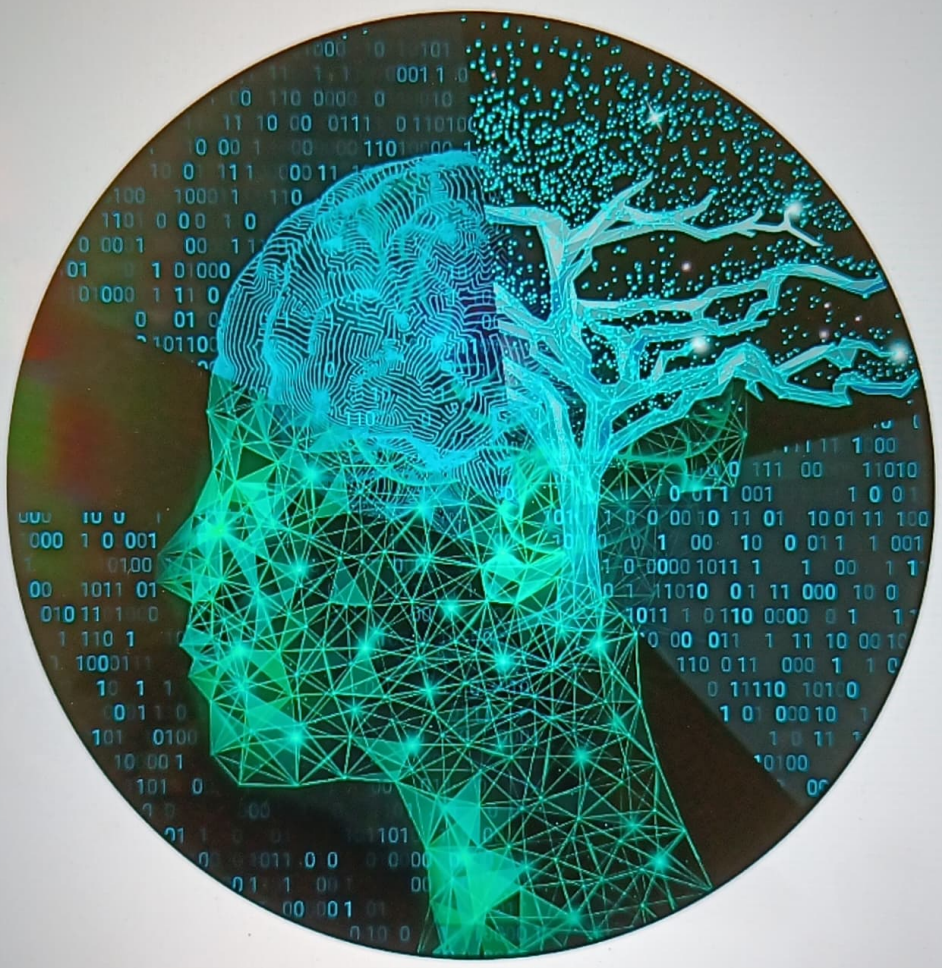
Deloitte.



Issue Tracking System Using Spring Boot, Microservices, and MySQL

Version Dated: 07.2024

Table of Contents

Problem Statement

Project Overview

This case study describes the development of an Issue Tracking System (ITS) based on a microservices architecture. The system will handle project details, issues related to these projects, and user assignments. The system will allow users to create, update, and manage issues with various statuses, priorities, and assignments associated with those projects, and assignees (users) who interact with the application. It will be developed using Spring Boot for creating microservices and MySQL for database management. The system will provide RESTful services using Spring Boot and store data in a MySQL database.

