Software Requirements Specification

for

Community Management System

Prepared by

Ahmed Mustafa, Haziq Naeem, Muneeb-ul-Islam

NUCES ISLAMABAD

20-3-2025

1. Table of Contents

---

1. Introduction

1.1 Purpose

The **Communi3y** is Community Management System (CMS) that is a web-based application designed to streamline property management, public service requests, digital booking, facility reservations, and incident reporting within a community.

1.2 Document Conventions

This document follows the IEEE SRS template format.

1.3 Intended Audience and Reading Suggestions

- **Developers & Database Administrators** – Understand system functionalities.
- **Project Managers & Scrum Masters** – Track development milestones.
- **Residents & Admins** – Comprehend system capabilities.

1.4 Product Scope

The system provides **digital solutions** to manage properties, request public services, participate in digital voting, reserve community facilities, and report incidents. The backend will be implemented using **Next.js, Node.js, and MySQL**.

1.5 References

- IEEE 830-1998 Software Requirements Specification Standard.
- Project Diagrams: Use Case Diagram, Sequence Diagram, Class Diagram.

---

2. Overall Description

2.1 Product Perspective

This system is a standalone web-based application that digitalizes community management operations, replacing manual processes.

2.2 Product Functions

- **Property Registration**
- **Public Service Requests**
- **Digital Voting System**
- **Recreation Facility Reservation**
- **Crime & Incident Reporting**

2.3 User Classes and Characteristics

- **Residents** – Property owners or tenants accessing services.
- **Admins** – Management personnel handling operations.

2.4 Operating Environment

- Web-based system accessible via modern browsers.
- Hosted on a **local server infrastructure**.
- Uses **Html, CSS and React(if applicable at the end of sprint 3) (Frontend), Node.js (Backend), MySQL (Database)**.

2.5 Design and Implementation Constraints

- The system should be deployed on a **MySQL database**.
- Must follow **security protocols** for data encryption and access control.

2.6 User Documentation

- **User Manual** (for residents & admins)
- **API Documentation** (for developers)

2.7 Assumptions and Dependencies

- Reliable **internet connection** for cloud-based functionality.
- Compliance with **local property laws** for registration validation.

3. External Interface Requirements

3.1 User Interfaces

- **Web-based UI** developed using **HTML, CSS and REACT.js (applicable at the end of sprint 3)**.
- Interactive forms for **property registration, voting, and service requests**.

3.2 Hardware Interfaces

- Deployable on **standard server hardware**.
- Must support **cloud hosting** for scalability.

3.3 Software Interfaces

- Integration with **MySQL database**.
- **Node.js for backend operations**.

3.4 Communications Interfaces

- Email & SMS notifications for critical updates.
- Secure **HTTPS** for data transmissions.

4. System Features

- ## 4.1 User Stories & Acceptance Criteria

GWT-Formatted User Stories with Pre/Postconditions

1. Register Property

Scenario: Register a property in the system
Given the resident is logged in and on the property registration page,
And the resident has valid property documents ready,
When the resident fills in property details and uploads the documents,
Then the system should validate the data and documents,
And the registration request should be accepted,
And the resident should receive a confirmation notification.

Preconditions:

- Resident has an active user account.

- Property is not already registered.

Postconditions:

- Property is linked to the resident's account.
- Confirmation is recorded in the resident's notification history.

---

2. Manage Resident Data

Scenario: Add, update, and manage resident data
Given the admin is logged in with appropriate access rights,
When the admin views, edits, or deletes a resident profile,
Then the changes should be reflected immediately in the database,
And a confirmation message should be displayed.

Preconditions:

- Admin is authenticated and authorized.

Postconditions:

- Database reflects updated resident information.
- Action is logged with a timestamp.

---

3. Make Payment

Scenario: Pay society fees and bills
Given the resident is logged in and has outstanding bills,
When the resident selects a bill and completes a payment through a valid method,
Then the payment should be processed securely,
And a digital receipt should be generated,
And the payment status should update in the dashboard.

Preconditions:

- Resident has valid payment method linked.
- Bill is generated and pending.

Postconditions:

- Payment is recorded in the system.
- Dashboard shows updated payment status.

---

4. Track & Update Payments

Scenario: Admin monitors and updates payment records
Given the admin is authenticated and viewing a resident's payment history,
When the admin updates the payment status manually,
Then the changes should be saved,
And a timestamped log should be created.

Preconditions:

- Admin has access to financial records.

Postconditions:

- Updated payment status visible in resident record.
- Audit trail log is updated.

---

5. Manage Bills

Scenario: Admin generates and sends bills
Given the admin is on the bill generation module,
When the admin enters or adjusts bill details (amount, due date, etc.),
Then bills should be assigned to the appropriate residents,
And a notification should be sent.

Preconditions:

- Admin has access to billing module.

Postconditions:

- Bill is stored in system.
- Notification reaches residents via email/dashboard.

---

6. Book Parking & Reserve Facility

Scenario: Reserve community resources
Given the resident is logged in and on the reservation portal,
When they select an available parking or facility slot,
Then the resource should be marked as reserved,
And the resident should receive a confirmation.

Preconditions:

- Slots are available.
- Resident has no conflicting reservations.

Postconditions:

- Resource is booked.
- Reservation log is updated.

---

7. Submit Maintenance Request

Scenario: Request property maintenance
Given the resident is logged in and has identified an issue,
When the resident fills out a request and uploads supporting media,
Then the system should log the request and assign a ticket number,
And the resident should see the request in their history and track the status.

Preconditions:

- Resident account is active.
- Issue is within maintenance scope.

Postconditions:

- Request is logged in system.
- Resident receives ticket and status tracking.

---

8. Schedule Infrastructure Maintenance

Scenario: Schedule community infrastructure maintenance
Given the admin is logged in and on the maintenance module,
When the admin schedules a new task with date, time, and location,
Then the task should be assigned to the relevant maintenance team,
And progress should be tracked until completion.

Preconditions:

- Admin has maintenance scheduling permissions.
- Teams are available.

Postconditions:

- Maintenance task created.
- Status updates reflected in the system.

---

9. Manage Event RSVPs

Scenario: RSVP to community events
Given the resident is logged in and on the events page,
When they select an event and confirm attendance,

Then the RSVP should be saved,
And the resident should receive a reminder or event pass.

Preconditions:

- Event is upcoming.
- RSVP deadline not passed.

Postconditions:

- Resident marked as attendee.
- Reminder/event pass is sent.

---

10. Send & Receive Notifications

Scenario: Admin sends notifications to residents
Given the admin is logged in and on the notification panel,
When they draft and send a notification,
Then it should appear in residents' dashboards/inboxes,
And the system should log the timestamp and audience.

Preconditions:

- Admin has messaging privileges.

Postconditions:

- Notification is delivered.
- Message log is updated.

---

11. Request Public Service

Scenario: Submit a public service request
Given the resident is logged in and needs a public service,
When they submit a request with description and location,
Then a request ticket should be generated,
And the resident should be able to track the status.

Preconditions:

- Service is available for request.

Postconditions:

- Ticket is generated.
- Request is tracked in the system.

---

12. Report Crime or Incident

Scenario: Report a community crime or incident
Given the resident is logged in and has witnessed an incident,
When they report the details and location,
Then the system should assign it to the concerned authority,
And send an acknowledgment.

Preconditions:

- Reporting feature is available to residents.

Postconditions:

- Report is saved and sent to authorities.
- Acknowledgment sent to resident.

---

13. Submit Public Feedback

Scenario: Submit community feedback
Given the resident is logged in and on the feedback module,
When they write and submit feedback,
Then the system should acknowledge the submission,
And the feedback should reflect in the admin analytics.

Preconditions:

- Resident is allowed to give feedback.

Postconditions:

- Feedback is logged.
- Analytics updated.

---

14. Digital Voting

Scenario: Participate in digital voting
Given the resident is logged in and there is an active poll,
When they view and submit their vote,
Then the system should confirm submission,
And the vote should remain anonymous.

Preconditions:

- Poll is active.

- Resident is eligible to vote.

Postconditions:

- Vote is cast anonymously.
- Confirmation message displayed.

---

15. Manage Institutes & Healthcare

Scenario: Manage education and healthcare facilities
Given the admin is logged in and on the institute/healthcare module,
When the admin adds or edits profiles and schedules,
Then the updates should reflect in the community portal.

Preconditions:

- Admin has relevant access.

Postconditions:

- Profiles and schedules are updated.
- Portal shows current information.

---

5. Nonfunctional Requirements

5.1 Product Requirements

- The system should support **1000+ concurrent users**.
- Should process **real-time transactions**.

