**Software Requirements Specification (SRS)**

**Community Management System: "CommUnity"**

**1. Project Overview**

"CommUnity" is a modular, multi-tenant Community Management System designed to simplify housing society management. It provides an efficient interface for both administrators and residents to enhance interaction, management, and engagement within the community. The application will be built using Java Spring Boot for the backend and ReactJS for the frontend.

The platform offers features such as user authentication, society and resident management, event and notice management, complaint tracking, and online payment for maintenance bills. These functionalities aim to foster transparent communication, efficient management, and enhanced community engagement.

**2. Business Objectives**

1. **Simplify Community Management:** Provide tools to manage housing society operations seamlessly.
2. **Enhance Communication:** Improve interaction between residents and administrators through notices, events, and feedback mechanisms.
3. **Transparency:** Enable clear financial transactions with bill generation, online payment, and tracking.
4. **Safety and Accessibility:** Provide easy access to emergency contact information and enhance security measures.
5. **Scalability:** Design the platform to handle multiple societies and a growing user base efficiently.

**3. Stakeholders**

| **Role** | **Description** |
| --- | --- |
| End Users | Residents and admins interacting with the platform for various purposes. |
| Society Administrators | Manage residents, complaints, and financial operations. |
| Developers | Build and maintain the application. |
| Project Managers | Oversee timelines, deliverables, and resource allocation. |
| QA/Testers | Ensure the system meets quality and performance standards. |

**4. Functional Requirements**

**4.1 User Authentication and Registration**

* User registration with email and password.
* JWT-based authentication and authorization.
* Profile management for residents and administrators.

**4.2 Resident and Society Management**

* Database schema for societies, residents, and flats.
* CRUD operations for resident and society details.

**4.3 Event and Notice Management**

* Admin interface for creating, updating, and deleting events and notices.
* Residents can view events, provide feedback, and interact with notices.

**4.4 Complaint and Service Request Management**

* Interface for residents to log complaints and service requests.
* Admin functionality to track, monitor, and resolve complaints.

