CAPSTONE PROJECT

EMPLOYEE MANAGEMENT SYSTEM

BATCH-8

JAVA J2EE

Name: Omkar Shadangule

Email: oshadangule@gmail.com

Date: September 2, 2024

Trainer: Ramakrishna (RK)

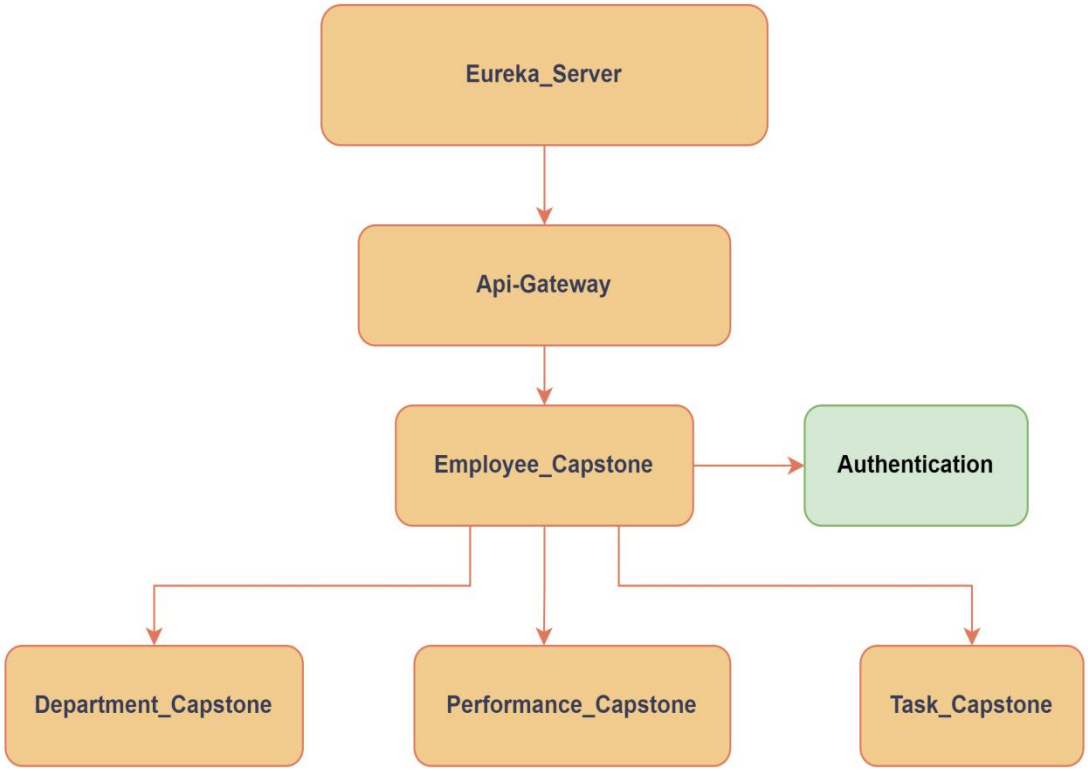
Contents:

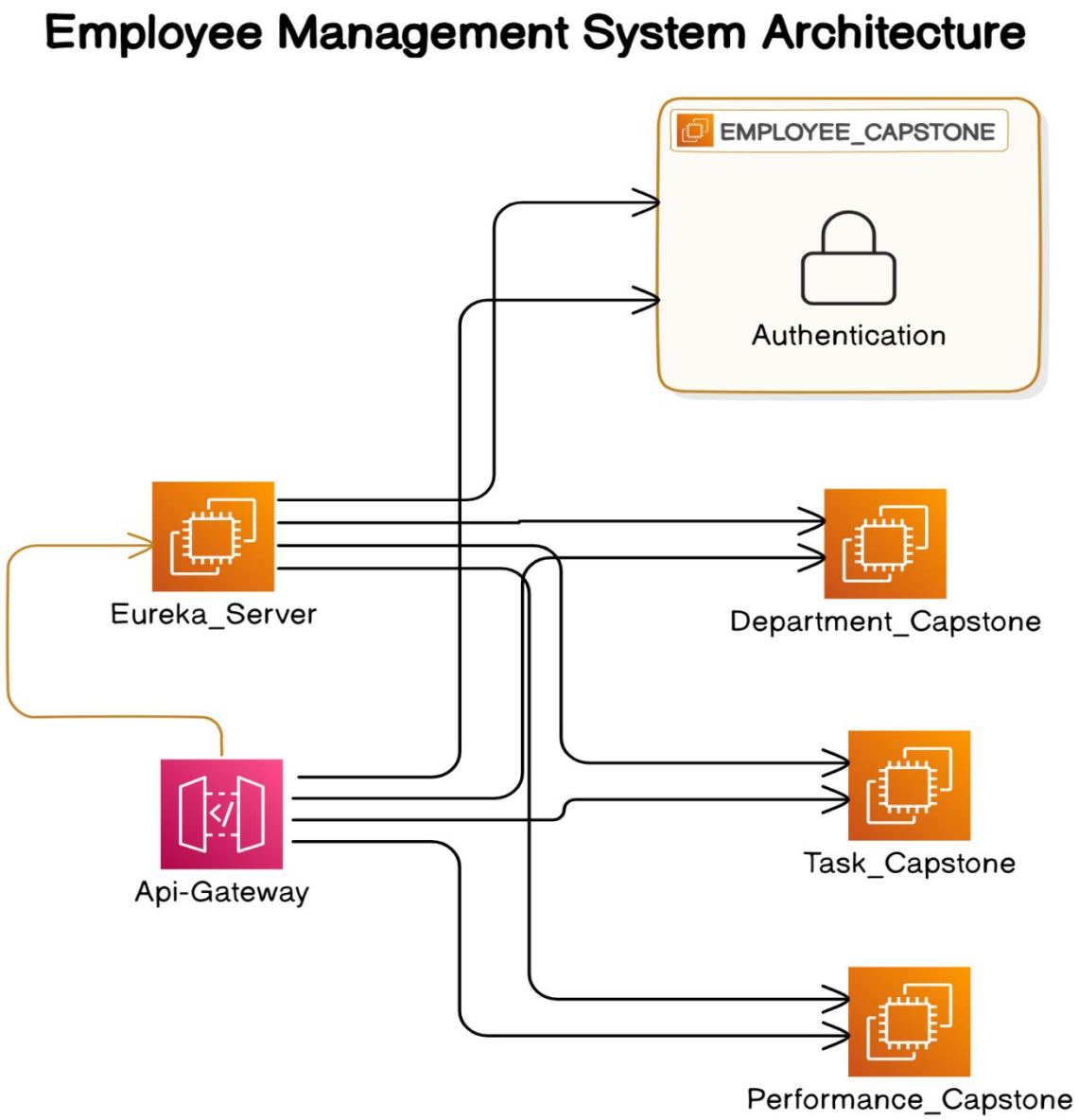
Content Table

1. Introduction

1.1 Project Overview

The Employee Management System is a distributed application designed to manage employee-related information within an organization efficiently. The system allows administrators to manage employees, departments, tasks, and performance evaluations. The architecture of the system is based on microservices, which ensures scalability and flexibility, enabling the organization to adapt and grow as needed.





