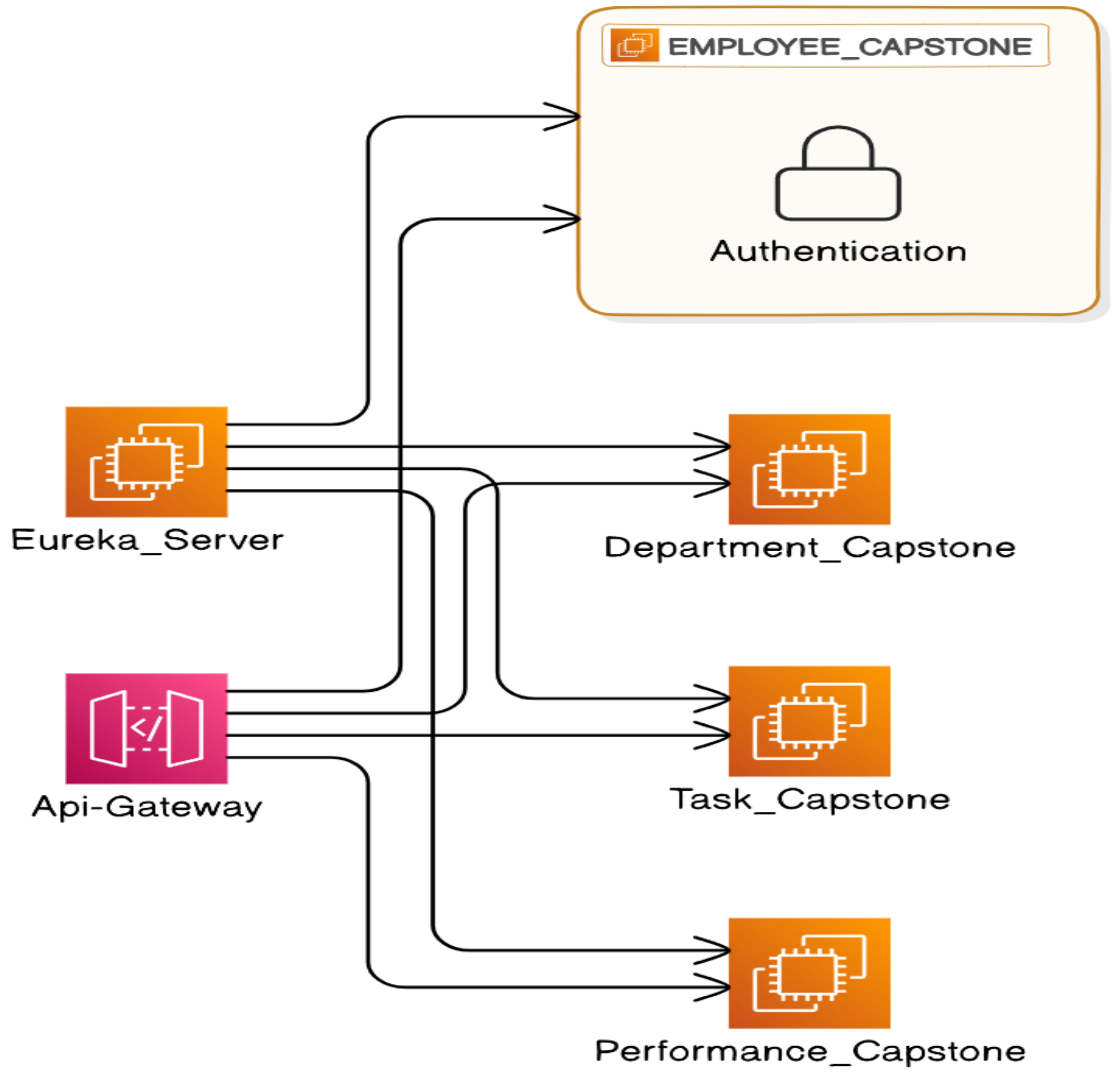
CAPSTONE PROJECT

ON

EMPLOYEE MANAGEMENT SYSTEM

Employee Management System Architecture



Introduction

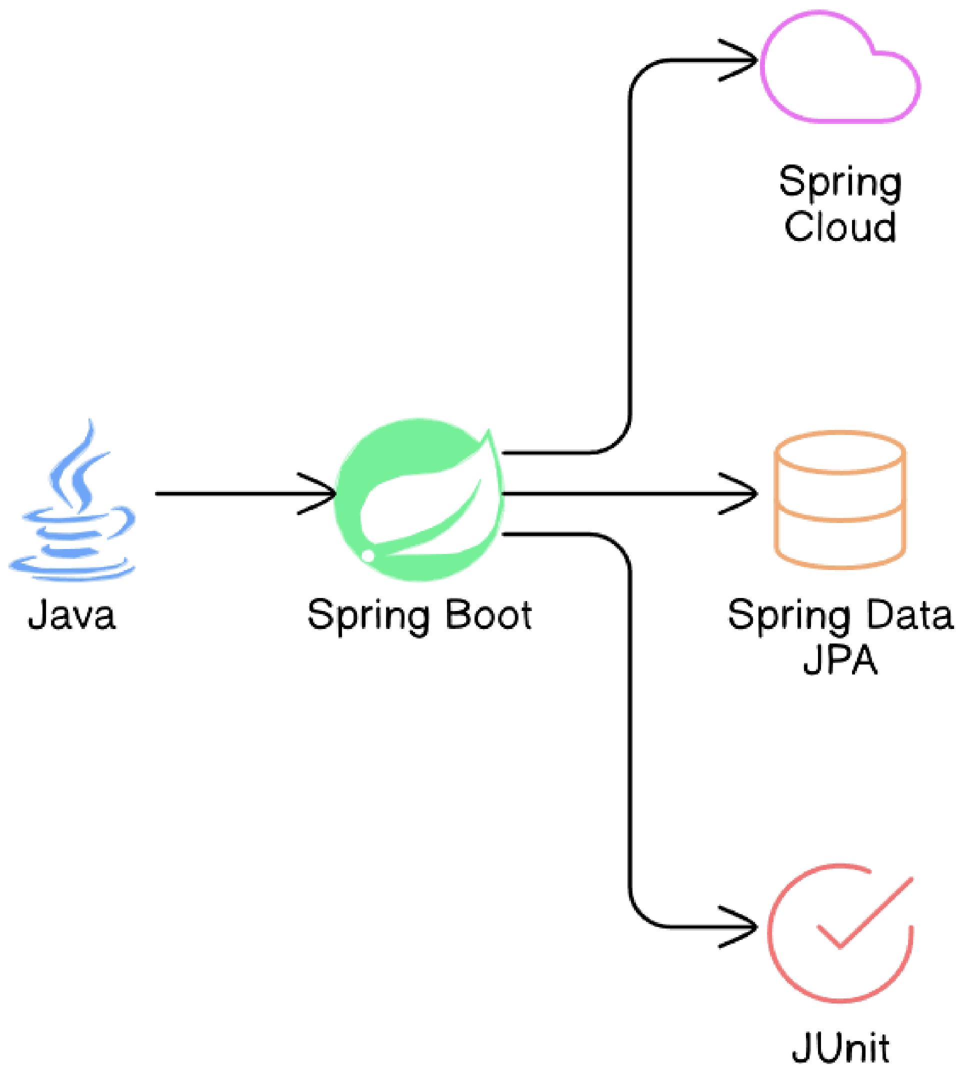
Project Overview:

- The Employee Management System is a distributed application designed to manage employee-related information within an organization. The system provides functionalities for managing employees, departments, tasks, and performance evaluations. It is built using microservices architecture, which allows for scalability and flexibility.

Technologies Used:

- **Java:** The core programming language used to develop the application.
- **Spring Boot:** A framework that simplifies the development of standalone, production-ready Spring-based applications. It provides an easy way to set up and run the application.
- **Spring Cloud:** Used to build microservices and handle cross-cutting concerns like configuration management, service discovery, circuit breakers, and distributed tracing.
- **Spring Data JPA:** A part of the Spring Data project, it simplifies data access by providing an abstraction over the database interaction.
- **MySQL/MariaDB:** The relational database management system used for storing the application's data.
- **RESTful APIs:** The architectural style used for designing networked applications and providing communication between different microservices.

