**Task Management CIC2 Project**

**Document**

**A diagram of a software project

Description automatically generated**

**How to Set Up and Run the Application**

Before setting up the installation of the application, the following software and tools are installed on your system:

* Java Development Kit (JDK) 17 or higher: Required for compilation and running of the application, for this project I used Temurin 23.
* Apache Maven: for project build and dependency management.
* MongoDB: Each microservice uses MongoDB independently to persist its data.
* Docker: Required for containerizing and running the application components.
* Docker Compose: Facilitates the orchestration of multi-container Docker applications.
* Git: For cloning the project repository.

**Cloning the Repository**

**Task microservice –** https://github.com/Oaderogba2022/TaskMicroservice.git

**Project microservice –** https://github.com/Oaderogba2022/ProjectMicroservice.git

**Notification microservice -** <https://github.com/Oaderogba2022/NotificationMicroservice.git>

**Configuring MongoDB**

* Open a terminal or command prompt and pull the official MongoDB image from Docker Hub by running: docker pull mongo
* Run MongoDB in Docker After the image is downloaded, run MongoDB in a Docker container: docker run -d -p 27017:27017 --name mongodbbb mongo

Configure the application.properties

spring.data.mongodb.database=mongodbbb

spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

Each service will start on its designated port:

Task Service: http://localhost:8080

Project Service: http://localhost:8081

Notification Service: <http://localhost:8082>

**Add to application.properties**

Sample JSON:

<http://localhost:8080/tasks>

POST Method

{

"title": "Design Database Schema",

"description": "Create the initial database schema for the project.",

"status": "Pending"

}

<http://localhost:8081/projects>

POST Method

{

"name": "Website Development",

"description": "Develop a new company website.",

"startDate": "2025-01-06",

"endDate": "2025-03-15",

"tasks": ["67854375898787568075"]

}

http://localhost:8082/notifications

POST Method

{

"message": "The project 'Website Development' has been completed successfully.",

"recipient": "team@company.com",

"type": "Email",

"isSent": false

}

**Start up on Intelliji  
  
Detailed Explanation of the Architecture**

My application uses the microservices architecture and consists of three services independent of each other. The Task Service provides management over tasks by Task Id, title, description, and status. Similarly, the Project Service is responsible for projects with such attributes as Project Id, Name, Description, Start Date, End Date, and List of Task Id, and the Notification service is responsible for notification management by ID, message, recipient, type, and sending status. Each microservice is running independently, using a MongoDB database to keep its data separated.

These microservices communicate through RESTful APIs, passing on information when needed. For example, the Project Service can use the API of the Task Service to fetch details about the tasks and attribute those to a project. The Notification Service monitors events such as completions of tasks or updates of projects and notify users when and where necessary. This structure keeps the services loosely coupled while making sure they cooperate well. In this application, all microservices share a database, which simplifies data management and ensures consistency across services.

These services interact with each other to provide their functionalities. The Task Service and Project Service can communicate with each other synchronously. For instance, a newly created task in the Task microservice can be associated with a project by pulling task details through REST endpoints from the Project Service. Similarly, the Notification Service listens for specific events, such as task completions or project updates, and sends notifications accordingly. This communication pattern lets each service perform its tasks while being decoupled from the others, which helps to maintain modularity in the system.

Also in my microservices architecture, I used RabbitMQ which enabled my services to communicate asynchronously. It allows my services to communicate by sending and receiving messages using message queues. When a new task is created within the Task Service, it publishes a message to one RabbitMQ exchange. That message then route to a queue which the Project Service can consume to associate that task with a project. Similarly, the Notification Service would listen for events like completions of tasks or project updates by consuming messages from its designated queues, sending notifications out.

**Instructions for running the tests, including unit tests and integration tests.**

* Make sure project dependencies - Spring Boot, Mockito, Junit are correctly defined in pom file.
* Navigate to the test directory (src/test/java) and right click on a test class.

**Unit tests:**

Are in ServiceTest class.

They are separated from the rest of the code - TaskService is tested in isolation and all dependencies, like TaskRepository and RabbitTemplate, are mocked.

Run by default during test.

**Integration tests:**

Are in Task/Project/Notification ControllerIntegrationTest class.

They use MockMvc instance to simulate HTTP requests and test the layer of Task/Project/Notification Controller.

**Key Assertions and Verifications**

**Unit Tests:**

* Service logic is tested with a mocked repository and RabbitTemplate.
* Data assertions: task title, description, and status.

**Integration Tests:**

* HTTP request method testing: GET, POST, PUT, DELETE on /tasks/notification or project endpoint
* JSON responses will be asserted with jsonPath.

**Test Coverage**

**Controller Layer:**  
Task/Project/Notification ControllerTest covers the functionality of all the endpoints.

**Service Layer:**  
Task/Project/Notification ServiceTest covers business logic and interactions with RabbitMQ.  
Integration Testing:

**TaskControllerIntegrationTest:**

Covers the integration of the Task/Project/Notification controller with the mocked service layer.