**Business Requirements Document (BRD)**

**Project Title:** CommUnity: Seamless Community Interaction and Management

**1. Project Overview**

CommUnity is a modular, API-centric, multi-tenant Community Management System designed to simplify housing society management by connecting members and streamlining daily activities. The platform provides an efficient interface for both administrators and residents, facilitating interaction, management, and engagement within the community.

CommUnity offers tools for administrators to manage accounts, post notices, generate and track maintenance bills, and manage society profiles. Additionally, admins can handle complaints, configure emergency contacts, maintain event calendars, and review feedback. For residents, CommUnity enables communication with fellow members, access to a digital noticeboard, online bill payments, complaint submissions, event participation, and more.

**2. Business Objectives**

* **Streamline Community Management:** Provide a platform that simplifies housing society management and enhances communication between residents and administrators.
* **Enhance Resident Engagement:** Increase resident participation in community activities through events, polls, clubs, and social interactions.
* **Improve Financial Transparency:** Facilitate transparent financial transactions with easy bill generation, online payments, and payment tracking.
* **Strengthen Security and Emergency Preparedness**: Ensure easy access to emergency contacts and security personnel information, enhancing the overall safety and security within the community.

**3. Stakeholders**

* **Administrators:** Manage community operations, financials, and resident engagement.
* **Residents**: Participate in community activities, pay maintenance bills, raise complaints, and engage with fellow residents.
* **Security Personnel:** Provide safety services within the community and interact with residents and admins as needed.

**4. Functional Requirements**

**4.1 User Management**

**User Registration:** Residents, non-residents, and administrators can register on the platform.

**Admin Approval:** Admins approve resident registrations to join the community.

**Profile Management:** Users can view and update their personal details.

**4.2 Noticeboard and Notifications**

**Noticeboard Access:** Residents can view past and current notices, circulars, and announcements on a digital noticeboard.

**4.3 Maintenance and Payment Management**

**Bill Generation**: Admins can generate maintenance bills for residents.

Online Payments: Residents can view bills and make payments via Credit or Debit cards.

**Payment Tracking:** Admins can track bill payments and identify overdue accounts.

Payment Receipts: Residents receive instant payment receipts upon successful transactions.

**External API:** We are using RazorPay API for making payment and transactions.

Link: <https://api.razorpay.com/v1>

**4.4 Complaint and Feedback Management**

**Raise Complaints:** Residents can submit complaints at the community or personal level.

**Complaint Tracking**: Residents can track the status of their raised complaints.

**Complaint Resolution:** Admins can manage and resolve raised complaints.

**Event Feedback:** Residents can submit feedback on community events.

**Handle Feedback:** Admins review and use feedback for improvements.

**4.5 Event Management**

**Event Calendar:** Admins can create and update an event calendar for residents.

**Access Event Calendar**: Residents can view the event calendar and stay updated on future events.

**External API:** We are using Cloudinary API for getting urls for the images uploaded by the administrator.

**LINK:** <https://api.cloudinary.com/v1_1/digbbzwlfx/image/upload>

**4.6 Commercial Posts and Recommendations**

**Admin Commercial Posts:** Admins can post commercial content visible to all residents.

**Resident Recommendations:** Residents can post personal recommendations or share interesting content.

**'I Am Interested' Poll:** Residents can engage with commercial posts by marking interest via a poll.

**4.7 Emergency Contact Management**

**Configure Emergency Contacts:** Admins can maintain and update a list of local emergency contacts.

**Access Emergency Contacts**: Residents can view and access emergency contact details.

**4.8 Security Service Functionality**

**Security Personnel Allocation:** Admins can allocate security personnel to specific blocks.

**View Security Details**: Residents can view details of security personnel assigned to their block.5. Non-Functional Requirements

**5. Non-Functional Requirements**

**5.1 Security**

* **Basic Encryption:** Sensitive data is encrypted in transit using HTTPS.
* **Role-Based Access:** Implement role-based access control for authorized user actions.

**5.2 Usability**

* **User Interface**: The application will have an intuitive and clean design, optimized for desktop use.
* **Responsiveness:** The application will be fully responsive across devices and screen sizes.
* **Basic Accessibility:** The application will cater to users with varying technical abilities.

**5.3 Maintainability**

* **Code Simplicity:** The code will be modular, well-organized, and documented for easy maintenance.
* **Automated Testing:** Implement basic automated tests to ensure stability and ease future modifications.