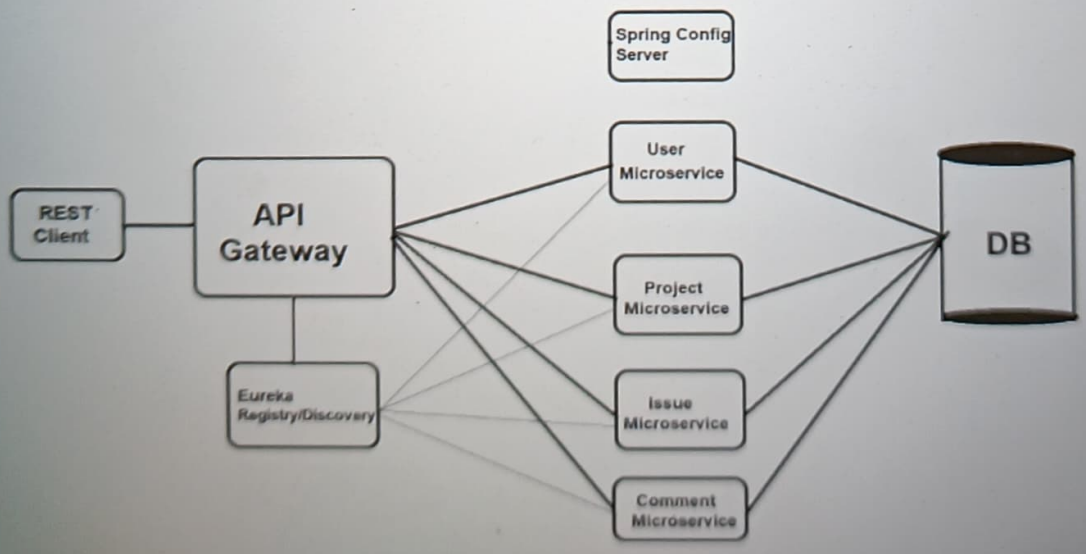
Objective

Develop a RESTful service using Spring Boot, which will interact with a MySQL database to manage issues effectively. The system should provide endpoints for adding, retrieving, updating, and deleting issue records.

Tools and Technologies

Spring Boot	Framework for building microservices
Spring Cloud	For microservices patterns like configuration management, service discovery
Spring Cloud Gateway	For API gateway service
MySQL	Database for storing data of different services
Eureka Server	For service discovery

Architecture



Functional Requirements

The system will be divided into several microservices, each handling a specific domain of the application.

Link to all Sample End Points:

References - Endpoints & DB.xlsx

User Management

1. Sign Up

- **Description:** Allows creation of a new user with details including name, password, and role
- **Inputs:** Name, password, role
- **Outputs:** Confirmation of user creation, user ID
- **Process:** Validates input and adds a new user to the database

2. Login

- **Description:** Validating user by ID and password to access project dashboard
- **Inputs:** User ID, password
- **Outputs:** redirects to Project Dashboard with project names auto populated on successful login else error is handled
- **Process:** Fetch user details and project name from the database tables

Project Management

1. Create Project

- **Description:** Allows creation of a new project with details such as project name, product owner, start date, and end date
- **Inputs:** Project name, product owner ID, start date, and end date
- **Outputs:** Confirmation of project creation, project ID
- **Process:** Validates input and adds a new project to the database

2. Retrieve Projects

- **Description:** Allows retrieval of all projects or a single project by ID
- **Inputs:** Project ID
- **Outputs:** List of projects or single project details
- **Process:** Fetches project details from the database

Issue Tracking

1. Create Issue

- **Description:** Allows creation of a new issue related to a project
- **Inputs:** Summary, project ID, description, priority, assignee ID, status, created date, last updated date, and comments
- **Outputs:** Confirmation of issue creation, issue ID
- **Process:** Validates input and adds a new issue to the database

2. Retrieve Issues

- **Description: Allows retrieval of all issues, a single issue by ID, issues by project, or issues by user (assignee)**
- **Inputs: Issue ID or project ID or user ID**
- **Outputs: List of issues or single-issue details**
- **Process: Fetches issue details from the database based on the provided criteria**

3. Update Issue

- **Description: Allows updating existing issue details**
- **Inputs: Issue ID, fields to update such as status, assignee**
- **Outputs: Updated issue details**
- **Process: Validates changes and updates the issue details in the database**

4. Add, Update, and Fetch Comments Related to an Issue

- **Description: Allows adding and fetching comments**
- **Inputs: Issue ID**
- **Outputs: Updated list of comments**
- **Process: Validates changes and updates in the comments table in the database**

Reporting and Analytics

1. View Insights

- **Description: Customizable dashboard views for users to monitor project and issue statuses, updates, and metrics**
- **Inputs: User preferences**
- **Outputs: Dashboard view**
- **Process: Retrieves and displays information based on user-configured settings and permissions**

Microservices Breakdown

- **Project Service:** Manages all project-related operations
- **Issue Service:** Handles issues within projects
- **User Service:** Manages user information and roles
- **Comment Service:** Manages comments related to an issue

Inter-Service Communication Between the Services

- Fetch all issues related to a project: Project Service: fetch project id of that project name, and calling issue service by passing project id to fetch all issues from Issue Service.
- Returns all issues assigned to a specific user: Fetch user Id from assignee name, calling issue service to fetch all issues tagged to that assignee).
- Fetch all comments related to an issue (fetch issue details in Issue Service, get issue id and call Comment Service to fetch all comments related to specific issue id).