2. Technologies Used

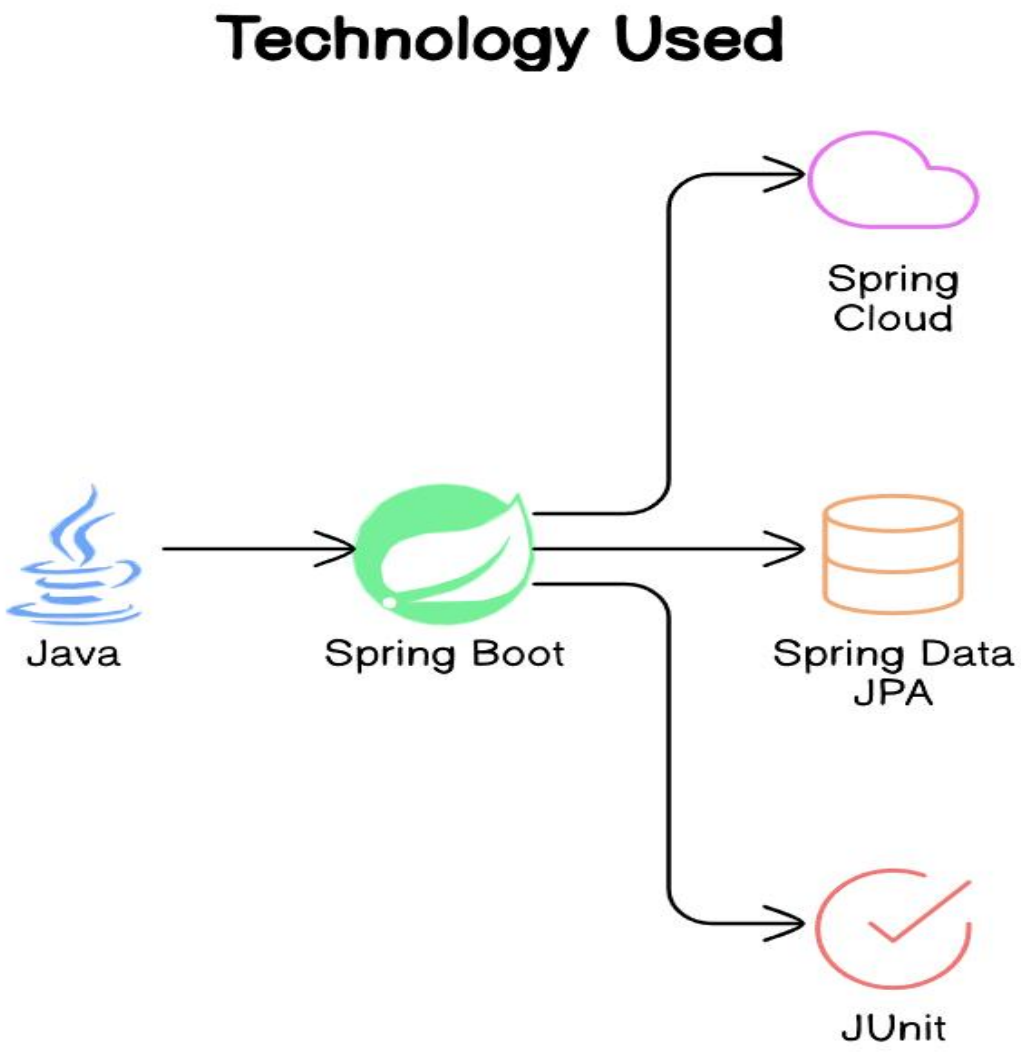
2.1 Java: Core programming language used for application development.

2.2 Spring Boot: Framework that simplifies the creation of production-ready Spring applications, allowing for easy setup and rapid deployment.

2.3 Spring Cloud: Handles cross-cutting concerns like configuration management, service discovery, circuit breakers, and distributed tracing, enabling microservices architecture.

2.4 Spring Data JPA: Simplifies data access and provides a standard API for database interactions.

2.5 JUnit: A testing framework used for unit testing the application's components, ensuring code quality and reliability.



3. Problem Statement:

The primary goal of this project is to create a robust Employee Management System that caters to the following requirements.

3.1 For Admins:

- A centralized system to manage employees, departments, and roles.
- CRUD (Create, Read, Update, Delete) operations on employee records.
- Assignment of roles to employees and organizing them into departments.
- Management of tasks and performance evaluations.

3.2 For Employees:

- Ability to manage personal profiles.
- View assigned tasks and update task statuses.
- Track performance and receive feedback from managers.