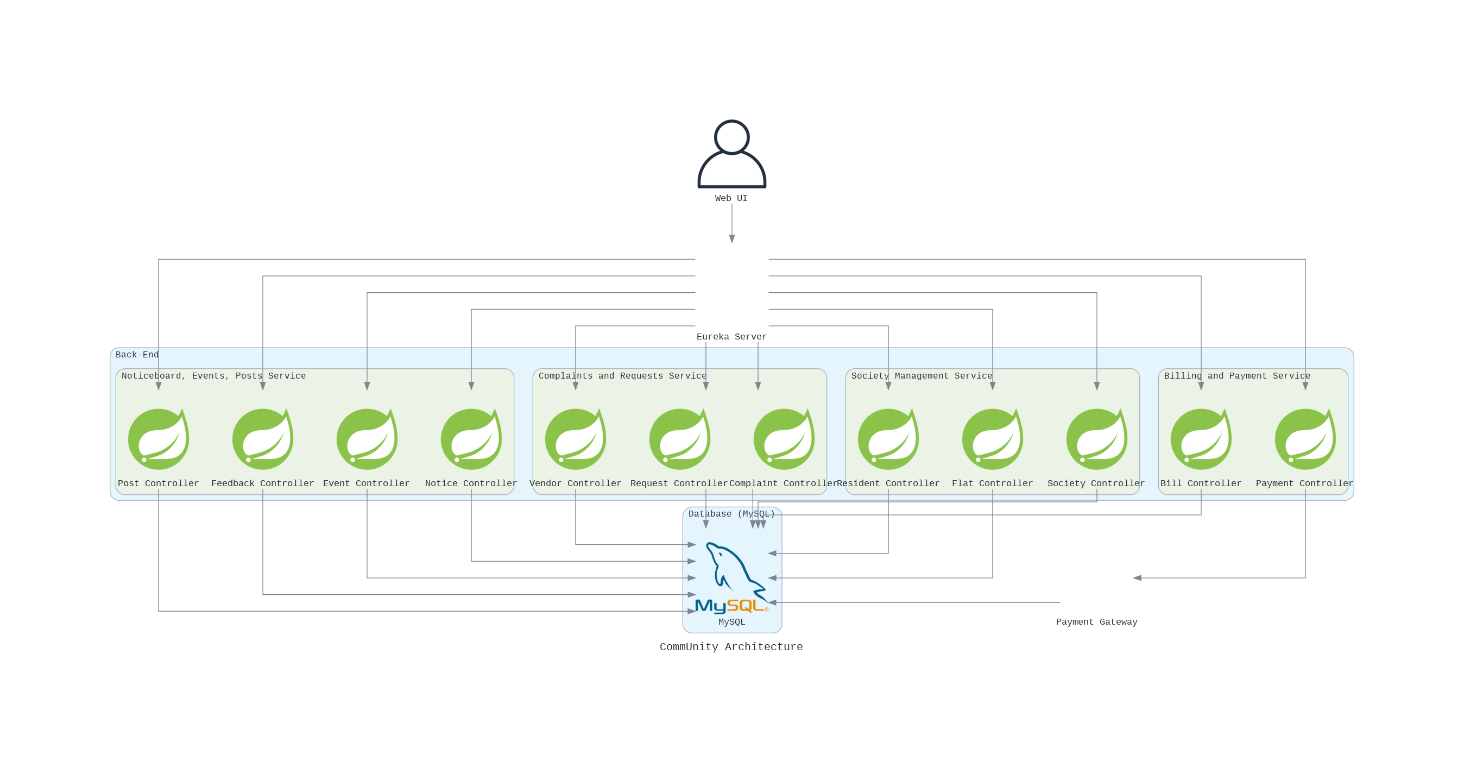
**4.5 Maintenance Payment System**

* Razorpay integration for online maintenance payments. (link)
* Automated bill generation and payment history tracking.
* Admin panel for monitoring payment statuses.

**5. Non-Functional Requirements**

1. **Performance:** Response time must not exceed 1 second for 90% of requests.
2. **Scalability:** Support up to 100,000 concurrent users across multiple societies.
3. **Security:**
   * Secure payment processing with Razorpay.
   * Data encryption at rest and in transit.
   * Protection against SQL injection, XSS, and CSRF attacks.
4. **Availability:** Ensure 99.9% uptime.
5. **Usability:**
   * Intuitive user interface compatible with mobile and desktop devices.
   * Responsive design for enhanced accessibility.
6. **Project Architecture**

**Diagram:**

****

**6.1 Frontend (ReactJS)**

* **State Management**: Use **Redux** for handling global state, ensuring efficient data management and synchronization across components.
* **Integration with Backend**: Communicate with backend using **RESTful APIs** for data exchange, with **Axios** or **Fetch** to handle requests.
* **UI/UX**: Responsive design with **React Router** for seamless navigation and component-based UI.
* **Authentication**: Integrate **JWT-based authentication** to manage user sessions and provide access control.

**6.2 Backend (Spring Boot)**

* **RESTful API Design**: Design REST APIs using **Spring Boot** to handle client requests for operations such as user management, event creation, and payment processing.
* **Business Logic**: Implement core business logic for various features like event scheduling, complaint resolution, and payment processing.
* **Authentication & Security**: Implement **JWT-based authentication** for secure access and role-based authorization using **Spring Security**.
* **Database Integration**: Use **JPA (Java Persistence API)** for seamless database integration and entity management.
* **Scalability**: Design with scalability in mind to handle multiple societies and users, possibly using **Spring Cloud** for microservices-based architecture in the future.

**6.3 Database (MySQL)**

* **Relational Schema**: Use **MySQL** to store structured data, with tables for users, societies, events, payments, complaints, and more.
* **Entities**:
  + **Users**: Information about residents, admins, and staff members.
  + **Societies**: Details about each housing society, including name, location, and associated residents.
  + **Events**: Scheduled events and activities, with time, location, and participants.
  + **Payments**: Records of online payments for maintenance bills, including payment status and amounts.
  + **Complaints**: Track complaints, their status, and resolution progress.