5.2 Organizational Requirements

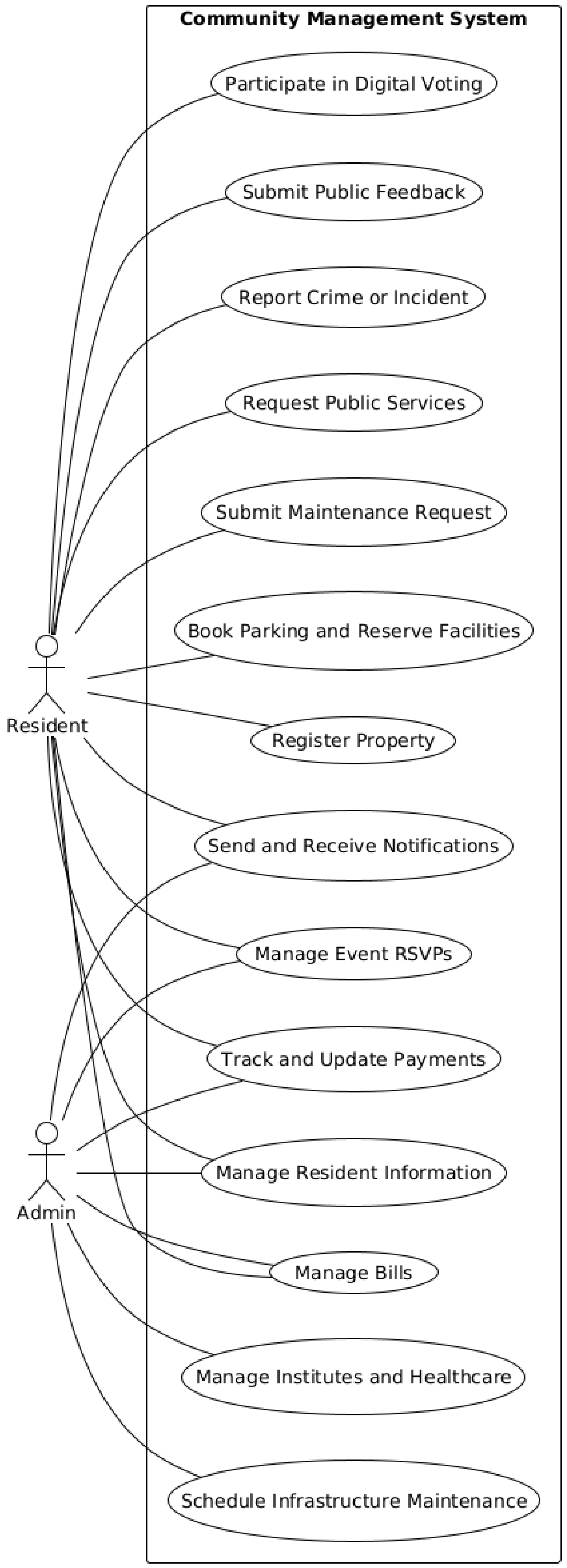
- Uses **Agile methodology** for iterative development.

5.3 External Requirements

- Should comply with **GDPR & local property laws**.

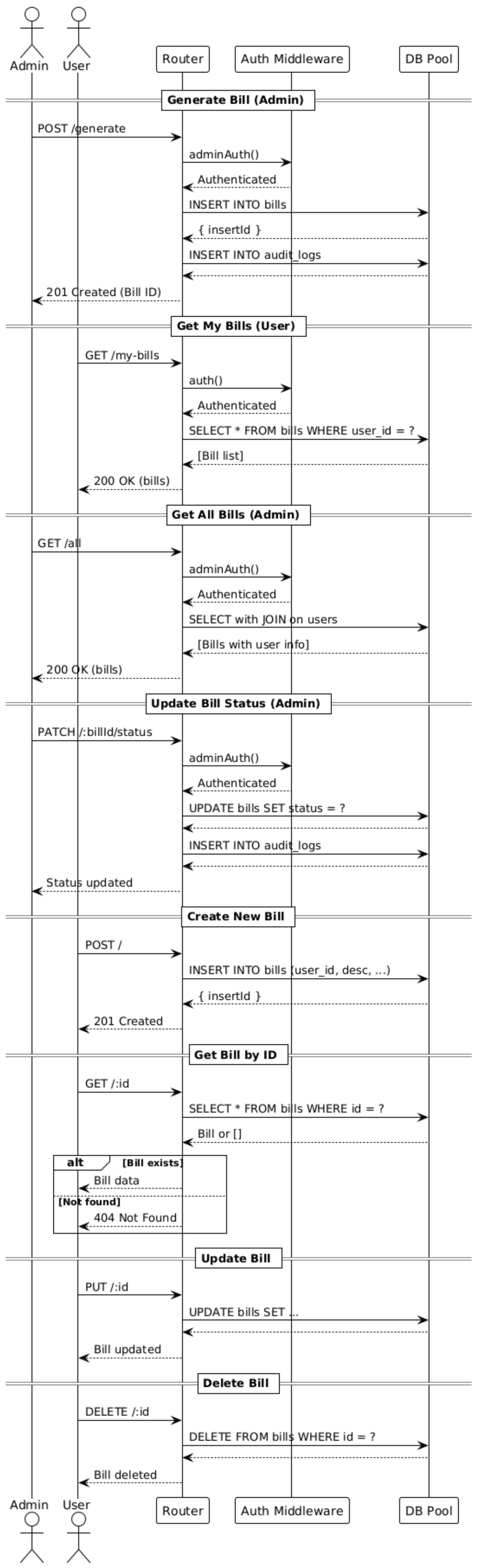
---

6. Use Case Diagram & User Stories

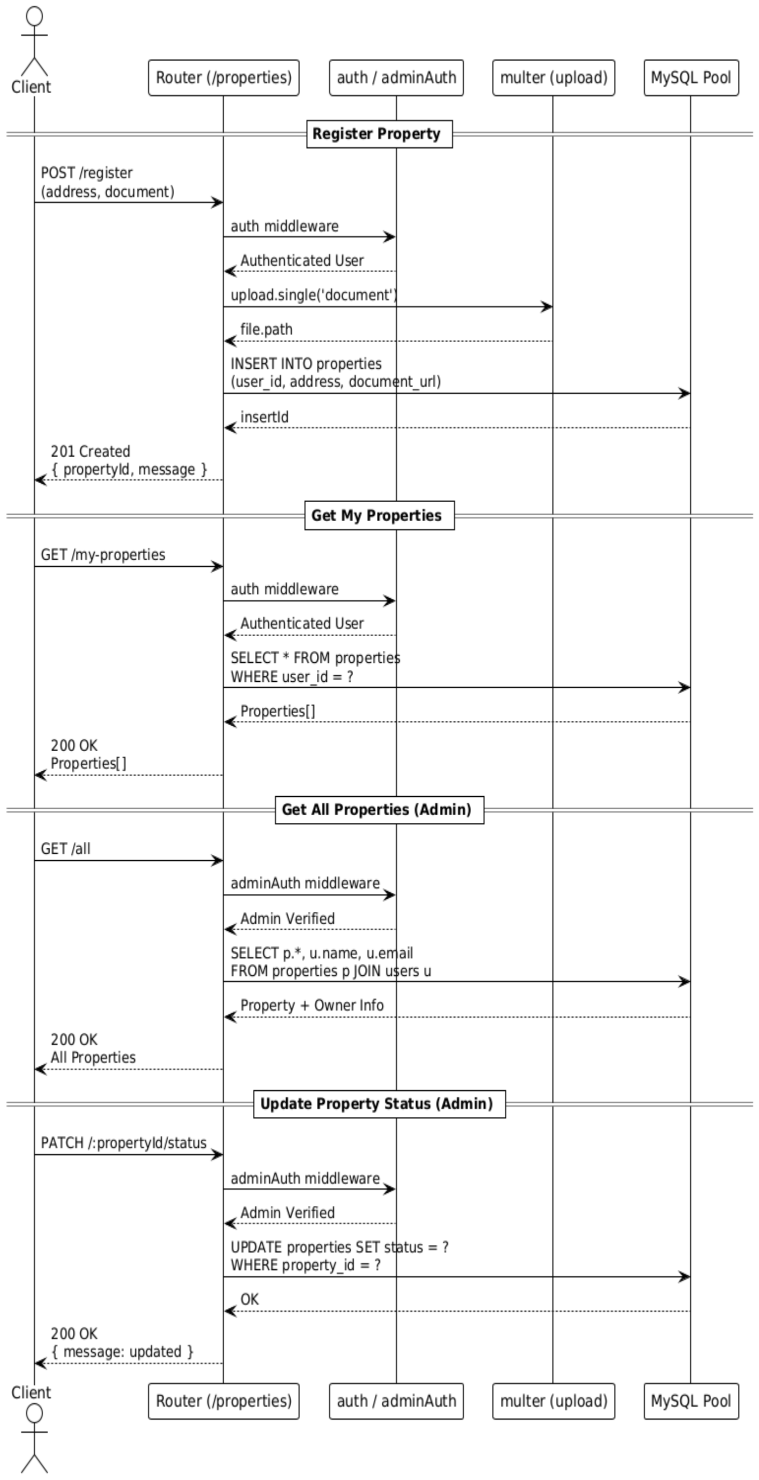

Community Management System
Participate in Digital Voting
Submit Public Feedback
Report Crime or Incident
Request Public Services
Submit Maintenance Request
Book Parking and Reserve Facilities
Register Property
Send and Receive Notifications
Manage Event RSVPs
Track and Update Payments
Manage Resident Information
Manage Bills
Manage Institutes and Healthcare
Schedule Infrastructure Maintenance
Resident
Admin