4. Project Flow

Admin Dashboard

4.1.1 Role: Acts as the centralized interface for Admins to manage the organization.

4.1.2 Features: Provides analytics, reports on employee performance, and departmental efficiency.

4.2 Employee Management

4.2.1 Role: Enables Admins to perform CRUD operations on employee records.

4.2.2 Features: Role assignment, department organization, and employee record management.

4.3 Department Management

4.3.1 Role: Allows Admins to manage departments by creating, viewing, editing, or deleting department records.

4.3.2 Features: Organizes employees within departments and manages departmental data.

4.4 Task Management

4.4.1 Role: Enables Admins to create and assign tasks to employees, monitor progress, and update task statuses.

4.4.2 Features: Task tracking and monitoring of completion rates and deadlines.

4.5 Performance Management

4.5.1 Role: Allows Admins to track and evaluate employee performance.

4.5.2 Features: Performance reviews, feedback provision, and generation of evaluation reports.

4.6. Employee Registration & Authentication

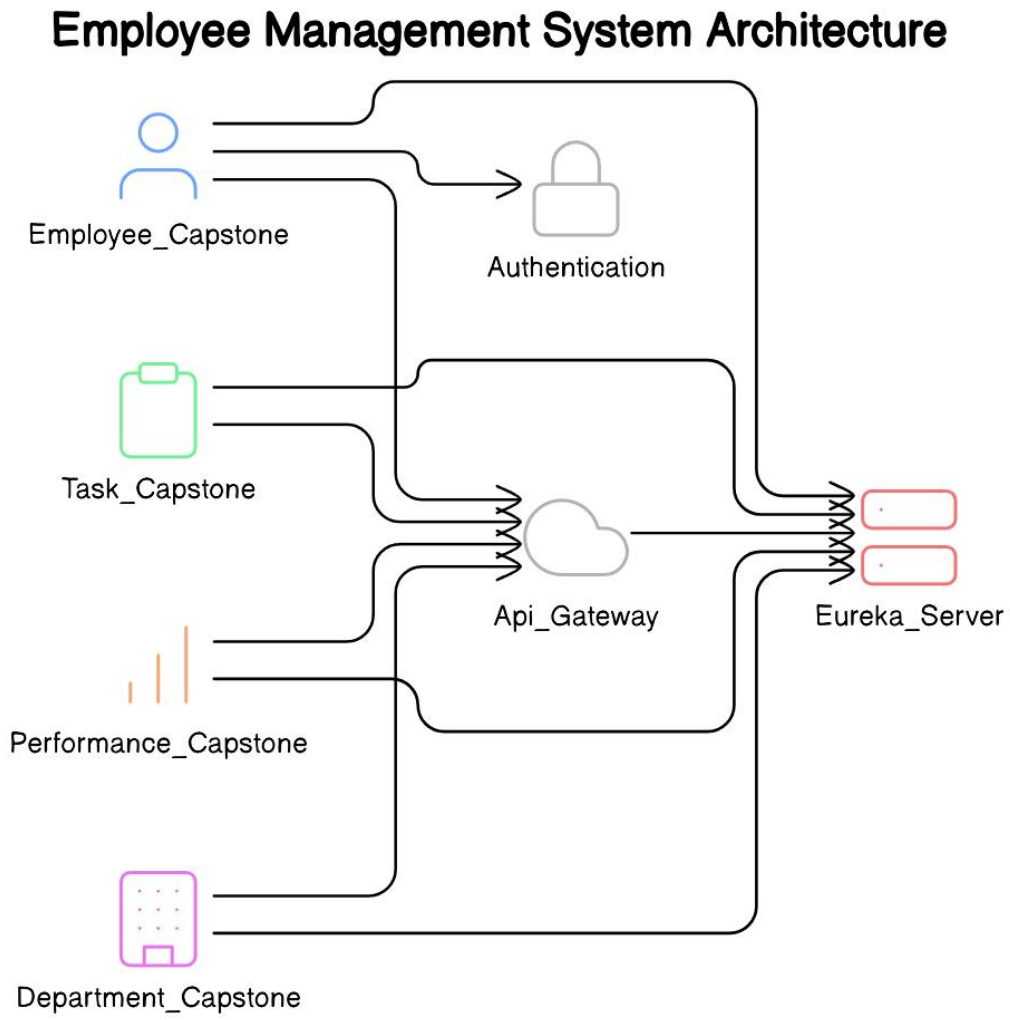
4.1.1 Role: Manages employee registration and login processes, ensuring secure access to the system.

4.1.2 Features: Profile management, access to assigned roles and departments.

5. Microservices Architecture

5.1 Overview

Microservices architecture breaks down the application into small, independent services, each responsible for specific business functionalities. These services communicate through APIs, allowing for greater flexibility, scalability, and maintainability.



5.2 Service Registry & Discovery

Eureka Server

5.2.1 Role: Eureka is a service registry used to keep track of all available microservices and their instances within the system. Each service registers itself with Eureka upon startup and periodically sends heartbeats to confirm its availability.

5.2.2 Functionality: Eureka provides a central directory where services can look up the locations (IP addresses and ports) of other services. This enables dynamic scaling and ensures that services can discover and communicate with one another even if they are deployed on different servers or cloud instances.

5.2.3 Failover: Eureka can work in a high-availability mode where multiple Eureka servers are deployed, ensuring that the service registry remains available even in case of a server failure.

5.3 API Gateway

Spring Cloud Gateway

5.3.1 Role: The API Gateway acts as the single entry point for client requests, abstracting the complexities of the microservices architecture from the client. It routes requests to the appropriate backend services based on the configured routes.

5.3.2 Functionality:

Routing: Directs client requests to the corresponding microservices based on URL patterns or other request properties.

Security: Integrates with authentication services to enforce security policies, including authentication and authorization.

Load Balancing: Distributes incoming requests across multiple instances of a microservice, improving performance and reliability.

Rate Limiting: Controls the rate at which requests are processed, protecting the system from overload by limiting the number of requests per client within a specified time frame.

Request Validation and Transformation: Validates incoming requests and can transform them before forwarding them to the backend services.

5.4 Authentication Service

5.4.1 Role: Manages the authentication and authorization processes across the application, ensuring that only authorized users can access certain features and data.

5.4.2 Functionality:

User Registration and Login: Handles user registration and login, securely storing user credentials.

JWT (JSON Web Token) Generation: Issues JWTs upon successful authentication, which are then used to secure communications between the client and server.

Role-Based Access Control (RBAC): Enforces role-based access policies, ensuring that Admins and Employees have access only to the functionalities they are permitted to use.

OAuth2 Support: Optionally integrates with OAuth2 for external authentication providers, such as Google or GitHub, providing flexibility in authentication methods.

5.5 Employee Management Microservice

Employee Directory:

5.5.1 Role: This service is responsible for all CRUD operations related to employee data, including personal information, job titles, department affiliations, and roles within the organization.

5.5.2 Functionality:

Employee Records Management: Supports the creation, update, retrieval, and deletion of employee records, ensuring that the organization's employee data is always up-to-date.

Role Management: Allows Admins to assign and update roles for each employee, dictating their permissions and access levels within the system.

Search and Filter: Provides APIs for searching and filtering employee records based on various criteria, such as department, job title, or performance metrics.

Integration with Other Services: Seamlessly integrates with the Task and Performance Management microservices, ensuring that changes to employee roles or departments are reflected across the system.

Variables	Data Type
Id	Long
Name	String
Email	String
PhoneNumber	Long
JobRole	String
Salary	Double
DepartmentCode	String
PerformanceId	Long
TaskId	Long

5.6 Department Management Microservice

Department Catalog:

5.6.1 Role: Manages department-related data, allowing for the organization and categorization of employees within specific departments.

5.6.2 Functionality:

Department CRUD Operations: Provides APIs to create, update, delete, and retrieve department records, ensuring that the organizational structure is accurately represented.

Employee Assignment: Manages the assignment of employees to departments, facilitating clear organizational hierarchies and reporting structures.

Departmental Analytics: Offers analytics and reports on departmental performance, including headcount, productivity, and inter-departmental collaboration metrics.

Variables	Data Type
Id	Long
Name	String
Description	String
DepartmentCode	Long

5.7 Task Management Microservice

Task Assignment:

5.7.1 Role: This microservice is central to the management of tasks within the organization, enabling efficient task distribution and tracking.

5.7.2 Functionality:

Task Creation and Assignment: Admins can create tasks and assign them to specific employees, setting deadlines and priorities to ensure timely completion.

Progress Tracking: Tracks the progress of tasks, allowing both Admins and Employees to monitor the status of each task in real-time.

Task Notifications: Sends notifications to employees when new tasks are assigned or when deadlines are approaching, helping to keep tasks on track.