Technology Used



Problem Statement

The main objective of this project is to develop an Employee Management System that meets the following requirements:

- **For Admins:**
  - Admins need a centralized system to manage employees, departments, and roles effectively. They should be able to add, update, delete, and view employee details, organize employees into departments, assign roles, manage tasks, and evaluate performance.
- **For Employees:**
  - Employees should be able to manage their profiles, view assigned tasks, update task statuses, and track their performance. They should also have access to feedback and performance evaluations from their managers.

Microservices Architecture

Microservices are an architectural style that structures an application as a collection of small, autonomous services modeled around a business domain. This approach helps in building scalable and flexible applications. Each microservice is responsible for a specific piece of functionality and communicates with other services via APIs.

A. Service Registry & Discovery

Eureka Server:
Eureka is a service registry that allows microservices to register themselves at runtime. This enables each service to discover the location of other services, making the communication dynamic and scalable.
Role: Eureka acts as a lookup service where each microservice registers, and other services can discover and communicate with them using the information provided by Eureka.

B. API Gateway

Spring Cloud Gateway:
The API Gateway serves as a single entry point for all client requests. It routes requests to the appropriate backend services based on the routing configuration.
Functions: Apart from routing, the gateway can handle cross-cutting concerns such as authentication, logging, rate limiting, and request validation.

C. Authentication Service

User Authentication & Authorization:

This service handles the authentication and authorization processes. It manages user registration and login and generates JWT (JSON Web Token) for secure communication.

Role Management: It distinguishes between different types of users (Admin and Employee) and restricts access to specific functionalities based on their roles.

D. Employee Management Microservice

Employee Directory:

This microservice is responsible for managing employee-related data. It provides APIs to create, read, update, and delete (CRUD) employee records.

Functionality: It handles storing personal information, job roles, and department assignments. It also allows for the management of employee roles within the organization.

E. Department Management Microservice

Department Catalog:

This service manages department-related information within the organization.

APIs: It provides endpoints for creating, updating, deleting, and retrieving departments. It also organizes employees within specific departments and manages department-specific data.

F. Task Management Microservice
Task Assignment:
This microservice handles the creation and assignment of tasks to employees.
APIs: It provides functionalities to create tasks, update task details, track progress, and mark tasks as complete. It is crucial for managing work distribution and tracking employee workload.

G. Performance Management Microservice
Employee Performance Tracking:
This service is designed to track and evaluate the performance of employees based on their task completion and other metrics.
APIs: It offers features for performance reviews, providing feedback, and generating evaluation reports. These reports can be used by Admins to make informed decisions about promotions, training, or other HR-related activities.

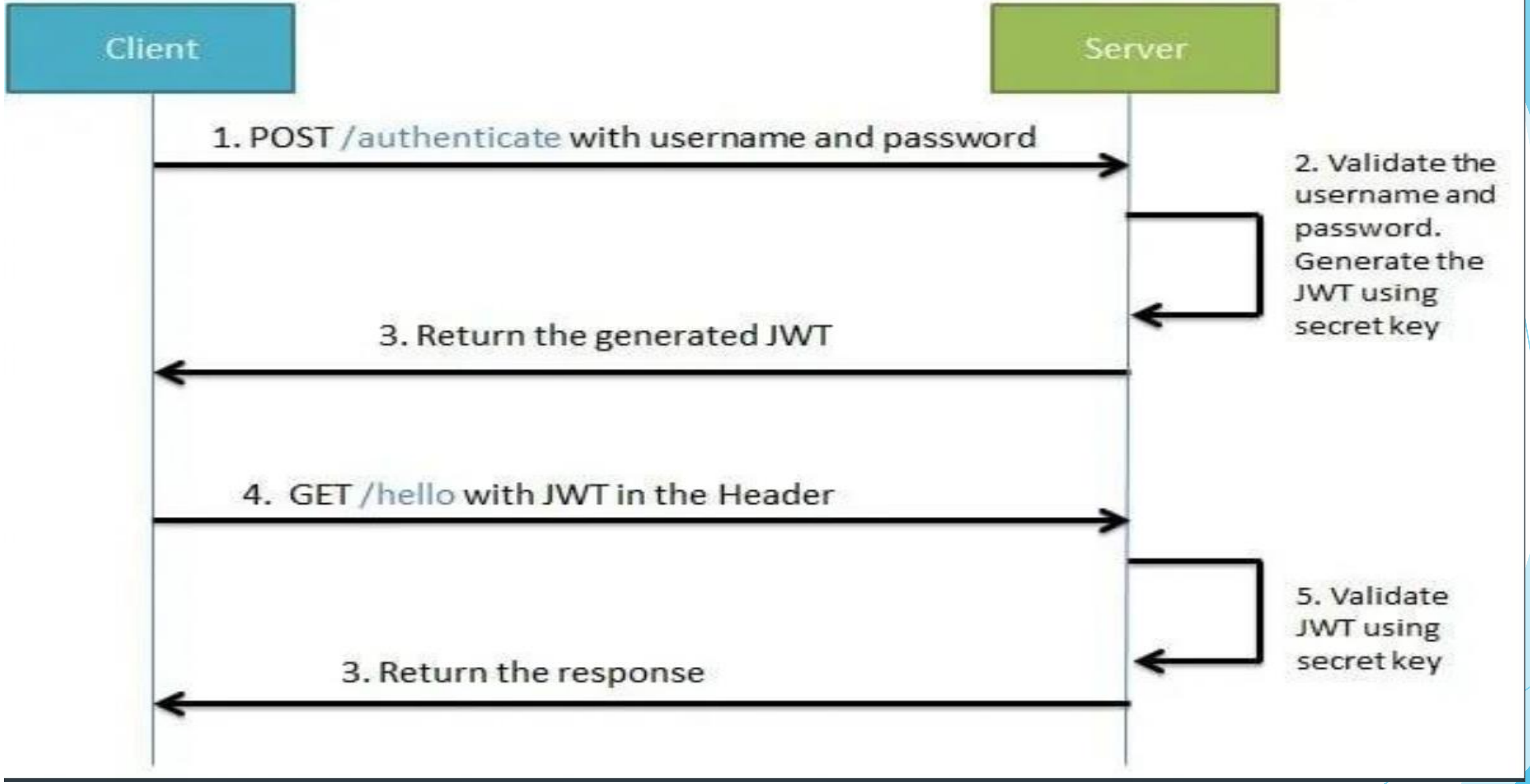
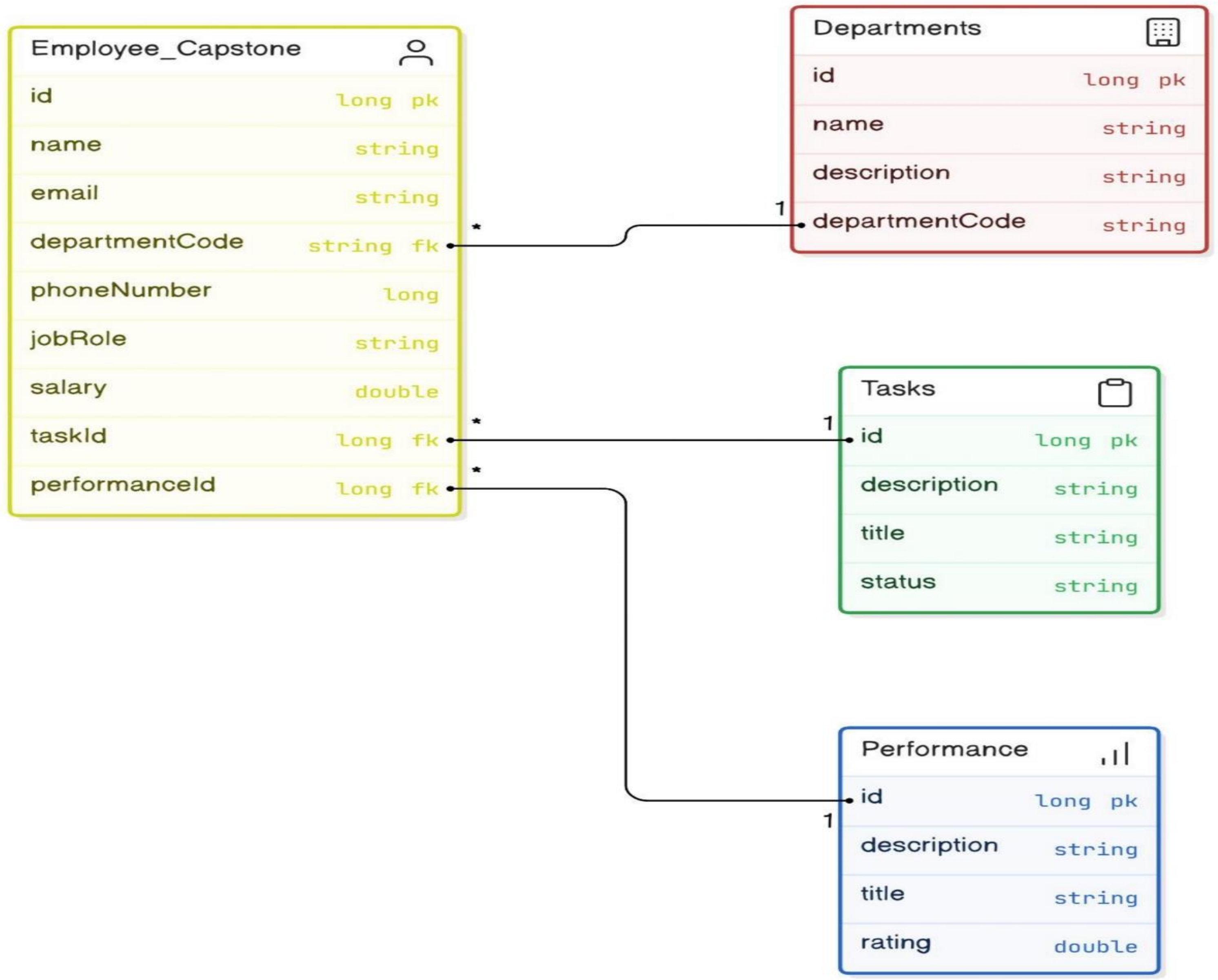