7. Sequence Diagrams

- **Three key activities visualized.**

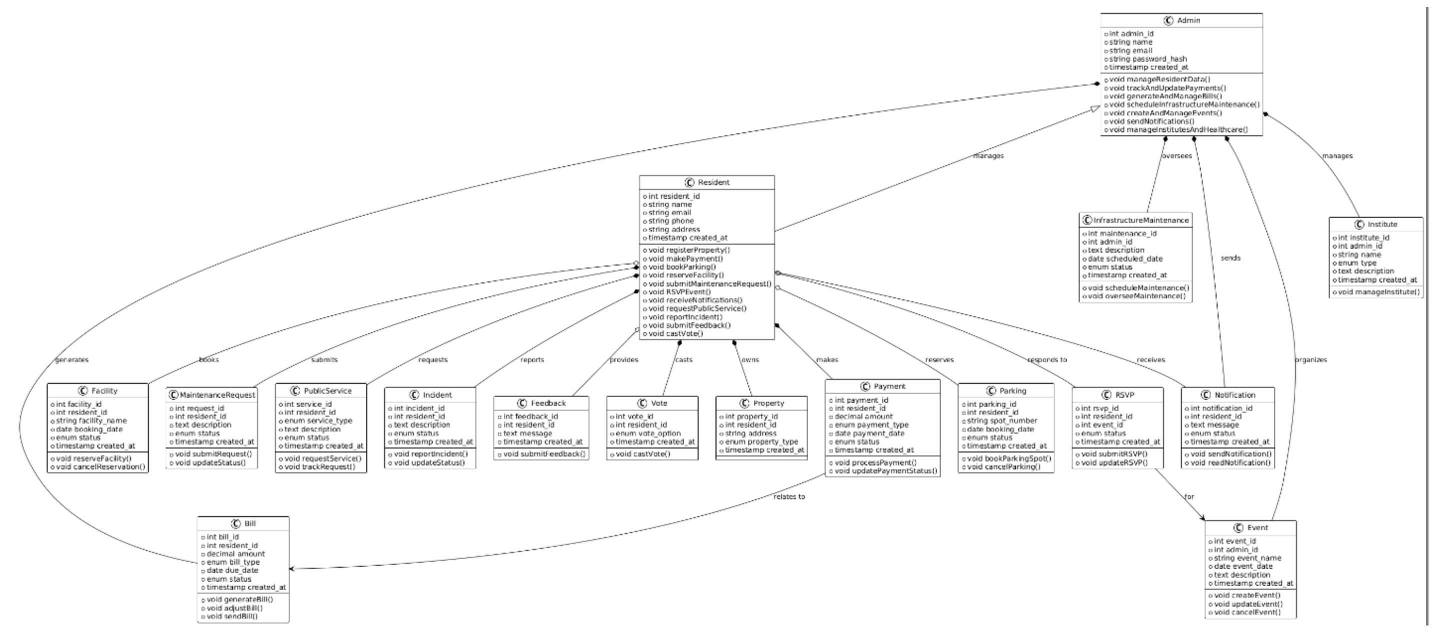

Admin
User
Router
Auth Middleware
DB Pool
Generate Bill (Admin)
POST /generate
adminAuth()
Authenticated
INSERT INTO bills
{ insertId }
INSERT INTO audit_logs
201 Created (Bill ID)
Get My Bills (User)
GET /my-bills
auth()
Authenticated
SELECT * FROM bills WHERE user_id = ?
[Bill list]
200 OK (bills)
Get All Bills (Admin)
GET /all
adminAuth()
Authenticated
SELECT with JOIN on users
[Bills with user info]
200 OK (bills)
Update Bill Status (Admin)
PATCH /:billId/status
adminAuth()
Authenticated
UPDATE bills SET status = ?
INSERT INTO audit_logs
Status updated
Create New Bill
POST /
INSERT INTO bills (user_id, desc, ...)
{ insertId }
201 Created
Get Bill by ID
GET /:id
SELECT * FROM bills WHERE id = ?
Bill or []
alt
[Bill exists]
Bill data
[Not found]
404 Not Found
Update Bill
PUT /:id
UPDATE bills SET ...
Bill updated
Delete Bill
DELETE /:id
DELETE FROM bills WHERE id = ?
Bill deleted

•

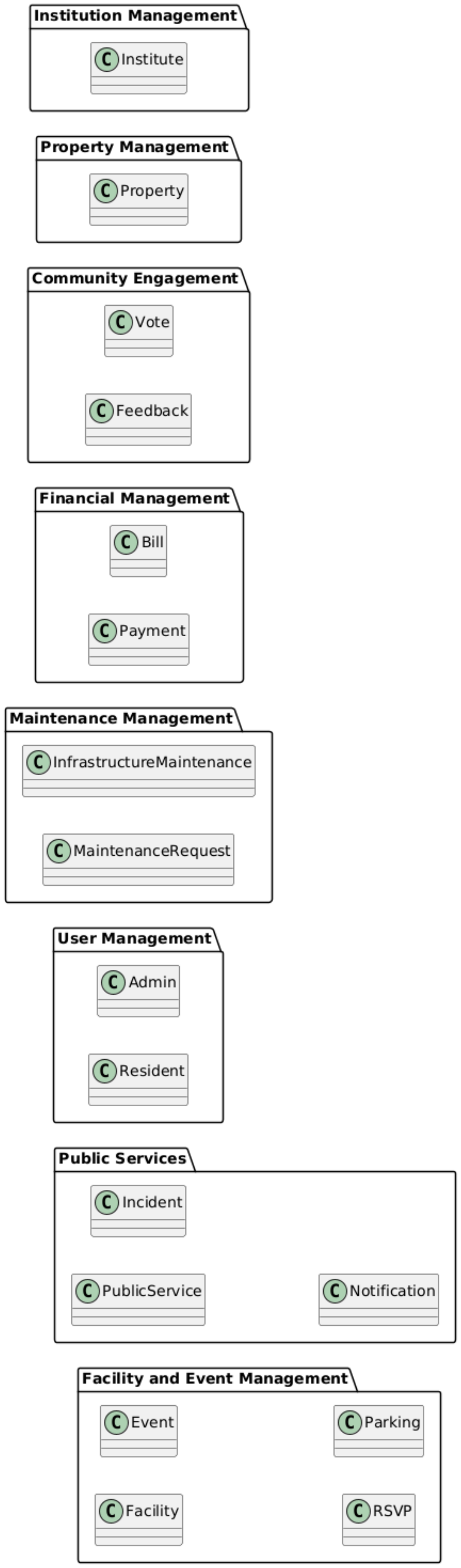

Client
Router (/properties)
auth / adminAuth
multer (upload)
MySQL Pool
Register Property
POST /register (address, document)
auth middleware
Authenticated User
upload.single('document')
file.path
INSERT INTO properties (user_id, address, document_url)
insertId
201 Created { propertyId, message }
Get My Properties
GET /my-properties
auth middleware
Authenticated User
SELECT * FROM properties WHERE user_id = ?
Properties[]
200 OK Properties[]
Get All Properties (Admin)
GET /all
adminAuth middleware
Admin Verified
SELECT p.*, u.name, u.email FROM properties p JOIN users u
Property + Owner Info
200 OK All Properties
Update Property Status (Admin)
PATCH /:propertyId/status
adminAuth middleware
Admin Verified
UPDATE properties SET status = ? WHERE property_id = ?
OK
200 OK { message: updated }