Integration with Performance Management: Links task completion data to the Performance Management microservice, providing input for employee evaluations.

Variables	Data Type
Id	Long
Status	String
Description	String
Title	String

5.8 Performance Management Microservice

Employee Performance Tracking:

5.8.1 Role: Evaluates and tracks employee performance based on various metrics, including task completion, punctuality, and quality of work.

5.8.2 Functionality:

Performance Reviews: Allows Admins and managers to conduct regular performance reviews, providing structured feedback to employees.

Feedback Mechanism: Employees can receive and respond to feedback, facilitating continuous improvement and career development.

Evaluation Reports: Generates detailed reports based on employee performance data, which can be used for decisions on promotions, training, or disciplinary actions.

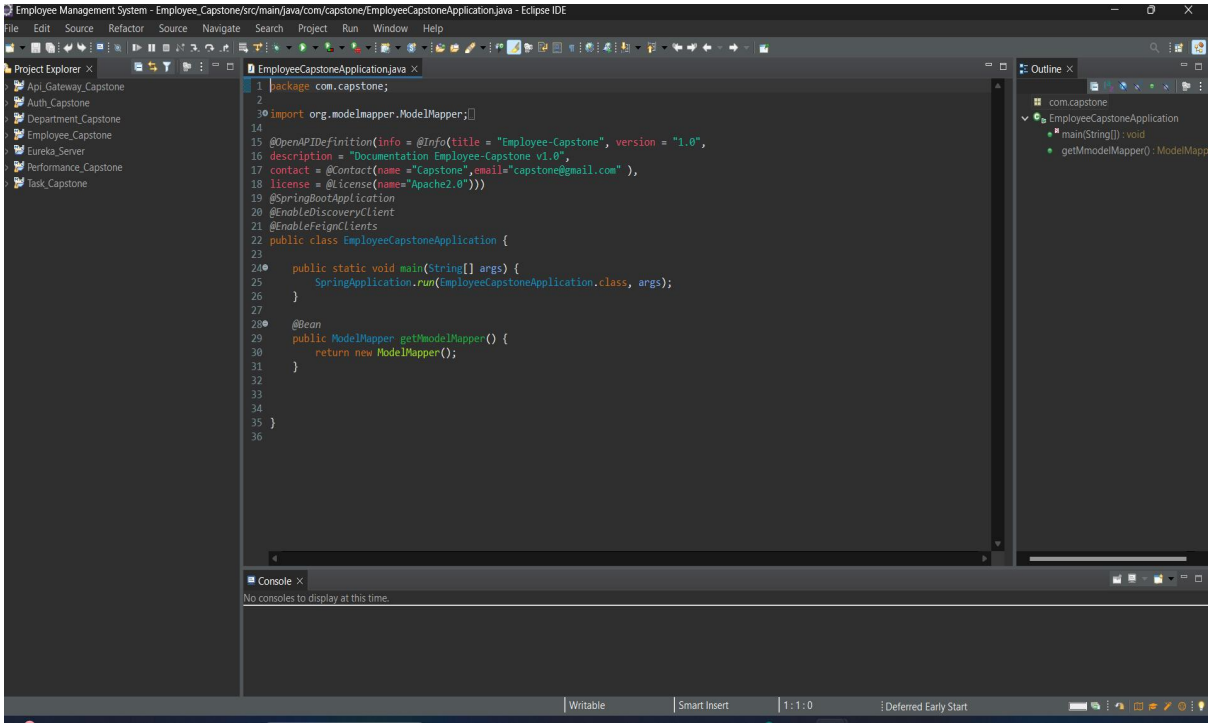
Variables	Data Type
Id	Long
Title	String
Description	String
Rating	Double

Goal Setting and Tracking: Admins can set performance goals for employees and track progress against these goals, aligning individual performance with organizational objectives.

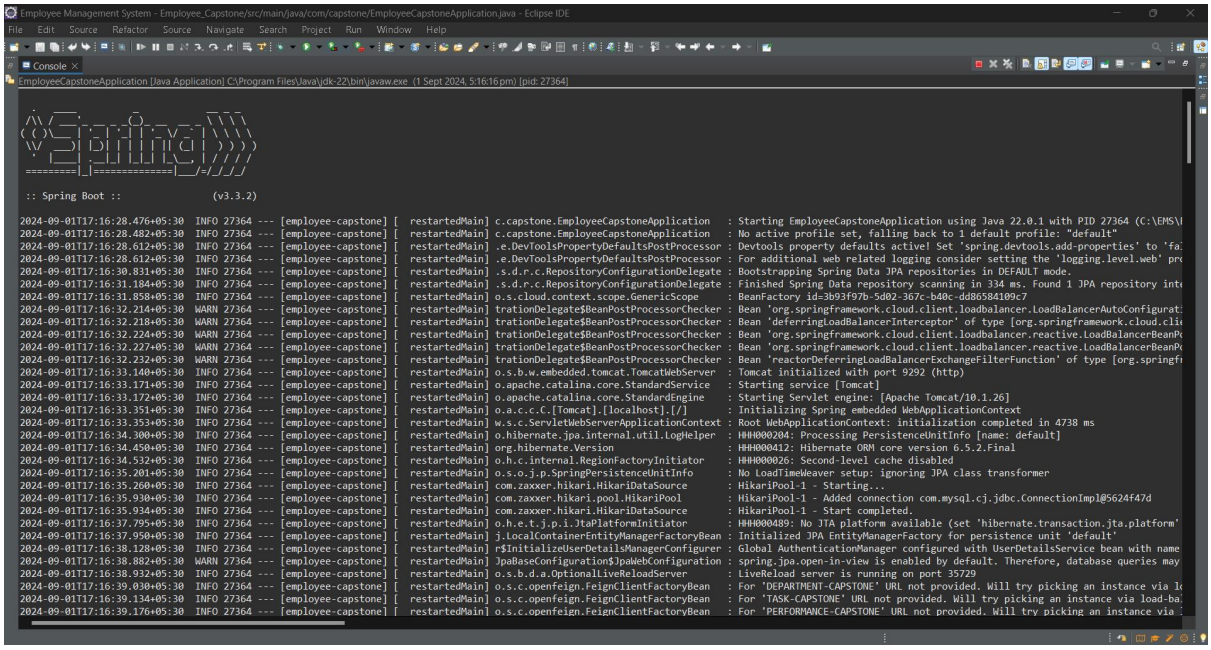
Note:We can perform CRUD operation using Postman and Swagger.