**6.4 Hosting and Deployment**

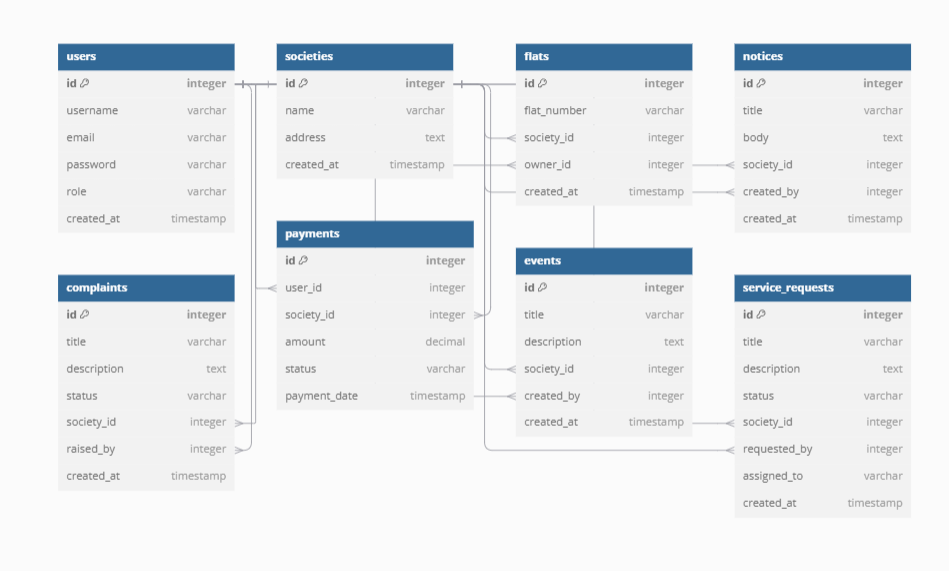
* **Backend**: Host the backend on **AWS** or **Azure** using services like EC2 or Azure App Service for high availability and scalability.
* **Frontend**: Deploy the frontend on a **CDN** (Content Delivery Network) for fast global access and better performance.
* **CI/CD Pipelines**: Implement **CI/CD** pipelines using tools like **Jenkins**, **GitHub Actions**, or **GitLab CI** for automated build, testing, and deployment.
* **Containerization**: Consider containerizing the backend with **Docker** and using **Kubernetes** for orchestration if scaling becomes necessary.

**7. Database Schema Overview**

**Key Tables:**

|  |  |
| --- | --- |
| **Table Name** | **Fields** |
| Users | user\_id (PK), email, password, name, profile\_picture, role |
| Societies | society\_id (PK), name, address, created\_by (FK) |
| Flats | flat\_id (PK), society\_id (FK), resident\_id (FK), flat\_number |
| Events | event\_id (PK), society\_id (FK), name, description, date\_time |
| Complaints | complaint\_id (PK), flat\_id (FK), description, status, assigned\_to |
| Payments | payment\_id (PK), flat\_id (FK), amount, status, timestamp |

**DB Diagram :**

****

**8. Milestones and Timeline**

**Milestone 1 (Weeks 1-2): User Authentication and Registration**

* Features: Registration, login, and JWT-based authentication.
* Evaluation: Successful login and registration for both admins and residents.

**Milestone 2 (Weeks 3-4): Resident and Society Management**

* Features: CRUD operations for societies and residents.
* Evaluation: Proper association of residents with societies.

**Milestone 3 (Weeks 5-8): Event and Notice Management**

* Features: Event creation, feedback collection, and notice updates.
* Evaluation: Residents access event details and provide feedback.

**Milestone 4 (Weeks 9-10): Maintenance Payment System**

* Features: Razorpay integration, bill generation, and payment tracking.
* Evaluation: Residents can pay bills; admins track payment statuses.

**9. Constraints**

1. Development must be completed within 10 weeks.
2. Technology stack includes ReactJS, Spring Boot, and MySQL/PostgreSQL.

**10. Assumptions**

1. Users have basic technical knowledge to navigate the system.
2. Internet access is required for system operation.
3. Third-party APIs like Razorpay will function as expected.

**11. Workflow Diagram**

Include the workflow diagram based on the project requirements (to be developed).