•

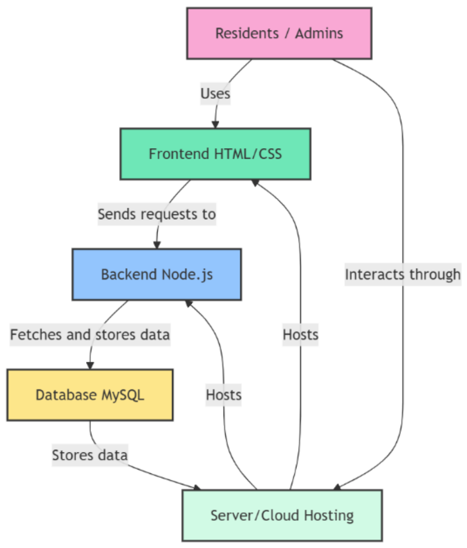
8. Class Diagram

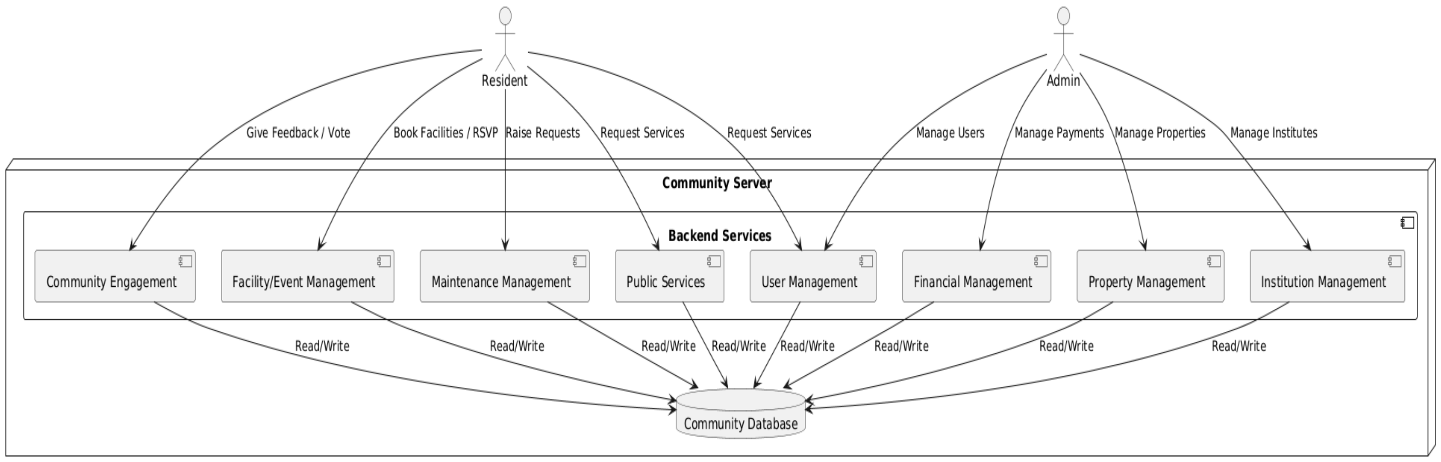


9. UML Package Diagram:

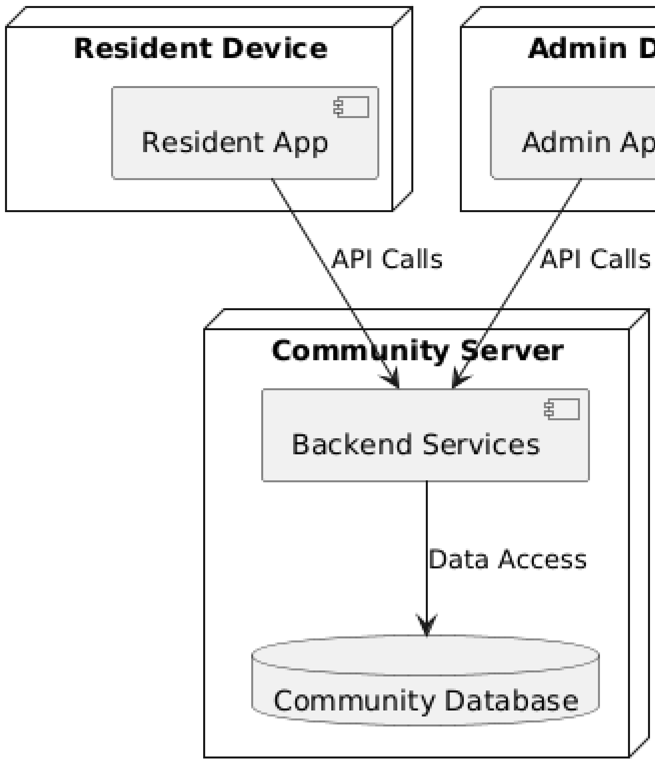

Institution Management
Institute
Property Management
Property
Community Engagement
Vote
Feedback
Financial Management
Bill
Payment
Maintenance Management
InfrastructureMaintenance
MaintenanceRequest
User Management
Admin
Resident
Public Services
Incident
PublicService
Notification
Facility and Event Management
Event
Parking
Facility
RSVP

10.Architectural Diagram (Client Server Architecture)