Class Diagram of microservices:

7. Eclipse Project Workspace-Backend:

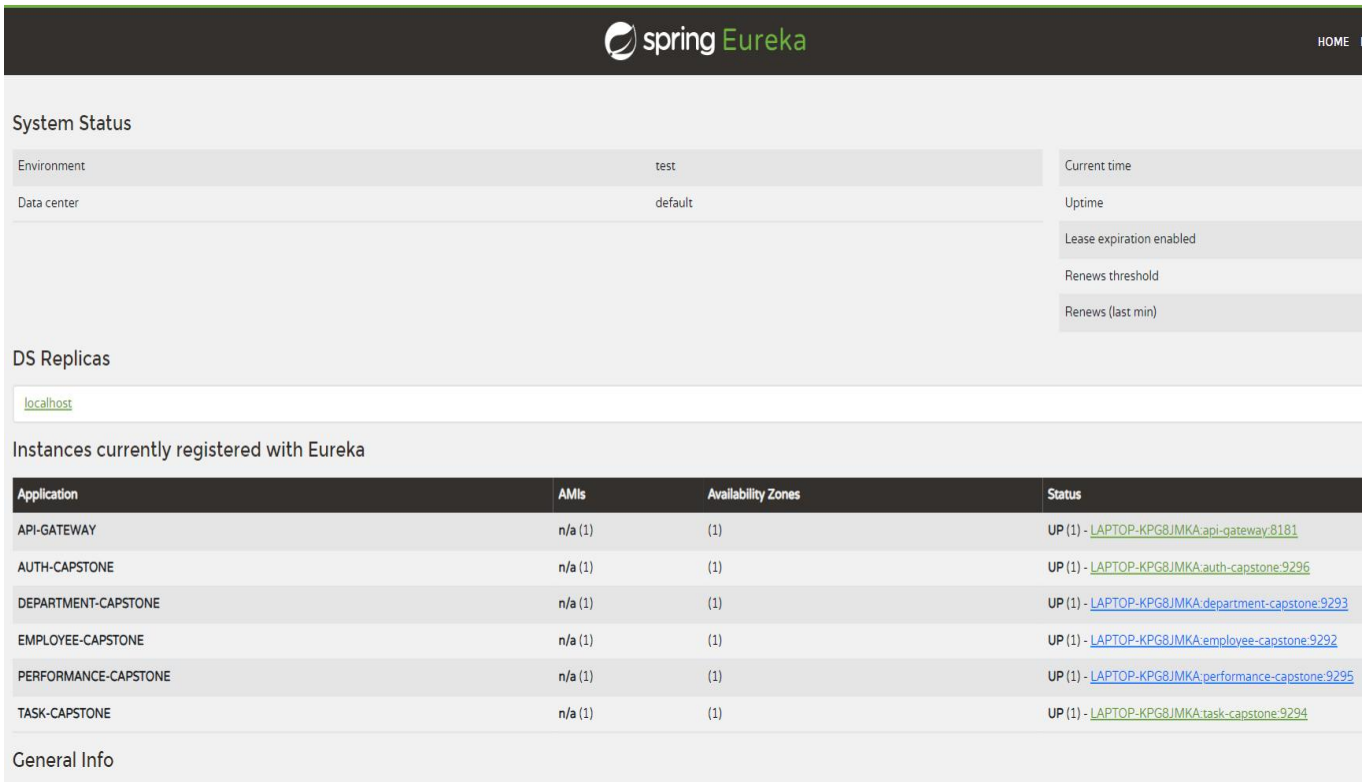


8. Running the Application as SpringbootApp:

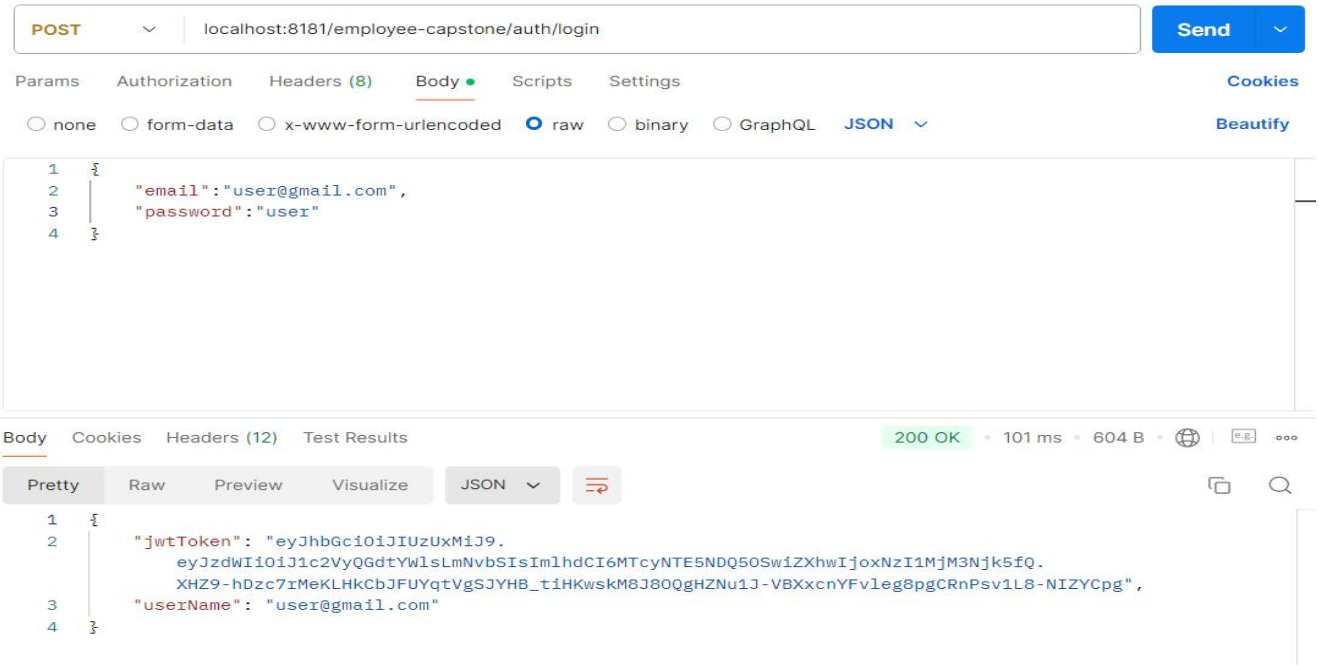


9. Eureka Server page:

All micorservices register on Eureka Server.



10. Authentication for Employee Microservice:



11. Fetching Employee data using jwtToken.

GET localhost:9292/api/employees Send

Params Authorization Headers (7) Body Scripts Settings Cookies

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.eyJzdWIiOiJ1c2VyQ...