Client
Server
1. POST /authenticate with username and password
2. Validate the username and password. Generate the JWT using secret key
3. Return the generated JWT
4. GET /hello with JWT in the Header
5. Validate JWT using secret key
3. Return the response

Fig: Client-Sever Connection

Flowchart of EMS



Project Flow

Admin Module

Admin Dashboard:
The dashboard acts as a centralized interface for Admins to manage various aspects of the organization.
Features: Admins can view analytics and reports on employee performance, departmental efficiency, and other key metrics.

Employee Management:
Admins can perform CRUD operations on employees, which include adding new employees, updating existing employee details, and deleting employees when necessary.
Role and Department Assignment: Admins can assign roles to employees and organize them into relevant departments.

Department Management:
Admins can manage departments by creating, viewing, editing, or deleting department records.
Employee Assignment: Manage which employees belong to which departments.

Task Management:
Admins can create and assign tasks to employees, monitor their progress, and update the status of tasks.
Task Monitoring: Helps track the completion rate and deadlines of tasks assigned to employees.

Performance Management:
Admins can track employee performance through various metrics, conduct performance reviews, and provide feedback.
Evaluation Reports: Generate reports based on the collected performance data, which can be used for employee development.

Employee Module

Employee Registration & Authentication:
Employees can register and log in using their credentials. The authentication service secures API access using JWT tokens.
Security:
Ensures that only authenticated users can access specific parts of the system.

Profile Management:

Employees can view and update their personal profiles, which include contact information, roles, and department details.

Access to Information: View their assigned roles and departments.

Task Management:

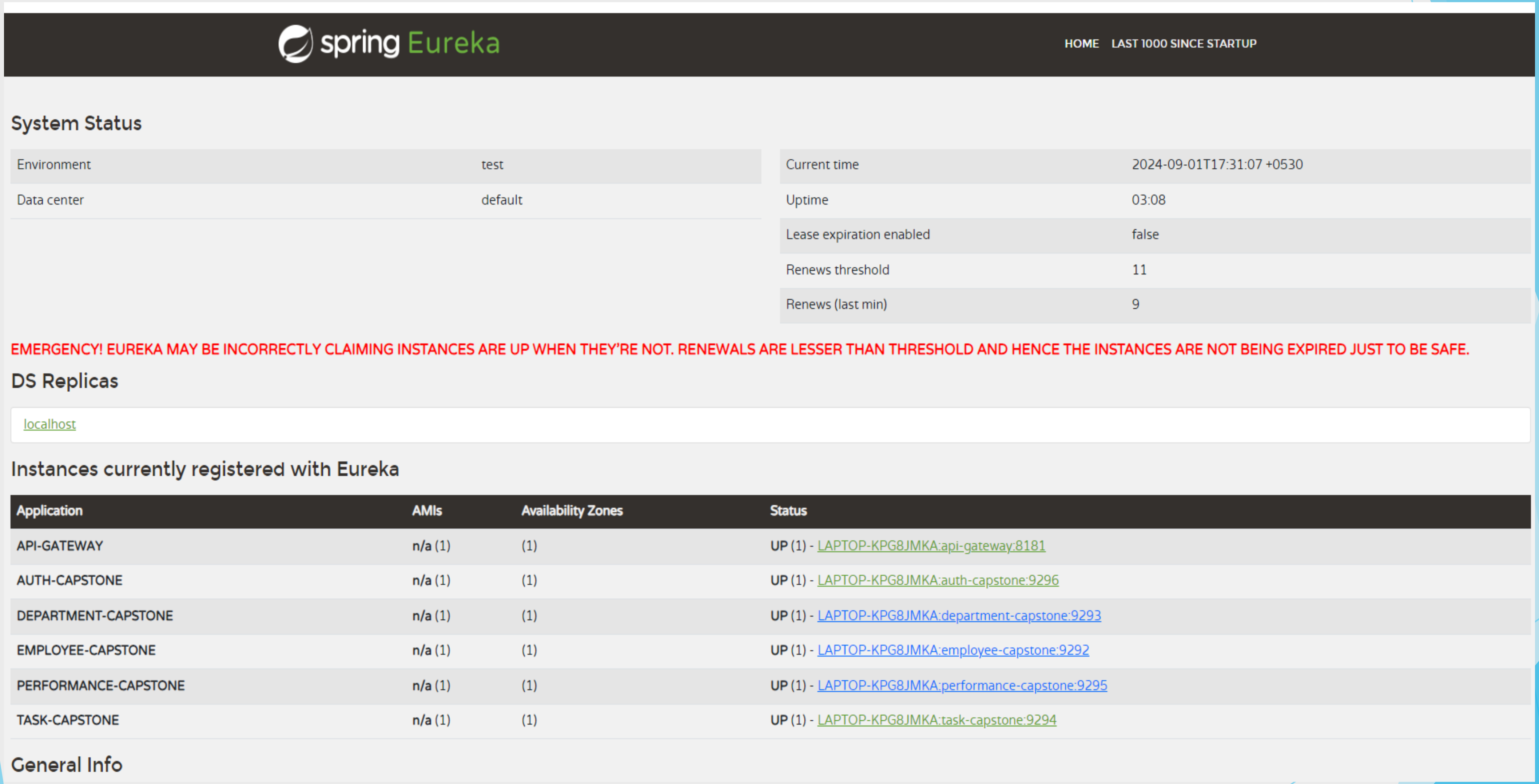
Employees can view tasks assigned to them and update the status of these tasks as they progress.

Task Tracking: Employees can track their progress and manage their workload effectively.

Performance Tracking:

Employees can view metrics related to their performance and receive feedback.