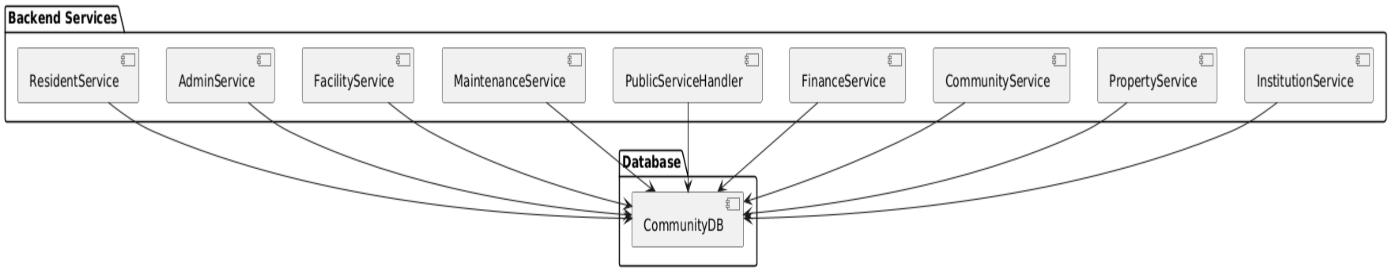




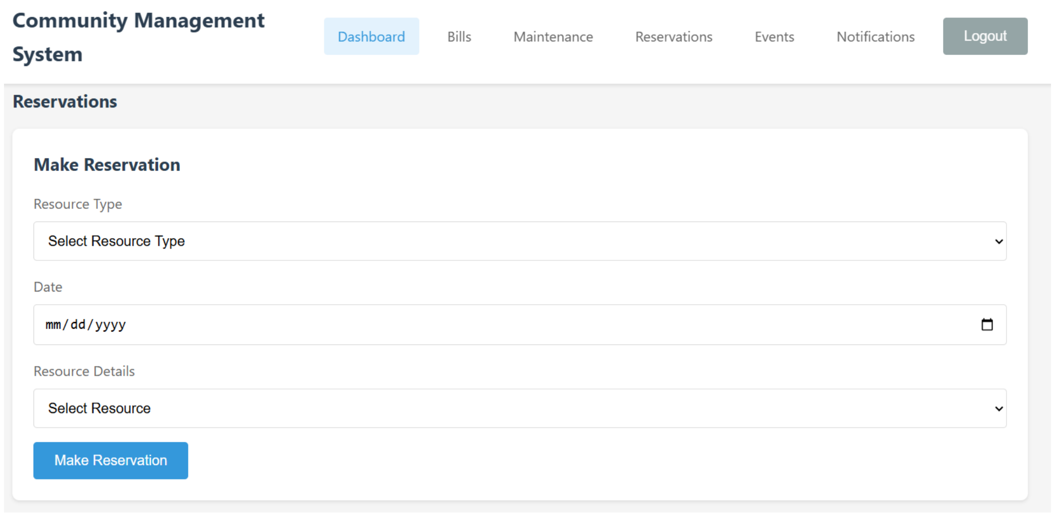
11. Deployement Diagram



12. Component Diagrams



13. Actual Implementation



Community Management System

Dashboard Bills Maintenance Reservations Events Notifications Logout

Reservations

Make Reservation

Resource Type

Select Resource Type

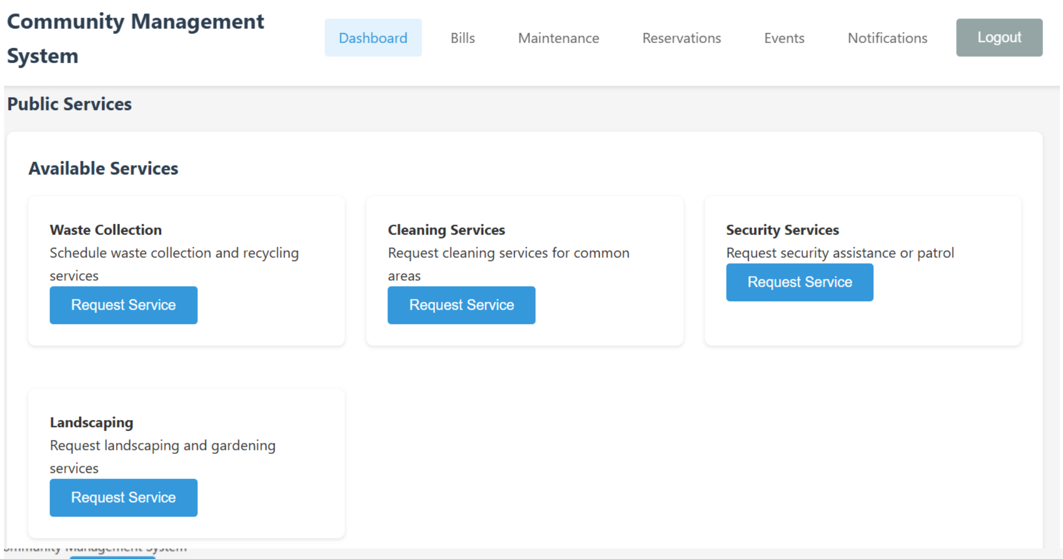
Date

mm/dd/yyyy

Resource Details

Select Resource

Make Reservation



Community Management System

Dashboard Bills Maintenance Reservations Events Notifications Logout

Public Services

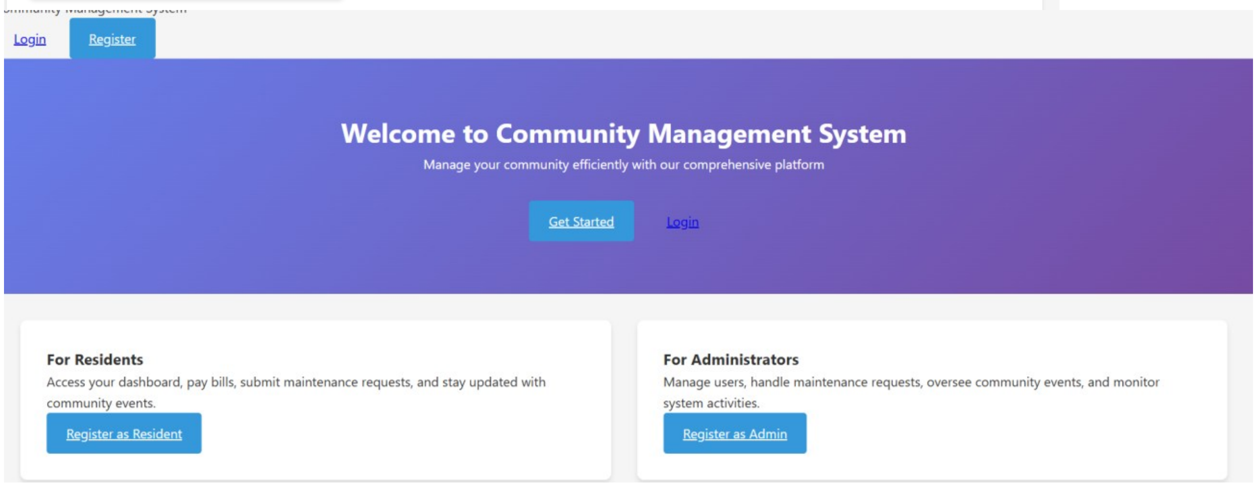
Available Services

Waste Collection
Schedule waste collection and recycling services
Request Service

Cleaning Services
Request cleaning services for common areas
Request Service

Security Services
Request security assistance or patrol
Request Service

Landscaping
Request landscaping and gardening services
Request Service



Login Register

Welcome to Community Management System

Manage your community efficiently with our comprehensive platform

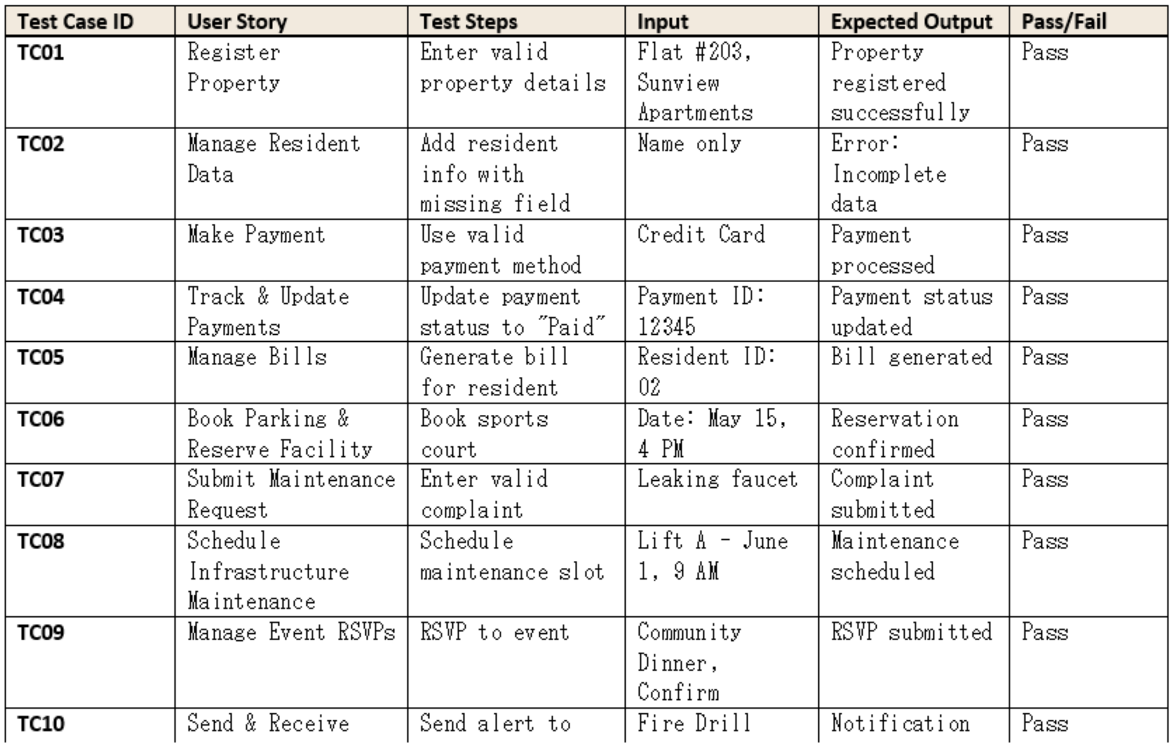
Get Started Login

For Residents
Access your dashboard, pay bills, submit maintenance requests, and stay updated with community events.
Register as Resident

For Administrators
Manage users, handle maintenance requests, oversee community events, and monitor system activities.
Register as Admin

14. BlackBox Testing

Equivalence Class Partitioning (ECP) testing:



Test Case ID	User Story	Test Steps	Input	Expected Output	Pass/Fail
TC01	Register Property	Enter valid property details	Flat #203, Sunview Apartments	Property registered successfully	Pass
TC02	Manage Resident Data	Add resident info with missing field	Name only	Error: Incomplete data	Pass
TC03	Make Payment	Use valid payment method	Credit Card	Payment processed	Pass
TC04	Track & Update Payments	Update payment status to "Paid"	Payment ID: 12345	Payment status updated	Pass
TC05	Manage Bills	Generate bill for resident	Resident ID: 02	Bill generated	Pass
TC06	Book Parking & Reserve Facility	Book sports court	Date: May 15, 4 PM	Reservation confirmed	Pass
TC07	Submit Maintenance Request	Enter valid complaint	Leaking faucet	Complaint submitted	Pass
TC08	Schedule Infrastructure Maintenance	Schedule maintenance slot	Lift A - June 1, 9 AM	Maintenance scheduled	Pass
TC09	Manage Event RSVPs	RSVP to event	Community Dinner, Confirm	RSVP submitted	Pass
TC10	Send & Receive	Send alert to	Fire Drill	Notification	Pass