API Specification

(Link is also shared above with sample data in table format and sample endpoint outcomes)

1. Project Service Endpoints

POST /projects: Create a new project
GET /projects: Retrieve all projects
GET /projects/{projectId}: Retrieve a project by ID

2. Project Service Endpoints

POST /issues: Create an issue within a project
GET / issues: List all issues for a project
GET /issues/{issueId}: Retrieve an issue by ID
PUT /issues/{issueId}: Update an issue
PUT /issues/{issueId}/comments: Update Comments

3. User Service Endpoints

POST /users: Add a new user (SignUp)
POST /users/login: Login User (Login)
GET /users: List all users
GET /users/{userId}: Get details of a specific user

4. Interservice Communication Endpoints

GET /projects/{projectId}/issues: Retrieve a project by ID
GET /users/{userId}/issues: Returns all issues assigned to a specific user

Database Structure

Each service will manage its own database schema, reflecting the principles of microservices architecture to ensure loose coupling and high cohesion.

- **Projects Database:** Projects table with fields id, name, description, and start_date
- **Issues Database:** Issues table with the fields that holds following data for - id, project_id, title, description, status, priority, and assignee_id
- **Users Database:** Users table with fields id, name, and email
- **Comments Database:** Comments table with fields commentId, issueId, text, and createdDate

Implementation Strategy

- **Microservices Setup:** Each microservice is set up as a separate Spring Boot application
- **Service Discovery:** Implement Eureka Server for dynamic service discovery
- **API Gateway:** A single-entry point for all clients. Routes requests appropriate microservices, handles failures, and provides some API aggregation
- **Inter-service Communication:** Services communicate using REST APIs, facilitated by client-side load balancing with Ribbon or Feign
- **Database Configuration:** Each service interacts with its own MySQL database instance
- **Testing and Deployment:** Each microservice is developed, tested, and deployed independently
- **Swagger:** Each of the above services should have swagger documentation of all APIs

Non-Functional Requirements

- The application should have low latency and high throughput.
- The application and data should be secured by Authentication.
- System should be scalable and maintainable.

Guidelines

- Make use of best coding practices.
- Develop the application in such a way that it facilitates easy plugging of other related modules/functionalities like getting real-time updates and targeted notifications in case of status changes, in-issue commenting, etc. while scaling the application at a later point in time.
- You can make assumptions and incorporate additional features or functionalities.
- A simple but working prototype is preferred over a non-working but good-looking application.
- The screenshots provided in this document are intended to help you understand and can be used as a reference to build the application.
- Add relevant comments for proper documentation.
- There would be a heavy focus on code quality and robustness of code in general.

Best Practices

Clear and Consistent Resource Naming	Use nouns that accurately reflect the resources your API manages (e.g., /products, /users)
Follow RESTful Principles	Design APIs around resources, using HTTP methods explicitly (GET for fetching, POST for creating, PUT/PATCH for updating, DELETE for removing)
HTTP Status Code	Use appropriate status codes to indicate API responses clearly
Embrace Dependency Injection (DI)	Use @Autowired to inject dependencies (services, repositories) into controllers. Promote loose coupling and testability
Exception Handling	Use @ControllerAdvice or @RestControllerAdvice to handle exceptions globally. Provide meaningful error responses that include status codes, error messages, and, if necessary, additional details