Performance Reviews: Participate in performance reviews and track their career development and goals.



spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2024-09-01T17:31:07 +0530
Uptime	03:08
Lease expiration enabled	false
Renews threshold	11
Renews (last min)	9

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

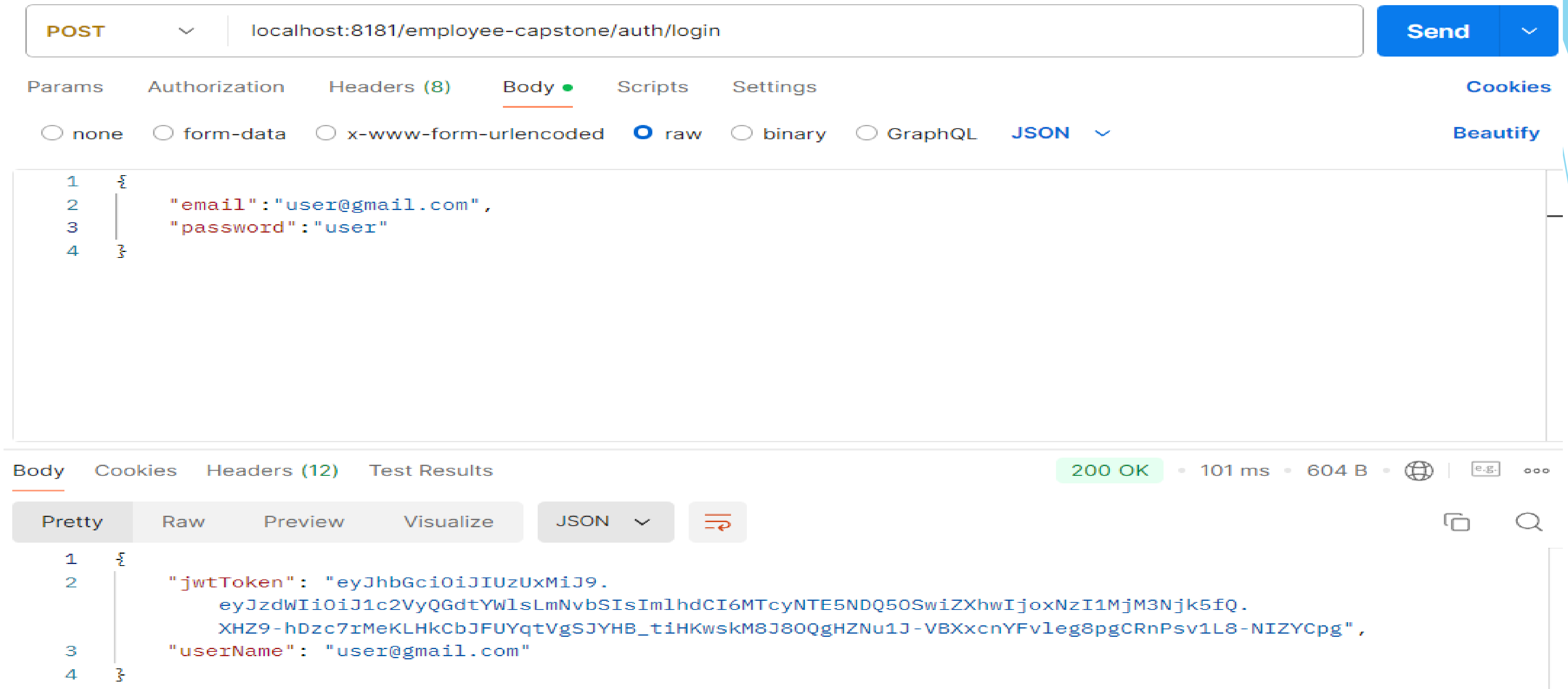
localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - LAPTOP-KPG8JMKA:api-gateway:8181
AUTH-CAPSTONE	n/a (1)	(1)	UP (1) - LAPTOP-KPG8JMKA:auth-capstone:9296
DEPARTMENT-CAPSTONE	n/a (1)	(1)	UP (1) - LAPTOP-KPG8JMKA:department-capstone:9293
EMPLOYEE-CAPSTONE	n/a (1)	(1)	UP (1) - LAPTOP-KPG8JMKA:employee-capstone:9292
PERFORMANCE-CAPSTONE	n/a (1)	(1)	UP (1) - LAPTOP-KPG8JMKA:performance-capstone:9295
TASK-CAPSTONE	n/a (1)	(1)	UP (1) - LAPTOP-KPG8JMKA:task-capstone:9294

General Info

Fig: Spring Eureka



localhost:8181/employee-capstone/auth/login

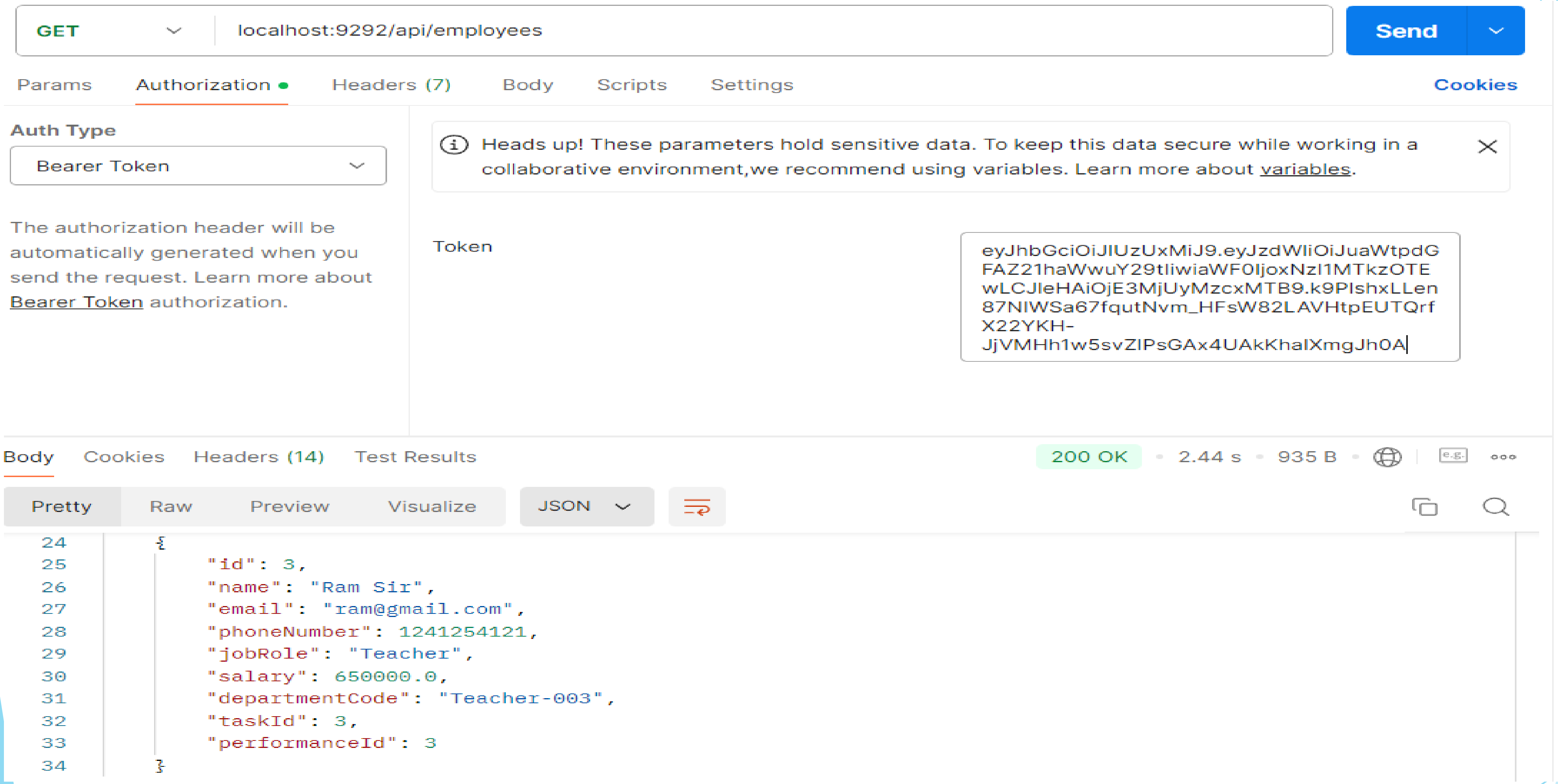
Body: raw [✓ raw] JSON

```
{
    "email":"user@gmail.com",
    "password":"user"
}
```

200 OK • 101 ms • 604 B

```
{
    "jwtToken": "eyJhbGciOiJIUzUxMiJ9.
        eyJzdWIiOiJ1c2VyQGdtYWlsLmNvbSIsImlhdCI6MTcyNTE5NDQ5OSwiZXhwIjoxNzI1MjM3Njk5fQ.
        XHZ9-hDzc7rMeKLHkCbJFUYqtVgSJYHB_tiHKwskM8J8OQgHZNu1J-VBXxcnYFvleg8pgCRnPsv1L8-NIZYCpg",
    "userName": "user@gmail.com"
}
```

Fig. Authentication



```
GET localhost:9292/api/employees
```

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment,we recommend using variables. Learn more about variables.