15. White Box Testing

16. Work Breakdown Structure (WBS)

<u>Sofware Project Plan System (CMS:</u>

1. Project Initiation and Planning:

- **Project Kickoff**
    - **Hold a project kickoff meeting**
    - **Define project objectives and goals**
    - **Assign project team members**

• 1.2 Requirements Gathering

- 1.2.1 Conduct stakeholder interviews
- 1.2.2 Collect functional and non-functional requirements
- 1.2.3 Document requirements

• 1.3 Feasibility Study

- 1.3.1 Assess technical feasibility
- 1.3.2 Assess resource availability
- 1.3.3 Define project timeline and budget

2. Design Phase:

- 2.1 System Architecture Design
- 2.1.1 Define system architecture (client-server, database)
- 2.1.2 Select technology stack (e.g., HTML, CSS, JavaScript, Node.js,

MySql)

• 2.2 Database Design

- 2.2.1 Define database schema (tables, relationships)
- 2.2.2 Create ER diagrams
- 2.2.3 Set up database environment

• 2.3 UI/UX Design

- 2.3.1 Design wireframes for key pages (e.g., login, dashboard,

- incident reporting)
- 2.3.2 Create mockups for UI design

3. Development Phase:

• 3.1 Backend Development:

- 3.1.1 Set up backend server and APIs
- 3.1.2 Implement user authentication and authorization
- 3.1.3 Implement database integration
- 3.1.4 Develop core business logic (e.g., property management,

manage requests)

• 3.2 Frontend Development:

- 3.2.1 Implement frontend UI components
- 3.2.2 Integrate with backend APIs
- 3.2.3 Develop responsive design

• 3.3 Testing:

- 3.3.1 Unit testing for backend
- 3.3.2 Unit testing for frontend
- 3.3.3 Integration testing
- 3.3.4 User acceptance testing (UAT)

• 3.4 Deployment Setup:

- 3.4.1 Set up staging and production environments
- 3.4.2 Deploy to the server
- 3.4.3 Perform initial deployment testing

4. Testing Phase

• 4.1 Functional Testing

- 4.1.1 Test all features (e.g: property management, manage requests,
- etc.)

- 4.1.2 Verify user access control

• 4.2 Performance Testing

- 4.2.1 Test system performance under load
- 4.2.2 Optimize for scalability

• 4.3 Security Testing

- 4.3.1 Check for vulnerabilities (e.g., SQL injection, XSS)
- 4.3.2 Ensure data protection and compliance

5. Deployment and Maintenance

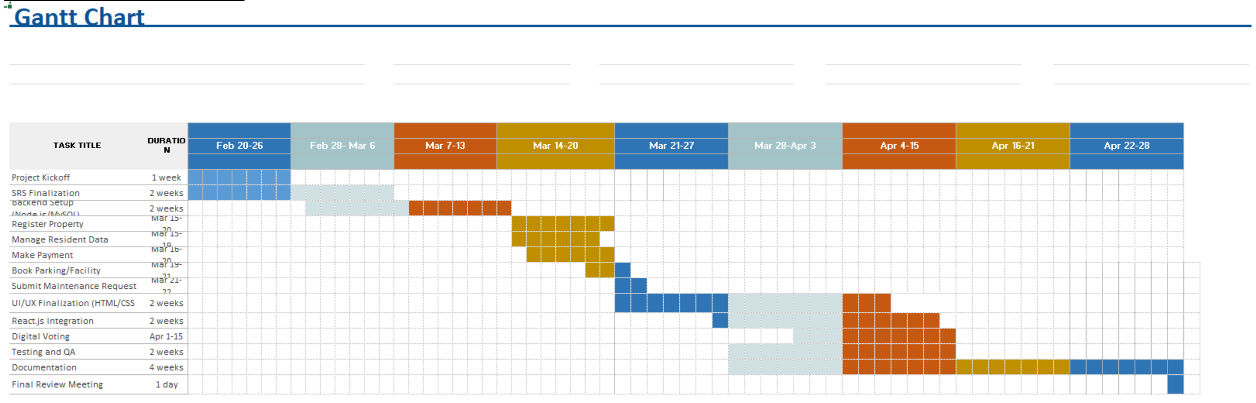
• 5.1 Final Deployment

- 5.2.1 Deploy the CMS to production
- 5.2.2 Perform final checks

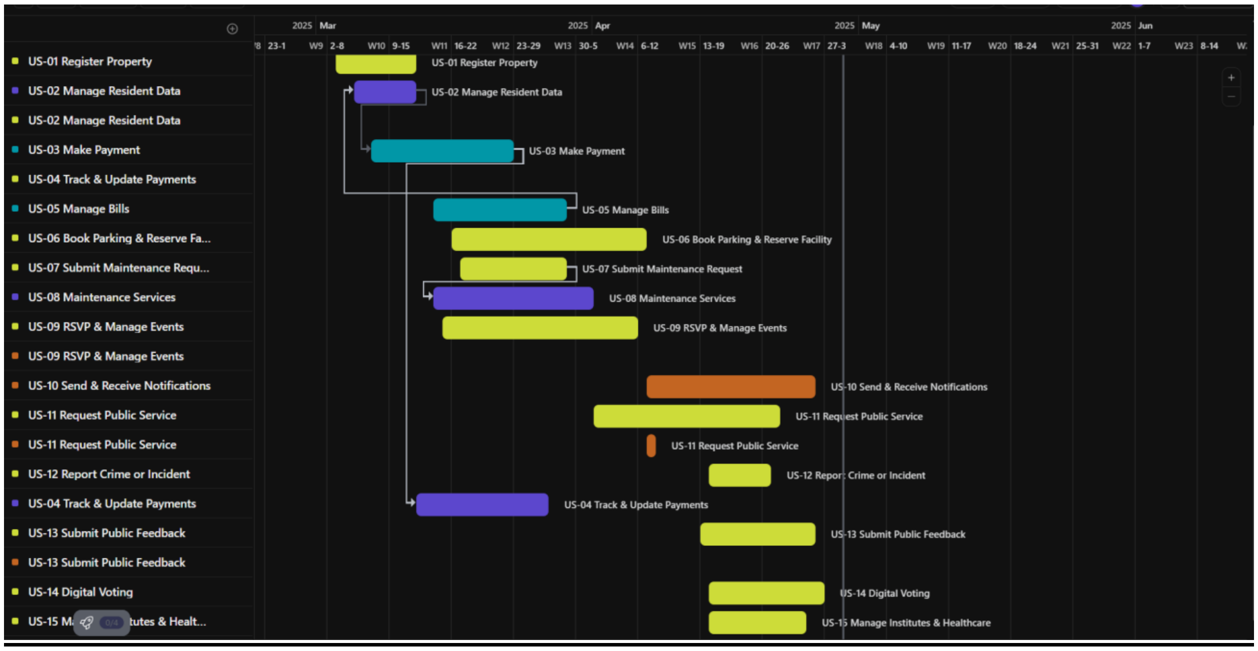
• 5.2 Ongoing Maintenance and Support

- 5.3.1 Provide system updates and patches
- 5.3.2 Offer technical support and bug fixes
- 5.3.3 Monitor system performance