**5.4 Scalability**

* **Basic Scalability:** The system will scale horizontally to accommodate more users and communities.
* **Elasticity:** The system will dynamically allocate resources to maintain performance during peak usage.

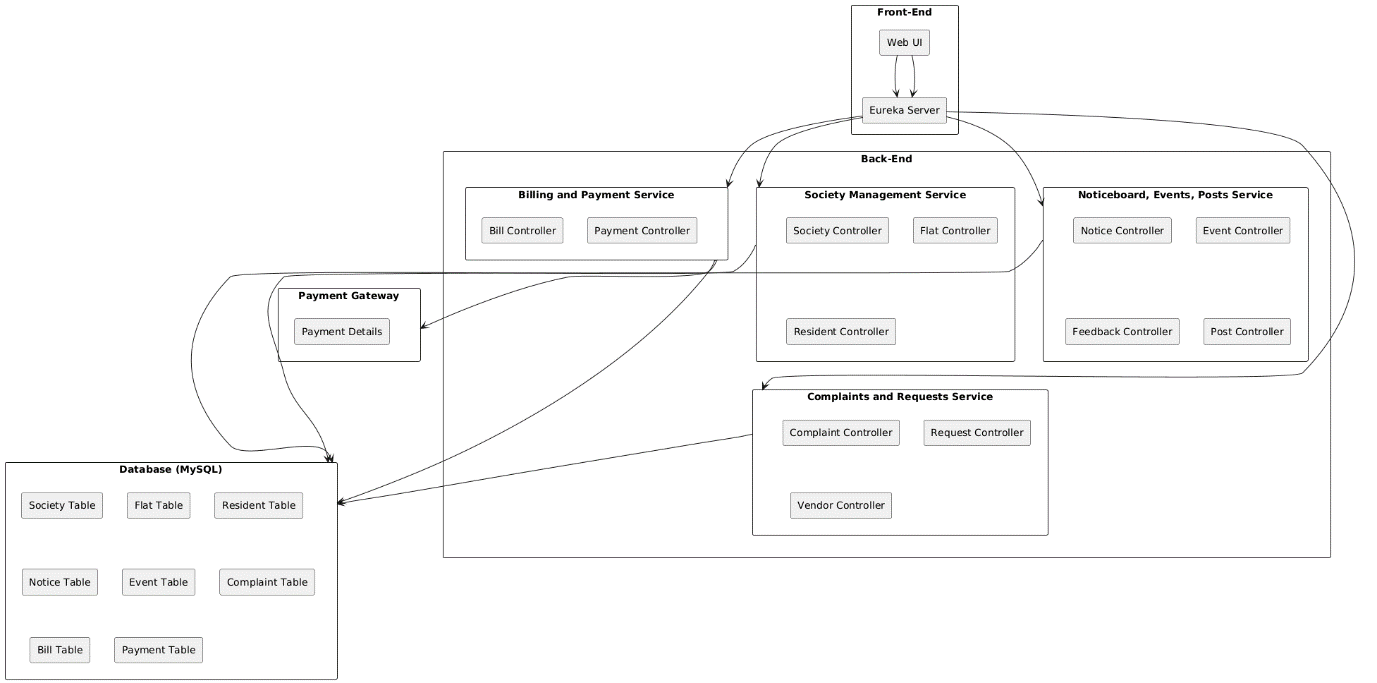
**5.5 Interoperability**

* **API Integration:** The system will expose RESTful APIs for integration with other services.
* **Modular Design:** The architecture will support modular components for easy integration of new features.

**6. Project Architecture**

The project architecture follows a microservices approach with a front end built using React and Redux for state management, and a back end developed with Spring Boot. The system leverages multiple Eureka server-based microservices to handle core functionalities such as society management, noticeboard and event management, complaint handling, and billing & payment processing. MongoDB is used for its scalability and flexibility in managing community-related data.

* **Front End**: React with Redux for state management.
* **Back End**: Spring Boot with RESTful APIs to manage the business logic.
* **Microservices**: Registered via Eureka servers for service discovery and load balancing.
* **Database:** MySQL for efficient data storage and retrieval, managing entities like user profiles, notices, events, payments, and complaints.