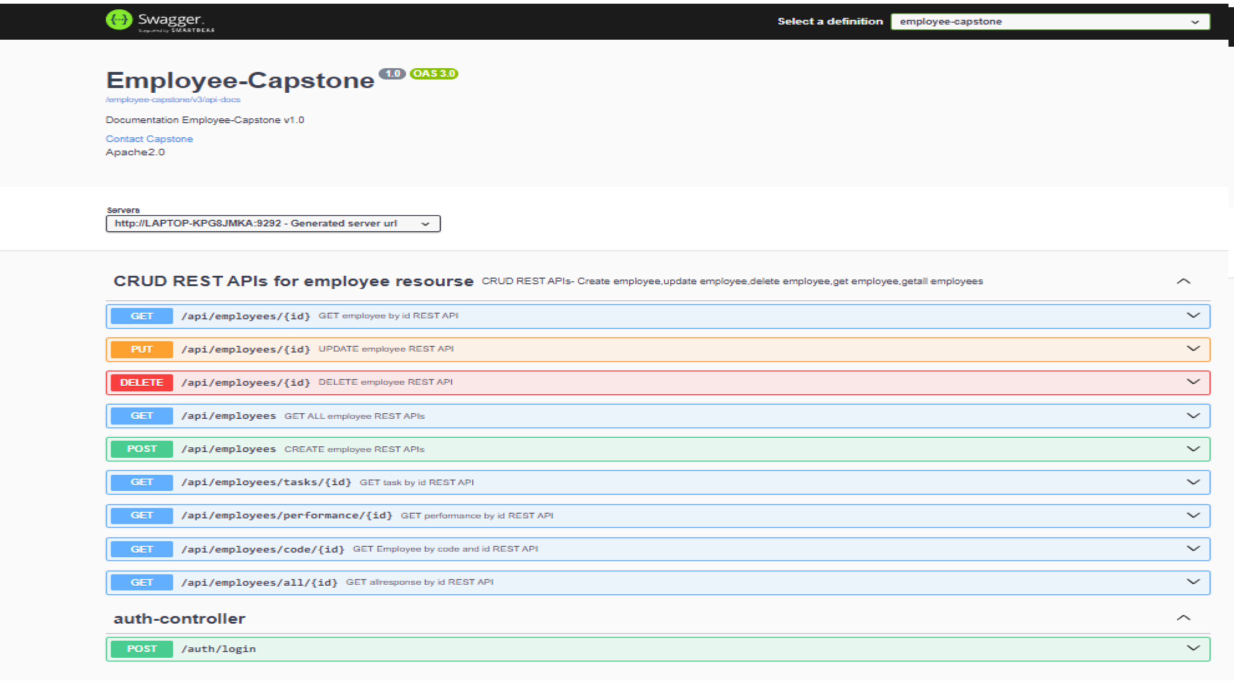
Token

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJuaWtpdGFAZ21haWwuY29tIiwiaWF0IjoxNzI1MTkzOTEwLCJleHAiOjE3MjUyMzcxMTB9.k9PIshxLLen87NIWSa67fqutNvm_HFsW82LAVHtpEUTQrfX22YKH-JjVMHh1w5svZIPsGAx4UAkKhaIXmgJh0A
```

200 OK • 2.44 s • 935 B