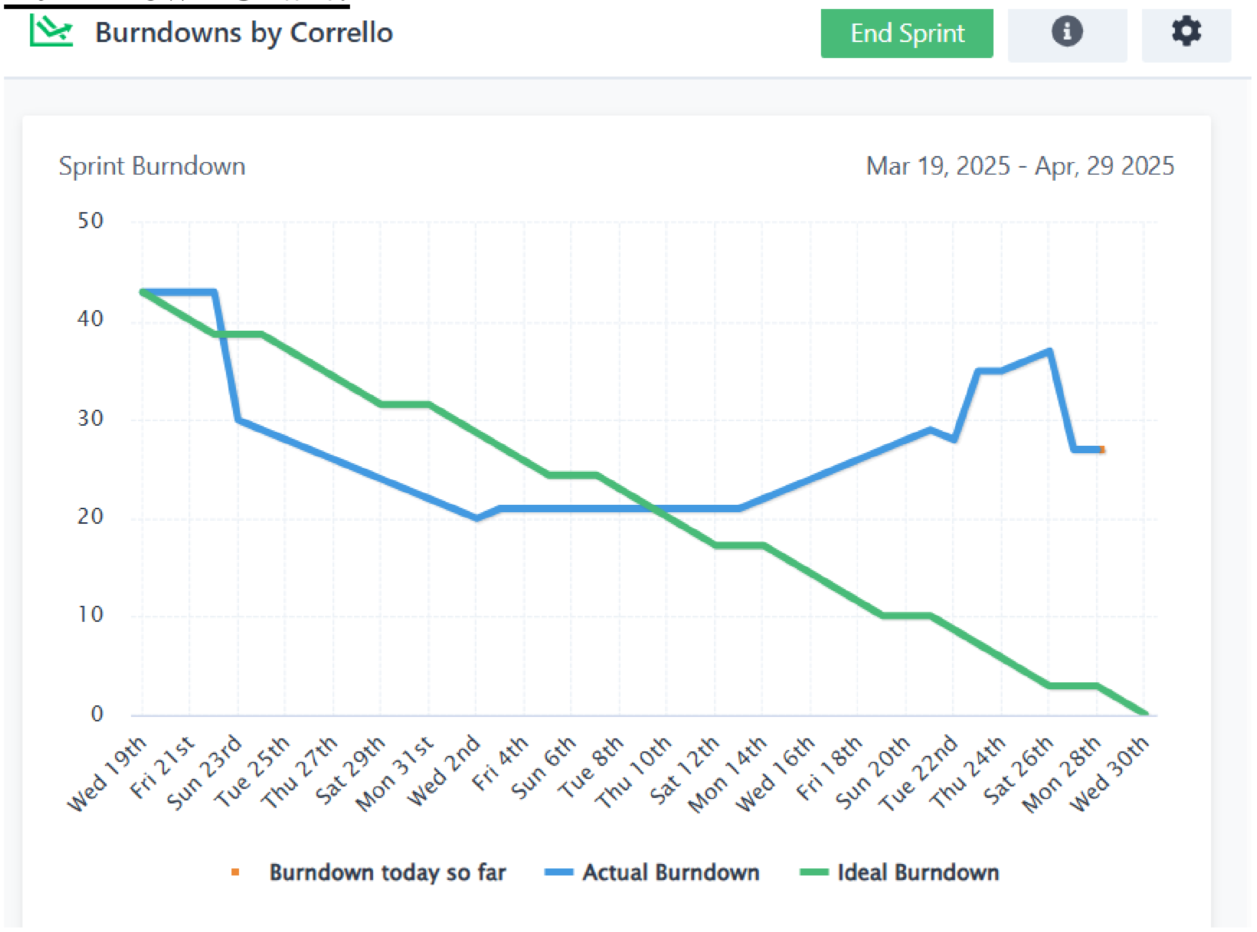
<u>Gantt Chart:</u>



TASK TITLE	DURATION
Project Kickoff	1 week
SRS Finalization	2 weeks
Backend Setup (Node.js/MySQL)	2 weeks
Register Property	Mar 15-20
Manage Resident Data	Mar 15-19
Make Payment	Mar 16-20
Book Parking/Facility	Mar 19-21
Submit Maintenance Request	Mar 21-22
UI/UX Finalization (HTML/CSS	2 weeks
React.js Integration	2 weeks
Digital Voting	Apr 1-15
Testing and QA	2 weeks
Documentation	4 weeks
Final Review Meeting	1 day



Burn Down Chart:



17. Product Backlog

The product backlog includes all user stories prioritized based on their importance.

User Story	**Priority**
Register Property	**High**
Manage Resident Data	**High**
Make Payment	**High**
Track & Update Payments	**High**
Manage Bills	**High**
Book Parking & Reserve Facility	**Medium**
Submit Maintenance Request	**Medium**
Schedule Infrastructure Maintenance	**Medium**
RSVP & Manage Events	**Low**
Send & Receive Notifications	**Low**
Request Public Service	**Low**
Report Crime or Incident	**Low**
Submit Public Feedback	**Low**
Digital Voting	**Low**
Manage Institutes & Healthcare	**Low**

18. Sprint Backlog

Sprint 1 Backlog

User Story	**Assigned To**	**Tasks**	**Milestones**
Register Property	Haziq	Database schema, API, UI	March 16: Backend, March 18: UI, March 20: Testing
Manage Resident Data	Muneeb	Schema, API, UI	March 15: API, March 17: UI, March 19: Testing
Make Payment	Ahmed	Payment API, UI,	March 16: Backend,

		Security	March 18: UI March 20: Security
Track & Update Payments	Haziq	Database, Admin panel, Notifications	March 17: Backend March 19: Testing
Manage Bills	Muneeb	Billing system, API, UI	March 15: Backend March 18: UI March 20: Testing

Sprint 2 Backlog (Subset of User Stories)

User Story	**Assigned To**	**Tasks**	**Milestones**
Book Parking & Reserve Facility	Ahmed	Reservation System, UI, API	March 19: Backend March 200: UI March 21: Testing
Submit Maintenance Request	Haziq	Request submission, File uploads, Status tracking	March 21: Backend, March 21: UI, March 21: Testing
Schedule Infrastructure Maintenance	Muneeb	Admin scheduling UI, Notifications	March 22: Backend, March 22: UI March 22: Testing

19. Meeting Minutes

Project: Communi3y – Community Management System (CMS)
Meeting Title: Functional and Technical Requirements Review
Date: 15 MARCH 2025
Time: 8:30 pm
Location: Virtual
Chairperson: Muneeb ul Islam, Product Owner
Recorder: Ahmed Mustafa, Scrum Master
Attendees:

1) Ahmed (Developer, Scrum Master)
2) Muneeb (Developer, Product Owner)
3) Haziq (Developer, Scrum Team)

a. Agenda

1. Review of SRS Document

2. Clarification of Roles & Responsibilities
3. Discussion on Sprint 1 and Sprint 2 Backlog
4. Confirmation of Design Artifacts (Use Case, Sequence, Class Diagrams)
5. Roadmap Review and Action Items

---

b. Meeting Notes:

- Reviewed the document and project purpose: A web-based CMS designed to facilitate digital community management (property registration, service requests, incident reporting, etc.).
- Reiterated the use of IEEE SRS format and audience scope (Developers, Admins, Residents).
- Confirmed product scope: Includes digital voting, payment, reservations, and feedback systems.
- Technologies finalized:
  - **Frontend**: HTML, CSS, React.js (planned post-Sprint 3)
  - **Backend**: Node.js
  - **Database**: MySQL
- Two main user classes: **Residents** and **Admins**
- System will be hosted locally but must support cloud scalability.

C. Sprint Planning

- **Sprint 1 Stories Reviewed:**
  - *Register Property*, *Manage Resident Data*, *Make Payment*, *Track Payments*, *Manage Bills*
  - Tasks assigned with clear milestones. Developers confirmed task deadlines.
- **Sprint 2 Stories Reviewed:**
  - *Book Parking*, *Submit Maintenance Request*, *Schedule Infrastructure Maintenance*
  - Emphasis placed on testing and UI readiness by March 22.

d. Non-Functional & Compliance Requirements

- Agreed on:
  - System must support **1000+ concurrent users**
  - Real-time transaction capability

---

e. Decisions Made

- React.js integration will be postponed until end of Sprint 3, meanwhile use HTML and CSS for testing the frontend purposes.
- Documentation (User Manual & API Docs) to be created alongside system features.
- Working on Backend using Node.js and MySQL for database.
- Digital voting and feedback systems and other user stories marked for Sprint 3 due to lower priority.

---

4) Action Items

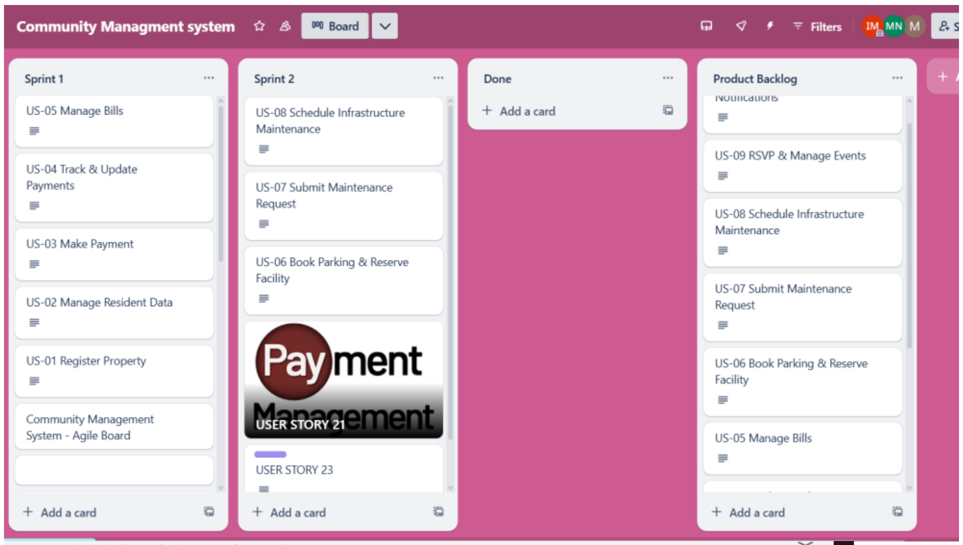
Action Item	Responsible	Deadline
Update Class and Sequence Diagrams	Haziq, Ahmed	March 15
Begin Sprint 2 Development	All Devs	March 19
Finalize UI/UX designs	Ahmed	March 22
Backend and DB setup	Muneeb. Haziq	March 16

Next Meeting: 15 April 2025
Objective: Review Sprint 3 progress and initial testing feedback.