Body Cookies Headers (14) Test Results 200 OK 444 ms 935 B

Pretty Raw Preview Visualize JSON


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


9. Swagger UI of Employee_Capstone:

Swagger /v3/api-docs Explore

Employee-Capstone 1.0 OAS 3.0

/v3/api-docs

Documentation Employee-Capstone v1.0

Contact Capstone

Apache2.0

Servers

http://localhost:9292 - Generated server url

CRUD REST APIs for employee resourse 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

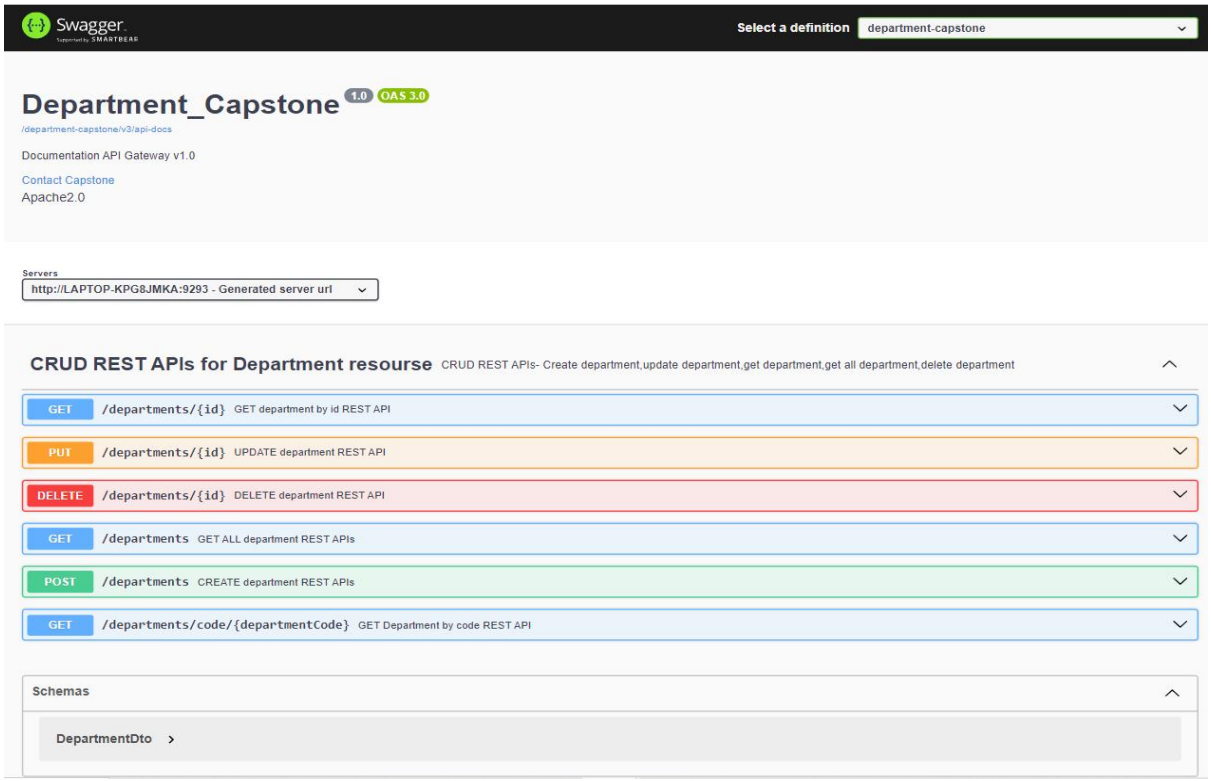
auth-controller

- POST /auth/login

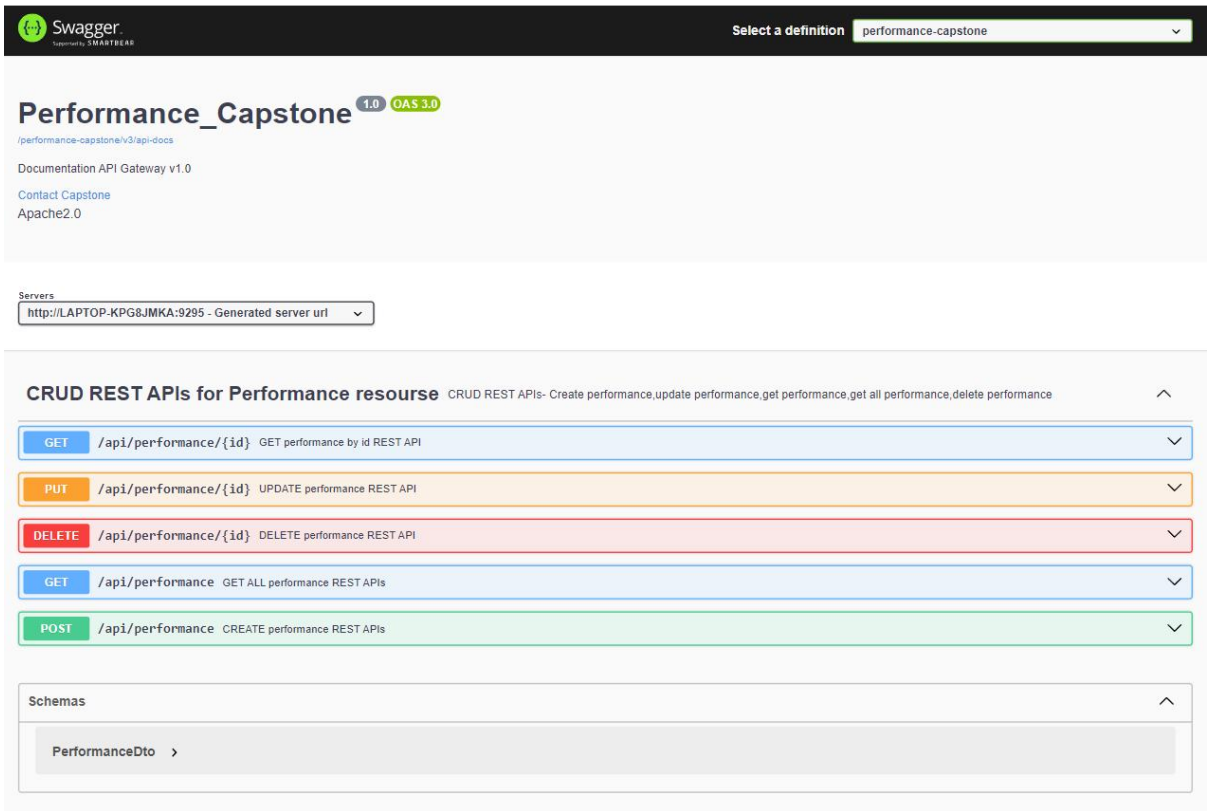
Schemas

EmployeeDto

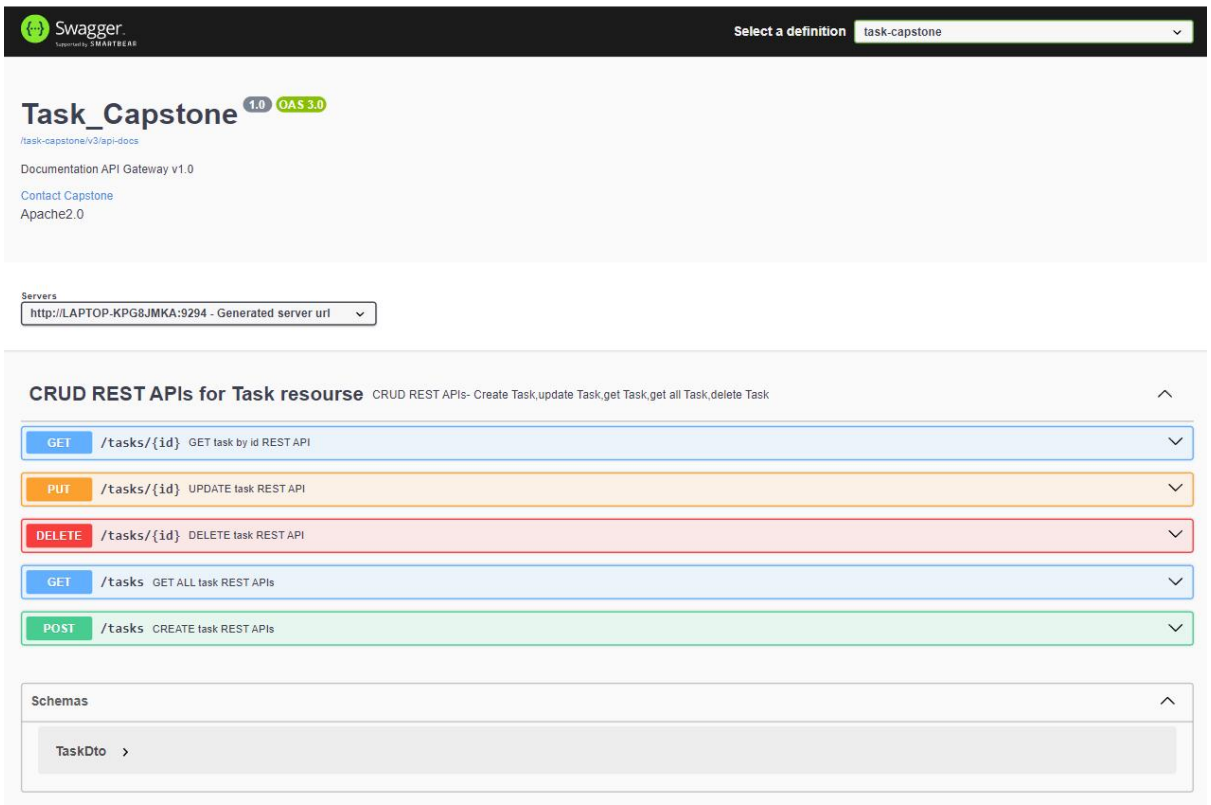