Milestones and Evaluation Criteria

S. No.	Details
Milestone 1	**Objective:** Create User Microservice with below endpoints created and tested using postman. **User Service Endpoints:** • **POST /users:** Add a new user. (SignUp) • **POST /users/login:** Login User (Login) • **GET /users:** List all users • **GET /users/{userId}:** Get details of a specific user
Milestone 2	**Objective:** Create Project Microservice with below endpoints created and tested using postman. **Project Service Endpoints:** • **POST /projects:** Create a new project • **GET /projects:** Retrieve all projects • **GET /projects/{projectId}:** Retrieve a project by ID • **PUT /projects/{projectId}:** Update a project • **DELETE /projects/{projectId}:** Delete a project
Milestone 3	**Objective:** Create Issue Microservice with below endpoints created and tested using postman. • **POST /issues:** Create an issue within a project • **GET / issues:** List all issues for a project • **GET /issues/{issueId}:** Retrieve an issue by ID • **PUT /issues/{issueId}:** Update an issue
Milestone 4	**Objective:** Create Comments Microservice with below endpoints created and tested using postman. • **POST /issues/{issueId}/comments:** Adds a new comment to a specific issue • **GET /issues/{issueId}/comments:** Retrieve all comments specific to an issue
Milestone 5	**Objective:** Implement the following as part of the **Microservices Setup.** • Implement Eureka Server for dynamic service discovery • Register all microservices on Eureka Service dashboard
Milestone 6	**Objective:** Implement below Interservice Communication endpoints. • GET /projects/{projectId}/issues: Retrieve a project by ID • GET /users/{userId}/issues: Returns all issues assigned to a specific user
Milestone 7	**Objective:** Use of ResponseEntity in all above microservices to effectively manage HTTP status codes and provide descriptive status messages.
Milestone 8	**Objective:** Implement API gateway to enable client-side load-balancing & accessing all Microservice endpoints through the API Gateway.
Milestone 9	**Objective:** Implement Swagger to enable API documentation and visualization.
Milestone 10	Push the build to your GitHub repository.

User Table			
userId (PK)	**name**	**password**	**role**
1	Alice Smith	abc123	productOwner
2	Bob Johnson	def456	assignee
3	Carol Lee	ghi789	assignee
4	Dave White	jkl012	productOwner
5	Eva Black	mno345	assignee
6	Frank Green	pqr678	assignee
7	Grace Hall	stu901	productOwner
8	Henry Adams	vwx234	assignee
9	Isla Fisher	yza567	productOwner
10	Jake Knox	bcd890	assignee

Project Table				
Id(PK)	**projectName**	**productOwner (FK-User)**	**startDate**	**endDate**
101	Project Alpha	1	1/1/2023	12/31/2023
102	Project Beta	4	2/15/2023	8/30/2023
103	Project Gamma	7	3/20/2023	9/15/2023
104	Project Delta	9	5/1/2023	1/1/2024
105	Project Epsilon	1	7/15/2023	12/20/2023

Issue Table Records

Id	summary	project	description	priority	assignee (FK-User)	status	created On	lastUpdated	comments
201	Login Feature	101	Implement login	HIGH	2	TO DO	1/10/2023	1/15/2023	Initial task creation
202	Payment Module	101	Setup payment gateway	MEDIUM	3	TESTING	1/12/2023	2/1/2023	Awaiting approval
203	Dashboard View	102	Fix dashboard refresh	LOW	2	DEVELOPMENT	2/20/2023	2/25/2023	Fixed refresh rate
204	User Management	103	Expand user management	HIGH	5	COMPLETED	3/1/2023	4/15/2023	Completed ahead of time
205	Notification System	104	Implement notifications	MEDIUM	6	TESTING	5/5/2023	6/1/2023	Needs additional testing
206	Export Data Feature	105	Address export issues	HIGH	8	TO DO	7/20/2023	7/22/2023	Export not working
207	Analytics Module	101	Build analytics module	LOW	10	DEVELOPMENT	1/25/2023	3/10/2023	High complexity
208	Mobile Interface	102	Improve mobile interface	MEDIUM	5	COMPLETED	2/28/2023	3/25/2023	Optimized for speed
209	API Development	103	Develop new APIs	HIGH	6	DEVELOPMENT	3/15/2023	4/10/2023	Finalizing documentation
210	Security Patch	104	Apply security patch	HIGH	8	TESTING			

Comments Table Records			
Id	issueId (FK-Issue)	text	createdDate
1	201	Test 1	1/10/2023
2	201	Update First	1/23/2023
3	201	Update 2	2/4/2023
4	207	Analytics - Update 1	7/12/2023
5	207	Analytics - Update 2	8/2/2023