```
    {
        "id": 3,
        "name": "Ram Sir",
        "email": "ram@gmail.com",
        "phoneNumber": 1241254121,
        "jobRole": "Teacher",
        "salary": 650000.0,
        "departmentCode": "Teacher-003",
        "taskId": 3,
        "performanceId": 3
    }
```

Fig. With token we are fetching employee data


Swagger

Select a definition: employee-capstone

Employee-Capstone 1.0 OAS 3.0

/employee-capstone/v3/api-docs

Documentation Employee-Capstone v1.0

Contact Capstone

Apache2.0

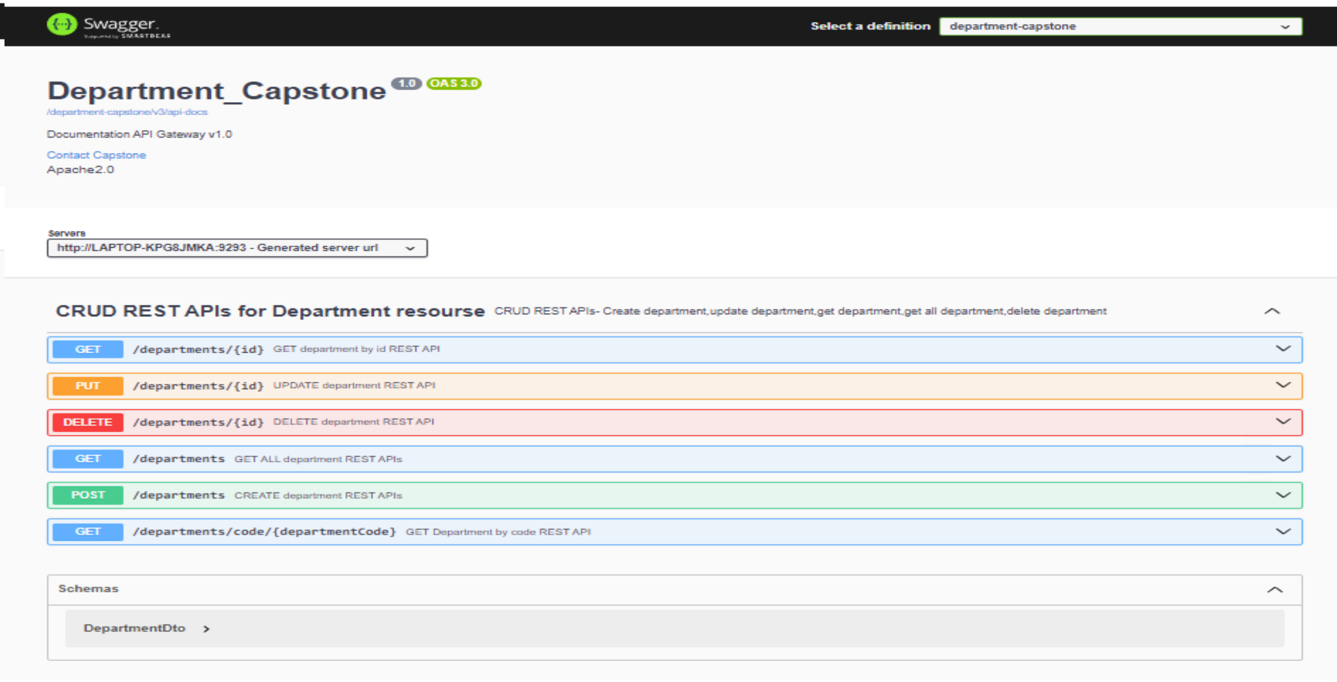
Servers: http://LAPTOP-KPG8JMKA:9292 - Generated server url

CRUD REST APIs for employee resource — CRUD REST APIs- Create employee,update employee,delete employee,get employee,getall employees

- GET `/api/employees/{id}` GET employee by id REST API
- PUT `/api/employees/{id}` UPDATE employee REST API
- DELETE `/api/employees/{id}` DELETE employee REST API
- GET `/api/employees` GET ALL employee REST APIs
- POST `/api/employees` CREATE employee REST APIs
- GET `/api/employees/tasks/{id}` GET task by id REST API
- GET `/api/employees/performance/{id}` GET performance by id REST API
- GET `/api/employees/code/{id}` GET Employee by code and id REST API
- GET `/api/employees/all/{id}` GET allresponse by id REST API

auth-controller

- POST `/auth/login`


Swagger

Select a definition: department-capstone

Department_Capstone 1.0 OAS 3.0

/department-capstone/v3/api-docs

Documentation API Gateway v1.0

Contact Capstone

Apache2.0

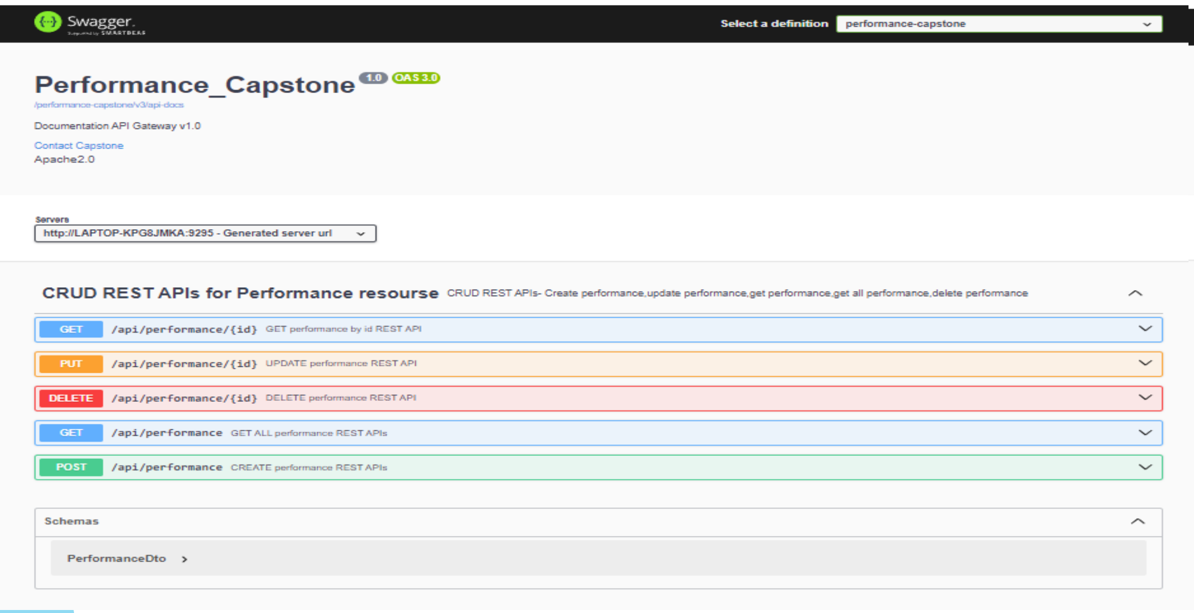
Servers: http://LAPTOP-KPG8JMKA:9293 - Generated server url

CRUD REST APIs for Department resource — CRUD REST APIs- Create department,update department,get department,get all department,delete department

- GET `/departments/{id}` GET department by id REST API
- PUT `/departments/{id}` UPDATE department REST API
- DELETE `/departments/{id}` DELETE department REST API
- GET `/departments` GET ALL department REST APIs
- POST `/departments` CREATE department REST APIs
- GET `/departments/code/{departmentCode}` GET Department by code REST API

Schemas

- DepartmentDto


Swagger

Select a definition: performance-capstone

Performance_Capstone 1.0 OAS 3.0

/performance-capstone/v3/api-docs

Documentation API Gateway v1.0

Contact Capstone

Apache2.0

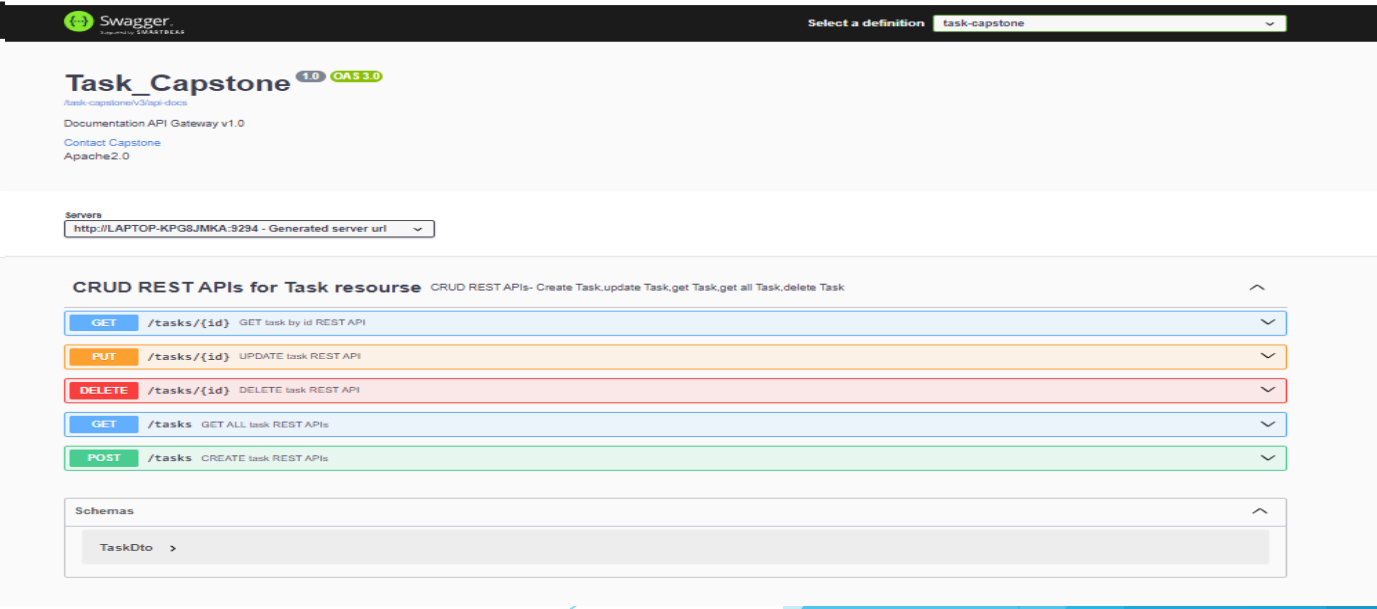
Servers: http://LAPTOP-KPG8JMKA:9295 - Generated server url

CRUD REST APIs for Performance resource — CRUD REST APIs- Create performance,update performance,get performance,get all performance,delete performance

- GET `/api/performance/{id}` GET performance by id REST API
- PUT `/api/performance/{id}` UPDATE performance REST API
- DELETE `/api/performance/{id}` DELETE performance REST API
- GET `/api/performance` GET ALL performance REST APIs
- POST `/api/performance` CREATE performance REST APIs

Schemas

- PerformanceDto


Swagger

Select a definition: task-capstone

Task_Capstone 1.0 OAS 3.0

/task-capstone/v3/api-docs

Documentation API Gateway v1.0

Contact Capstone

Apache2.0

Servers: http://LAPTOP-KPG8JMKA:9294 - Generated server url

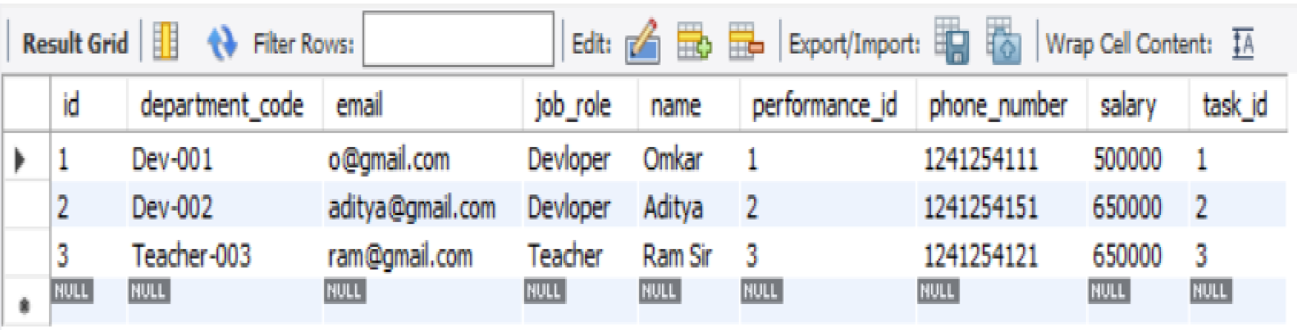
CRUD REST APIs for Task resource — CRUD REST APIs- Create Task,update Task,get Task,get all Task,delete Task

- GET `/tasks/{id}` GET task by id REST API
- PUT `/tasks/{id}` UPDATE task REST API
- DELETE `/tasks/{id}` DELETE task REST API
- GET `/tasks` GET ALL task REST APIs
- POST `/tasks` CREATE task REST APIs

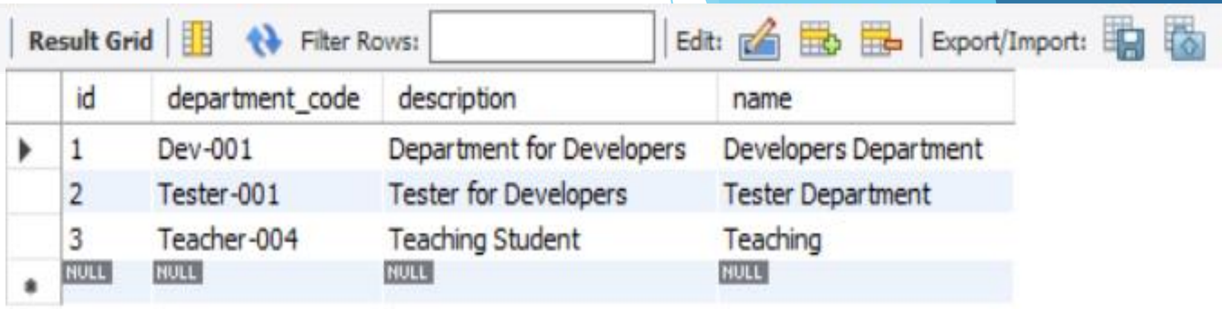
Schemas

- TaskDto

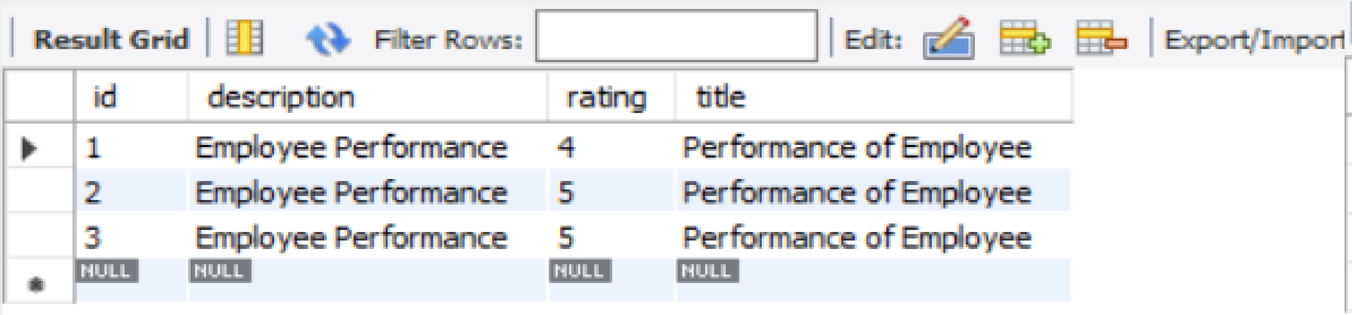
Fig:Swagger



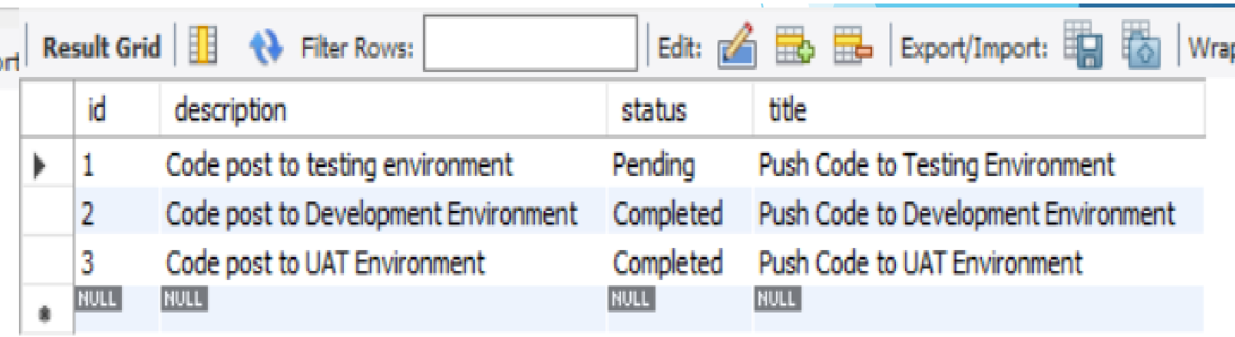
id	department_code	email	job_role	name	performance_id	phone_number	salary	task_id
1	Dev-001	o@gmail.com	Devloper	Omkar	1	1241254111	500000	1
2	Dev-002	aditya@gmail.com	Devloper	Aditya	2	1241254151	650000	2
3	Teacher-003	ram@gmail.com	Teacher	Ram Sir	3	1241254121	650000	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



id	department_code	description	name
1	Dev-001	Department for Developers	Developers Department
2	Tester-001	Tester for Developers	Tester Department
3	Teacher-004	Teaching Student	Teaching
NULL	NULL	NULL	NULL



id	description	rating	title
1	Employee Performance	4	Performance of Employee
2	Employee Performance	5	Performance of Employee
3	Employee Performance	5	Performance of Employee
NULL	NULL	NULL	NULL



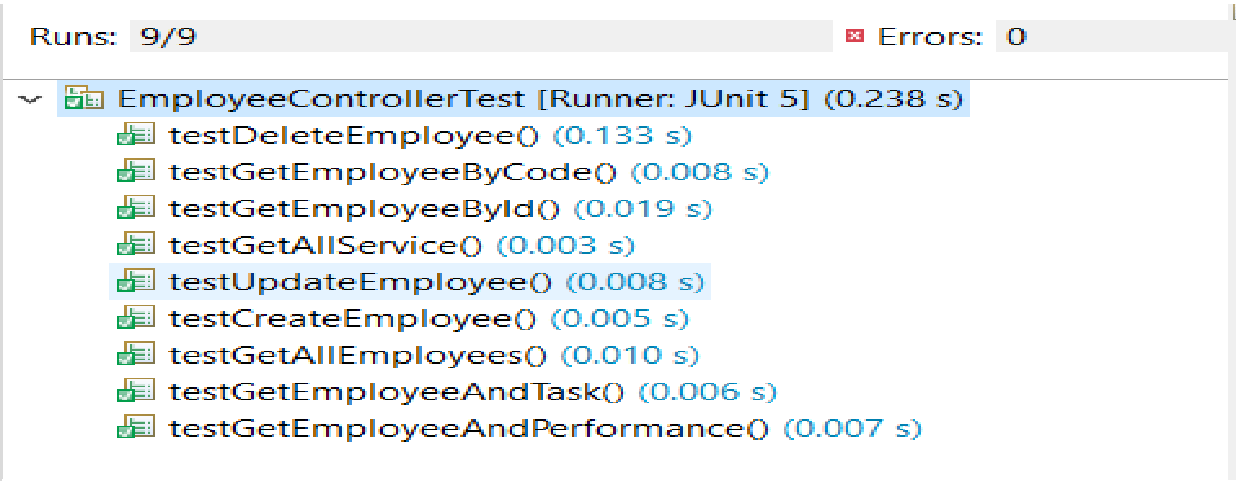
id	description	status	title
1	Code post to testing environment	Pending	Push Code to Testing Environment
2	Code post to Development Environment	Completed	Push Code to Development Environment
3	Code post to UAT Environment	Completed	Push Code to UAT Environment
NULL	NULL	NULL	NULL

Fig: MySQL Workbench

Testing and Refinement

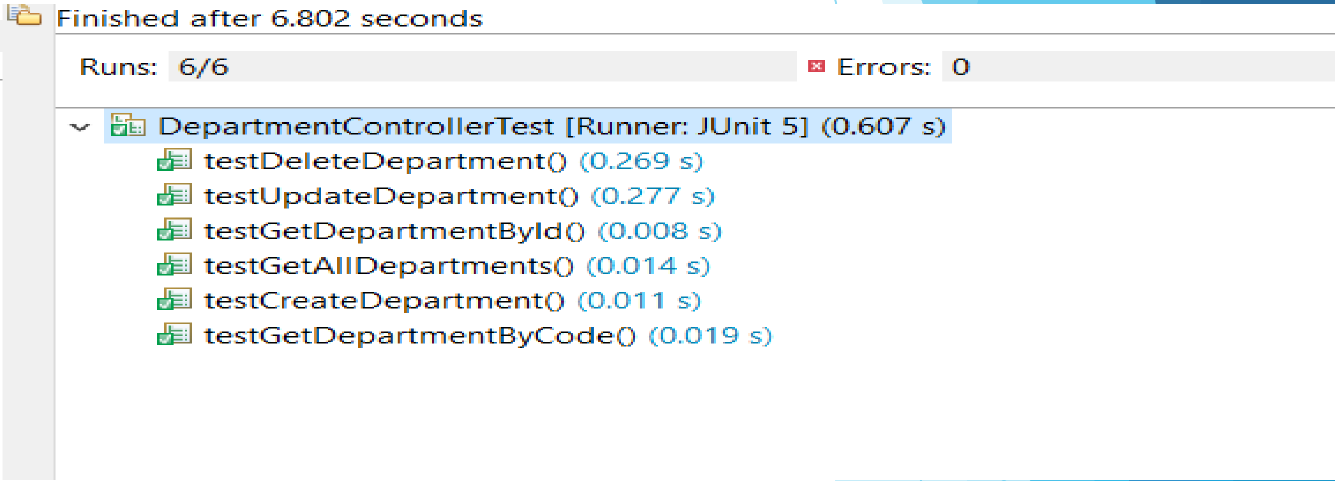
Testing Approaches:

Unit Testing: Each microservice tested independently using JUnit and Mockito.



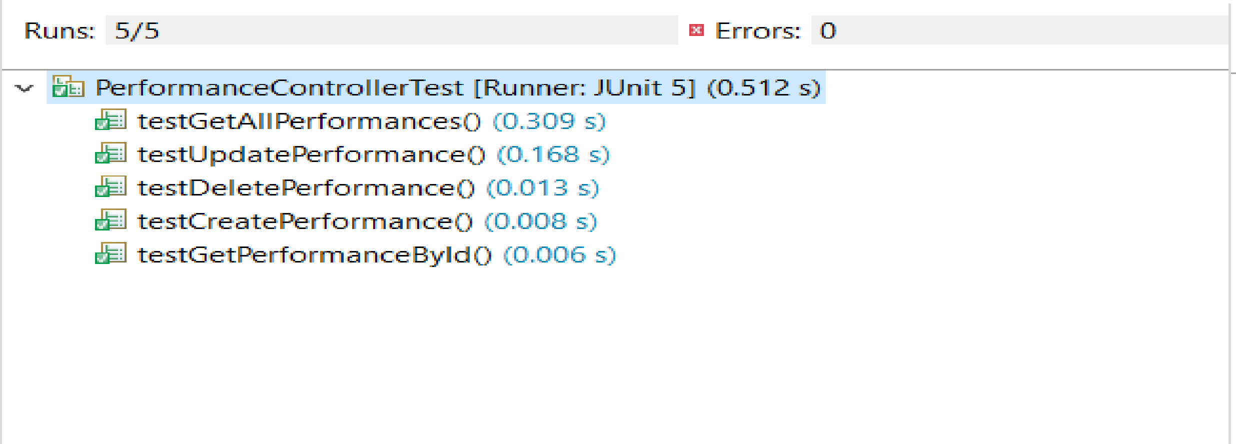
```
Runs:  9/9                    Errors:  0