10. Swagger UI of Department_Capstone:



11. Swagger UI of Performance_Capstone:

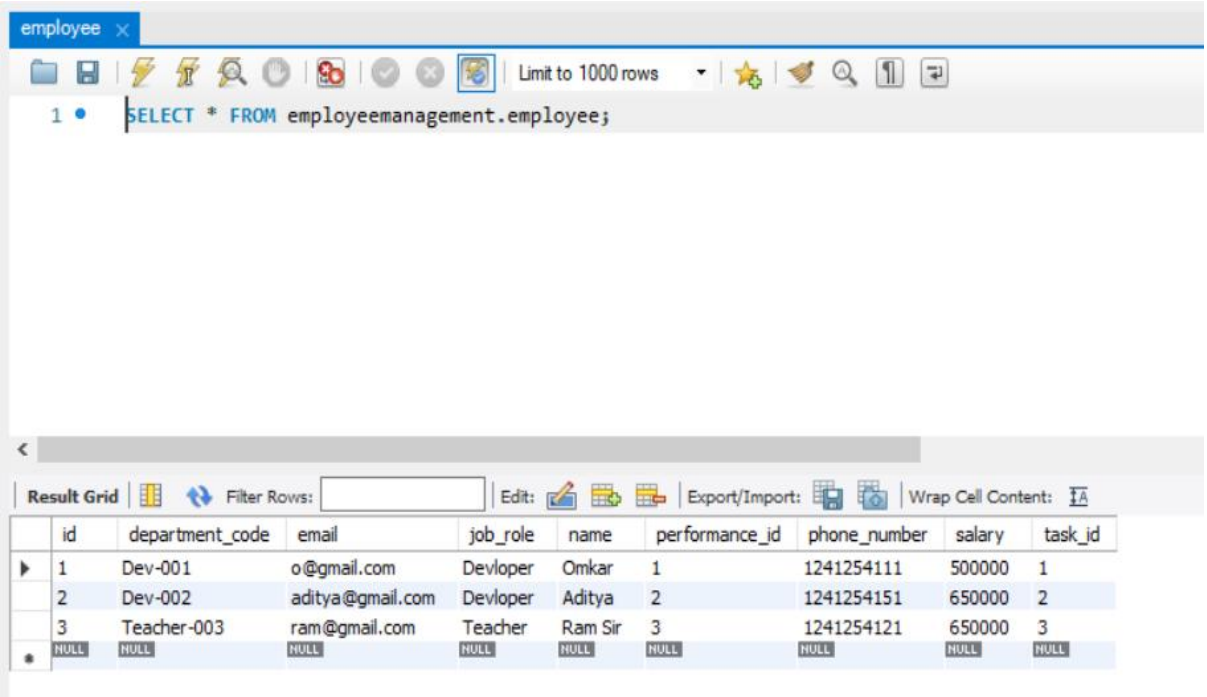


12. Swagger UI of Task_Capstone:



13. Data Base MySQL Workbench:

13.1 Employee Table:



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

13.2 Department Table:

department

```
1 • SELECT * FROM departmentmanagement.department;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

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

13.3 Performance Table:

performance

```
1 • SELECT * FROM performancemanagement.performance;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conten