**7. Database Schema Overview**

The application will include the following tables in its MySQL database schema:

**Society Table:** Stores information about housing societies with societyId (Primary Key), name, phoneNo, societyName, address, city, district, postal, email.

**Resident Table:** Stores details of residents living in the society with fields residentId (Primary Key), name, phoneNo, flatNo, postal, email, societyId (Foreign Key), flatId (Foreign Key), role (resident/admin).

**Flat Table:** Contains data related to flats within a society with fields flatId (Primary Key), flatNo, societyId (Foreign Key).

**Emergency Contact Table:** Holds emergency contact numbers relevant to the society with fields emergencyId (Primary Key), personName, serviceType, phoneNo, societyId (Foreign Key).

**Security Details Table:** Stores security personnel details assigned to blocks within the society with fields securityId (Primary Key), name, phoneNo, blockNo, societyId (Foreign Key).

**Notice Table:** Maintains notices and circulars posted in the society with fields noticeId (Primary Key), heading, content, datePosted, noticeImage, societyId (Foreign Key).

**Event Table:**

Stores information about events happening within the society with fields eventId (Primary Key), eventName, eventDate, eventDetails, eventImage, societyId (Foreign Key).

**Feedback Table:**

Collects feedback from residents regarding events with fields feedbackId (Primary Key), eventId (Foreign Key), content.

**Request Table:** Manages service requests raised by residents, linked to vendors with fields requestId (Primary Key), address, description, status, vendorId (Foreign Key).

**Vendor Table:** Stores vendor information who provide services to the society with fields vendorId (Primary Key), name, company, email, service, phoneNo, societyId (Foreign Key).

**Post Table:** Handles social posts created by residents with fields postId (Primary Key), content, title, likeCount, postImage, societyId (Foreign Key).

**Complaint Table:** Tracks complaints raised by residents regarding society services or issues with fields complaintId (Primary Key), residentId (Foreign Key), flatNo, personName, title, description, status, societyId (Foreign Key).

**Parking Table:** Contains information about parking spaces allocated to flats with fields parkingId (Primary Key), flatNo, parkingNo, flatId (Foreign Key), societyId (Foreign Key).

**Bill Table:** Manages maintenance bills for flats with fields billId (Primary Key), flatId (Foreign Key), amount, dueDate, status.

**Payment Table:** Tracks payments made by residents for their maintenance bills with fields paymentId (Primary Key), billId (Foreign Key), amountPaid, date.



**8. Assumptions**

* Stable Internet Connectivity: Users are expected to have access to stable internet connections to interact with the application effectively.
* Regular Input of Data: Users will regularly update their health, diet, and other relevant data to ensure accurate insights and recommendations.
* Integration with External APIs: The application will integrate with external APIs for functionalities like payment processing, WhatsApp notifications, and possibly other external services as needed.

**9. Constraints**

* Development and Deployment Timelines: Project milestones and deadlines must be adhered to, ensuring that the application is delivered and deployed within the specified time frame.
* Scalability and Performance: The system must be designed to handle varying loads, ensuring performance and scalability as the number of users and data volume grows.
* Compliance and Security: The application must comply with relevant data protection regulations and ensure the security of user data.

**10. Data Requirements**

* **User Data:** Includes user authentication details, profile information, contact details, and roles. This data is essential for user management, authentication, and authorization.
* **Society Data:** Information related to societies, including society details, emergency contacts, security personnel, parking information, and notices.
* **Resident Data**: Profiles of residents, including personal details, flat assignments, and roles within the society.
* **Flat Data**: Details about flats, including flat numbers and their association with societies and residents.
* **Notice Data:** Notices posted by societies, including content, headings, and posting dates.
* **Event Data:** Information about events organized by societies, including event details and dates.
* **Feedback Data**: Feedback from residents related to events and other aspects of the society.
* **Post Data**: Posts made by residents, including content and type of post.
* **Complaint Data:** Records of complaints raised by residents, including descriptions, statuses, and related societies.
* **Request Data**: Requests made by residents, including descriptions, statuses, and vendor assignments**.**
* **Vendor Data:** Details about vendors hired by societies, including service types and contact information.
* **Bill Data:** Billing information for flats, including amounts, due dates, and statuses.
* **Payment Data:** Payment details related to bills, including amounts paid and dates.
* **Parking Data**: Information about parking assignments, including details of parking spots and the number of vehicles.

**11. Workflow Diagram**

