Sample Task Management Microservice Design Document

1. Overview

taskmanagement**2. Entities (Java Objects)**

2.1. Task Entity (Task.java)

Field	Type	Description
id	Long	Auto-generated, Primary Key
title	String	Task title, 255 characters
description	String	Detailed description of the task
status	Enum	Task status: "Pending", "In Progress", "Completed"
priority	Enum	Task priority: "Low", "Medium", "High"
assignee_id	Long	Foreign Key to **User** entity (Many-to-One relationship)
project_id	Long	Foreign Key to **Project** entity (Many-to-One relationship)
due_date	Timestamp	Task deadline
created_at	Timestamp	Auto-generated, Task creation time
updated_at	Timestamp	Auto-updated, Last modified time

- **Relationships:**
    - **Many-to-One** with **User**: Each task is assigned to a single user.
    - **Many-to-One** with **Project**: Each task belongs to one project.

2.2. User Entity

Field	Type	Description
id	Long	Auto-generated, Primary Key
name	String	User name, 255 characters
email	String	Unique email, 255 characters
role	Enum	User role: "Admin", "Member"
created_at	Timestamp	Auto-generated, User creation time
updated_at	Timestamp	Auto-updated, Last modified time

- **Relationships:**

- **One-to-Many** with **Task**: A user can be assigned multiple tasks (but each task has only one user).
- **One-to-Many** with **Project**: A user (Admin) can create multiple projects.

2.3. Project Entity

Field	Type	Description
id	Long	Auto-generated, Primary Key
name	String	Project name, 255 characters
description	String	Project description
created_by	Long	Foreign Key to **User** entity (Many-to-One relationship)
created_at	Timestamp	Auto-generated, Project creation time
updated_at	Timestamp	Auto-updated, Last modified time

- **Relationships:**
  - **Many-to-One** with **User**: Each project is created by one user (Admin).
  - **One-to-Many** with **Task**: A project can have multiple tasks.

3. Entity Relationships

Entity	Relationship Type	Related Entity	Description
Task	Many-to-One	User	A task is assigned to one user (assignee)
Task	Many-to-One	Project	A task belongs to one project
User	One-to-Many	Task	A user can have multiple tasks
User	One-to-Many	Project	A user (Admin) can create multiple projects
Project	One-to-Many	Task	A project can have multiple tasks
Project	Many-to-One	User	A project is created by one user (Admin)

4. Operations

Task Operations

- **POST /tasks**: Create a new task.

  {

"title": "Complete API integration",

"description": "Integrate third-party payment gateway into the system.",

"status": "In Progress",

"priority": "High",

"assignee_id": 3,

"project_id": 2,

"due_date": "2024-09-30T18:00:00Z"

}

- **GET /tasks/{id}**: Fetch task details by ID.
- **PUT /tasks/{id}**: Update task details.
- **DELETE /tasks/{id}**: Soft delete a task.

User Operations

- **POST /users**: Create a new user.

  {

  "name": "Azhar Ahmed",

  "email": "azhar.ahmed@gmail.com",

  "role": "Developer"

  }

- **GET /users/{id}**: Fetch user details by ID.
- **PUT /users/{id}**: Update user details.
- **DELETE /users/{id}**: Soft delete a user.

Project Operations

- **POST /projects**: Create a new project.

  {

  "name": "Practice API ",

  "description": "Creating this API to serve as a hand on to stay relevant in the industry",

  "created_by": 1

  }

- **GET /projects/{id}**: Fetch project details by ID.
- **PUT /projects/{id}**: Update project details.
- **DELETE /projects/{id}**: Soft delete a project.

5. Use Cases

1. **Task Creation & Assignment:**
    - Admin assigns a user to a project and creates tasks for them.
    - The task is assigned a priority and a deadline.
2. **Task Progress Tracking:**
    - Users can update the status of their tasks as they work on them.
3. **User Role Management:**
    - Admins can create and assign roles to users, managing who can create projects and tasks.
4. **Project Management:**
    - Admins create and manage projects, with tasks linked to projects.

4. Phase-wise Implementation

4.1. Phase 1: Basic Implementation with H2 Database

- **Goal: Implement core functionalities (CRUD for tasks, users, projects) using an in-memory H2 database.**
- **Actions:**
    - **Create basic Spring Boot project.**
    - **Set up entities (Task, User, Project) with relationships.**
    - **Implement CRUD operations using Spring Data JPA.**
    - **Integrate H2 database for development and testing.**
    - **Deploy code to your personal Git repository for version control.**

4.2. Phase 2: Containerization with Docker

- **Goal: Containerize the application for easier deployment and scaling.**
- **Actions:**
    - **Create a Dockerfile for the Spring Boot application.**
    - **Build Docker images and run containers locally.**
    - **Push code and Docker setup to the Git repository.**

4.3. Phase 3: Kubernetes for Orchestration

- **Goal: Deploy the microservice on a Kubernetes cluster for scalability and load balancing.**
- **Actions:**
    - **Set up Kubernetes configuration files (Deployment, Service).**

- **Deploy the Docker container to a Kubernetes cluster (local or cloud).**
- **Push Kubernetes YAML files to the Git repository.**

4.4. Phase 4: Data Persistence with MySQL

- **Goal: Replace the in-memory H2 database with MySQL for persistent data storage.**
- **Actions:**
  - **Set up MySQL on the local machine or cloud (e.g., Amazon RDS).**
  - **Update Spring Boot configuration to use MySQL instead of H2.**
  - **Migrate data from H2 (if necessary).**
  - **Push code and configuration changes to the Git repository.**

4.5. Phase 5: Caching with Redis

- **Goal: Improve performance by introducing Redis caching for frequently accessed data.**
- **Actions:**
  - **Set up Redis in the microservice for caching task and project data.**
  - **Implement caching in the Spring Boot application using Spring Cache with Redis.**
  - **Push Redis-related code to the Git repository.**

4.6. Phase 6: Cloud Deployment (AWS)

- **Goal: Deploy the application on AWS with cloud-managed services.**
- **Actions:**
  - **Use AWS RDS for MySQL database.**
  - **Use AWS Elastic Kubernetes Service (EKS) for Kubernetes cluster management.**
  - **Implement AWS S3 for file storage (if needed for attachments).**
  - **Set up CloudWatch for monitoring and logging.**
  - **Push cloud-related deployment scripts to the Git repository.**

5. Version Control with Git

- **Set up a Git repository to track all changes throughout the implementation phases.**
- **Use branching strategies like feature/branch-name for new features and release/branch-name for deployments.**

- **Example structure:**

  - main

  - feature/task-crud

  - feature/docker

  - feature/kubernetes

  - feature/mysql

  - feature/redis

  - feature/aws

6. Technologies Used

- **Java 11 with Spring Boot**
- **Spring Data JPA for ORM and database interactions**
- **H2 Database for development**
- **MySQL for production**
- **Docker for containerization**
- **Kubernetes for orchestration**
- **Redis for caching**
- **AWS for cloud services (RDS, EKS, S3, CloudWatch)**