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

13.4 Task Table:

task

Limit to 1000 rows

```
1 • SELECT * FROM taskmanagement.task;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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

14. Testing and Refinement

- To ensure the reliability and quality of the Employee Management System, comprehensive testing and refinement were conducted using JUnit.
- Each microservice (Employee Management, Department Management, Task Management, Performance Management) was independently tested using JUnit to validate the functionality of individual components.
- Test cases included scenarios for creating, updating, retrieving, and deleting records (CRUD operations).

14.1 JUnit for Employee_Capstone.

Controller Layer:

Finished after 2.348 seconds

Runs: 9/9 | Errors: 0 | Failures: 0

EmployeeControllerTest [Runner: JUnit 5] (0.195 s)
- testDeleteEmployee() (0.112 s)
- testGetEmployeeByCode() (0.007 s)
- testGetEmployeeById() (0.006 s)
- testGetAllService() (0.007 s)
- testUpdateEmployee() (0.010 s)
- testCreateEmployee() (0.004 s)
- testGetAllEmployees() (0.006 s)
- testGetEmployeeAndTask() (0.007 s)
- testGetEmployeeAndPerformance() (0.004 s)

Failure Trace

Service Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit X

Finished after 11.442 seconds

Runs: 5/5 Errors: 0 Failures: 0

Employee_Service_ImplTest [Runner: JUnit 5] (0.177 s)
- testDeleteEmployee() (0.080 s)
- testDeleteEmployeeNotFound() (0.014 s)
- testGetEmployeeById() (0.027 s)
- testUpdateEmployee_NotFound() (0.011 s)
- testCreateEmployee() (0.021 s)

Failure Trace

Repository Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit X

Finished after 7.147 seconds

Runs: 5/5 Errors: 0 Failures: 0

EmployeeRepositoryTest [Runner: JUnit 5] (1.311 s)
- Test for Creating the Employee and saving it into DB... (1.042 s)
- Test for Deleting the Employee of given Id from DB... (0.040 s)
- Test for Fetching all Employees from DB... (0.198 s)
- Test for fetching the Employee data of given Id from DB... (0.014 s)
- Test for Updating the Employee of given Id in DB... (0.010 s)

Failure Trace

14.2 JUnit for Department_Capstone.

Controller Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit X

Finished after 5.592 seconds

Runs: 6/6 Errors: 0 Failures: 0

DepartmentControllerTest [Runner: JUnit 5] (0.721 s)
- testDeleteDepartment() (0.276 s)
- testUpdateDepartment() (0.277 s)
- testGetDepartmentById() (0.011 s)
- testGetAllDepartments() (0.017 s)
- testCreateDepartment() (0.010 s)
- testGetDepartmentByCode() (0.117 s)

Failure Trace

Service Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit X

Finished after 1.967 seconds

Runs: 7/7 Errors: 0 Failures: 0

DepartmentServiceImplTest [Runner: JUnit 5] (1.772 s)
- testDeleteDepartment() (1.710 s)
- testUpdateDepartment() (0.016 s)
- testGetDepartmentById_NotFound() (0.013 s)
- testGetDepartmentById() (0.005 s)
- testGetAllDepartments() (0.006 s)
- testCreateDepartment() (0.006 s)
- testGetDepartmentByCode() (0.007 s)

Failure Trace

Repository Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit X

Finished after 6.566 seconds

Runs: 1/1 Errors: 0 Failures: 0

DepartmentRepositoryTests [Runner: JUnit 5] (0.864 s)
- givenDepartmentCode_whenFindByDepartmentCode_thenReturnDepartment() (0.864 s)

Failure Trace

14.3 JUnit for Performance_Capstone.

Controller Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit X

Finished after 5.485 seconds

Runs: 5/5 Errors: 0 Failures: 0

PerformanceControllerTest [Runner: JUnit 5] (0.535 s)
- testGetAllPerformances() (0.355 s)
- testUpdatePerformance() (0.142 s)
- testDeletePerformance() (0.012 s)
- testCreatePerformance() (0.007 s)
- testGetPerformanceById() (0.008 s)

Failure Trace

Service Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit ×

Finished after 2.114 seconds

Runs: 8/8 Errors: 0 Failures: 0

PerformanceServiceImplTest [Runner: JUnit 5] (1.926 s)
- testUpdatePerformance_Success() (1.854 s)
- testGetPerformanceById_IdNotFound() (0.019 s)
- testGetPerformanceById_Success() (0.008 s)
- testGetAllPerformances() (0.010 s)
- testDeletePerformance_IdNotFound() (0.006 s)
- testUpdatePerformance_IdNotFound() (0.004 s)
- testCreatePerformance() (0.006 s)
- testDeletePerformance_Success() (0.009 s)

Failure Trace

Repository Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit ×

Finished after 2.033 seconds

Runs: 3/3 Errors: 0 Failures: 0

PerformanceRepositoryTests [Runner: JUnit 5] (1.830 s)
- junit tetsing for save performance (1.816 s)
- junit testing for find performance by Id (0.009 s)
- Junit testing for delete performance by id (0.003 s)

Failure Trace

14.4 JUnit for Task_Capstone.

Controller Layer:

Problems Servers Terminal Data Source Explorer Properties Console JUnit ×

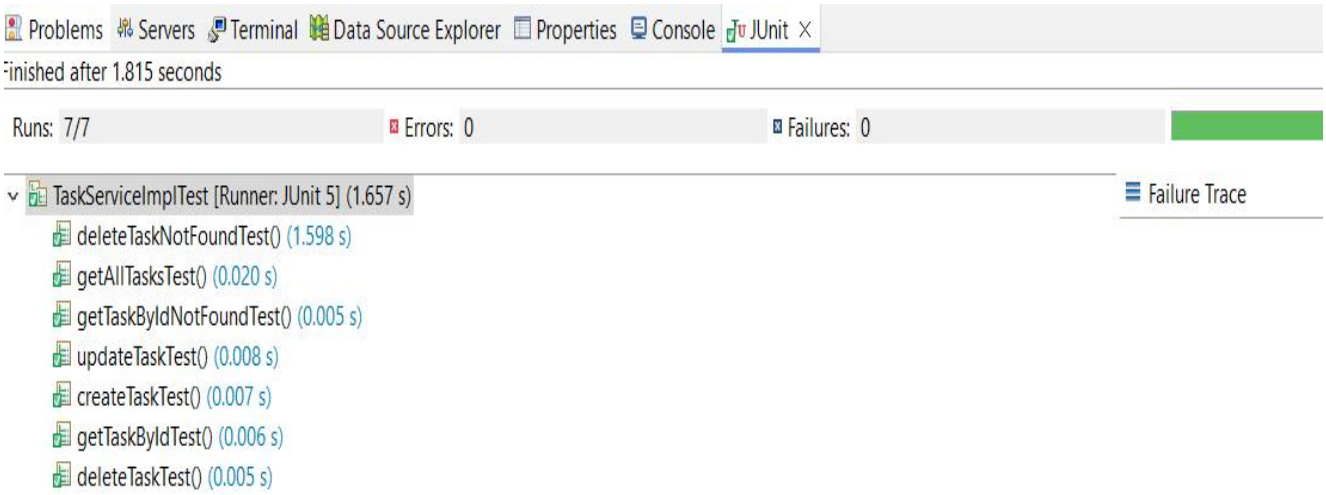
Finished after 5.355 seconds

Runs: 5/5 Errors: 0 Failures: 0

TaskControllerTest [Runner: JUnit 5] (0.536 s)
- getAllTasksTest() (0.333 s)
- updateTaskTest() (0.163 s)
- createTaskTest() (0.011 s)
- getTaskByIdTest() (0.008 s)
- deleteTaskTest() (0.012 s)

Failure Trace

Service Layer:



Repository Layer:

15. Conclusion

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. 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.

16. Future Enhancement

- **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.