EmployeeControllerTest [Runner: JUnit 5] (0.238 s)
    testDeleteEmployee() (0.133 s)
    testGetEmployeeByCode() (0.008 s)
    testGetEmployeeById() (0.019 s)
    testGetAllService() (0.003 s)
    testUpdateEmployee() (0.008 s)
    testCreateEmployee() (0.005 s)
    testGetAllEmployees() (0.010 s)
    testGetEmployeeAndTask() (0.006 s)
    testGetEmployeeAndPerformance() (0.007 s)
```



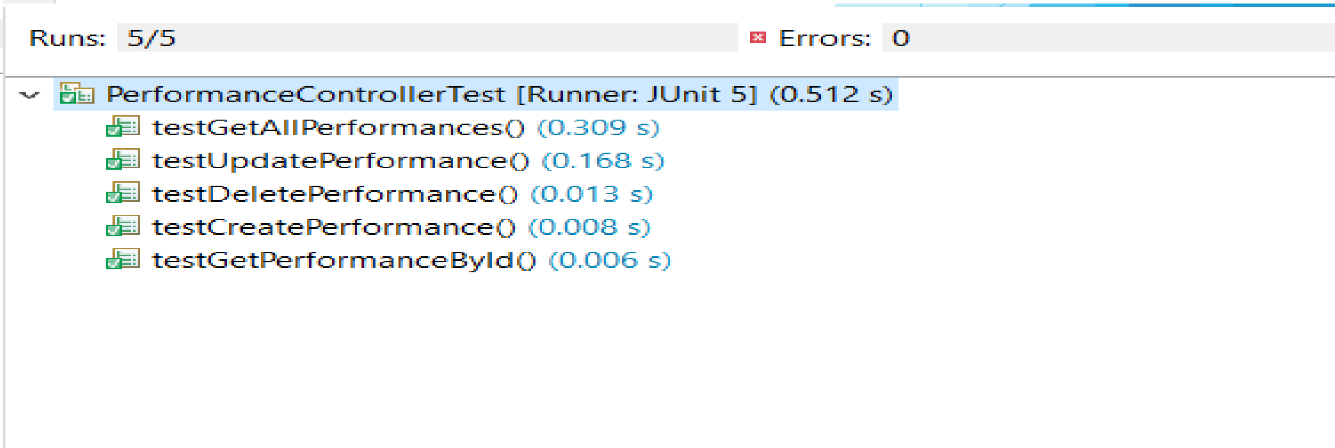
```
Finished after 6.802 seconds
Runs:  6/6                    Errors:  0

DepartmentControllerTest [Runner: JUnit 5] (0.607 s)
    testDeleteDepartment() (0.269 s)
    testUpdateDepartment() (0.277 s)
    testGetDepartmentById() (0.008 s)
    testGetAllDepartments() (0.014 s)
    testCreateDepartment() (0.011 s)
    testGetDepartmentByCode() (0.019 s)
```



```
Runs:  5/5                    Errors:  0

PerformanceControllerTest [Runner: JUnit 5] (0.512 s)
    testGetAllPerformances() (0.309 s)
    testUpdatePerformance() (0.168 s)
    testDeletePerformance() (0.013 s)
    testCreatePerformance() (0.008 s)
    testGetPerformanceById() (0.006 s)
```



```
Runs:  5/5                    Errors:  0

PerformanceControllerTest [Runner: JUnit 5] (0.512 s)
    testGetAllPerformances() (0.309 s)
    testUpdatePerformance() (0.168 s)
    testDeletePerformance() (0.013 s)
    testCreatePerformance() (0.008 s)
    testGetPerformanceById() (0.006 s)
```

Conclusion

Summary:

The Employee Management System project effectively showcases the advantages of using a microservices architecture for scalability and modularity. By utilizing Java, Spring Boot, Spring Cloud, and MySQL/MariaDB, the system efficiently manages employees, departments, tasks, and performance evaluations. It provides secure access via JWT-based authentication, ensuring data integrity and confidentiality. This system lays a solid foundation for further enhancements, including advanced analytics, mobile integration, and machine learning, making it a robust solution for modern human resource management.

Future Enhancements:

- **Advanced Analytics:** Integrate advanced analytics for deeper insights into employee performance and organizational efficiency.
- **Mobile Access:** Develop mobile applications for enhanced accessibility and a better user experience.
- **Machine Learning:** Implement predictive analytics to forecast employee performance trends.
- **HR System Integration:** Connect with payroll, attendance, and other HR systems for comprehensive management.
- **Real-Time Notifications:** Add real-time notifications for task updates and performance feedback.

THANK YOU