---

20. Version Control & Contribution Evidence

- https://github.com/Mustafaahmed10/SE_Project
- https://trello.com/b/CmcUFss2/community-managment-system
- ScreenSHots:



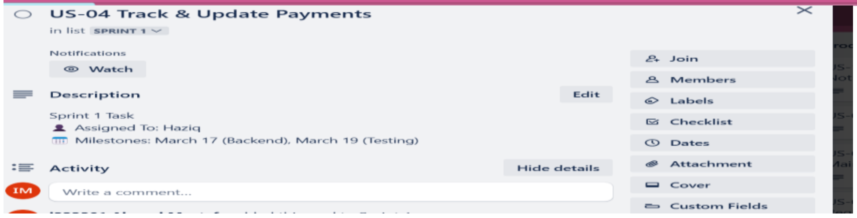
Community Managment system

Board

Filters

IM MN M

Sprint 1	Sprint 2	Done	Product Backlog
US-05 Manage Bills	US-08 Schedule Infrastructure Maintenance	+ Add a card	Notifications
US-04 Track & Update Payments	US-07 Submit Maintenance Request		US-09 RSVP & Manage Events
US-03 Make Payment	US-06 Book Parking & Reserve Facility		US-08 Schedule Infrastructure Maintenance
US-02 Manage Resident Data	Payment Management USER STORY 21		US-07 Submit Maintenance Request
US-01 Register Property	USER STORY 23		US-06 Book Parking & Reserve Facility
Community Management System - Agile Board			US-05 Manage Bills
+ Add a card	+ Add a card		+ Add a card

- 

US-04 Track & Update Payments

in list SPRINT 1

Notifications

Watch

Description

Edit

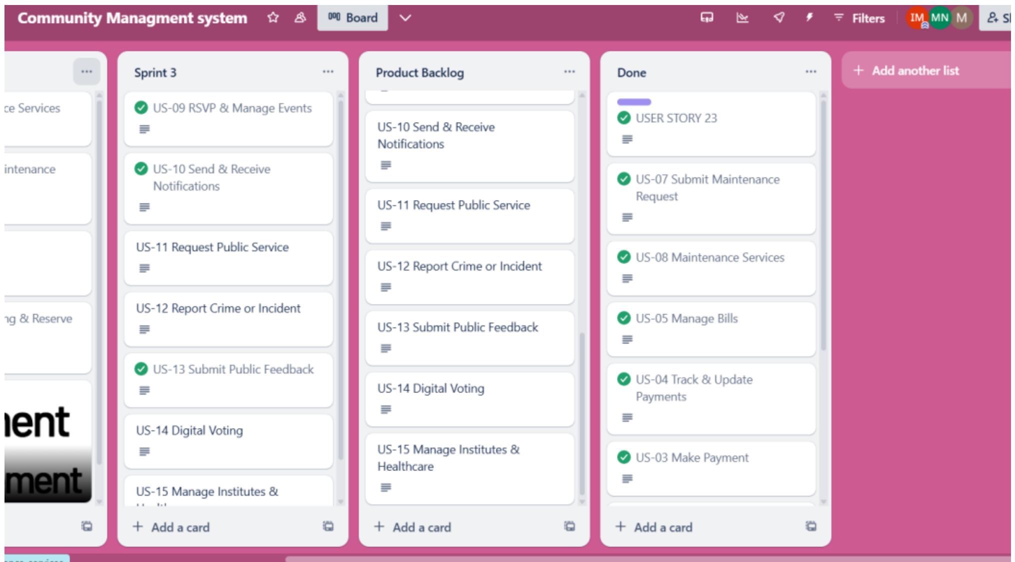
Sprint 1 Task
Assigned To: Haziq
Milestones: March 17 (Backend), March 19 (Testing)

Activity

Hide details

IM Write a comment…

Join
Members
Labels
Checklist
Dates
Attachment
Cover
Custom Fields



Community Managment system

Board

Filters

ce Services

intenance

ng & Reserve

Sprint 3

- US-09 RSVP & Manage Events
- US-10 Send & Receive Notifications
- US-11 Request Public Service
- US-12 Report Crime or Incident
- US-13 Submit Public Feedback
- US-14 Digital Voting
- US-15 Manage Institutes &

Add a card

Product Backlog

- US-10 Send & Receive Notifications
- US-11 Request Public Service
- US-12 Report Crime or Incident
- US-13 Submit Public Feedback
- US-14 Digital Voting
- US-15 Manage Institutes & Healthcare

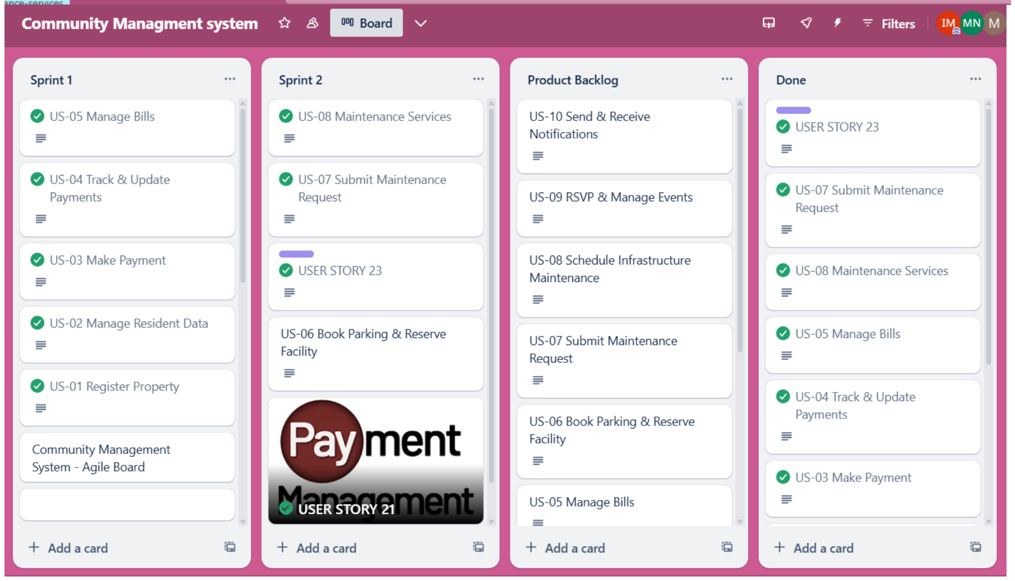
Add a card

Done

- USER STORY 23
- US-07 Submit Maintenance Request
- US-08 Maintenance Services
- US-05 Manage Bills
- US-04 Track & Update Payments
- US-03 Make Payment

Add a card

Add another list



Community Managment system

Board

Filters

Sprint 1

- US-05 Manage Bills
- US-04 Track & Update Payments
- US-03 Make Payment
- US-02 Manage Resident Data
- US-01 Register Property
- Community Management System - Agile Board

Add a card

Sprint 2

- US-08 Maintenance Services
- US-07 Submit Maintenance Request
- USER STORY 23
- US-06 Book Parking & Reserve Facility
- USER STORY 21

Add a card

Product Backlog

- US-10 Send & Receive Notifications
- US-09 RSVP & Manage Events
- US-08 Schedule Infrastructure Maintenance
- US-07 Submit Maintenance Request
- US-06 Book Parking & Reserve Facility
- US-05 Manage Bills

Add a card

Done

- USER STORY 23
- US-07 Submit Maintenance Request
- US-08 Maintenance Services
- US-05 Manage Bills
- US-04 Track & Update Payments
- US-03 Make Payment

Add a card

21. Lesson Learned By Group:

Creating the **Community Management System (Communi3y)** by Applying Software Engineering Principles has been a comprehensive learning experience for us as a team. Here's what **we learned** throughout this journey:

1. **Understanding Real-World Problem Solving**
   We realized how important it is to deeply understand the real-life challenges of community management before jumping into development. We gathered info from potential users and stakeholders, which helped us design features that genuinely addressed their needs.

2. **Importance of Clear Requirements Gathering**
   Early on, we learned that vague or incomplete requirements lead to rework. Following the IEEE 830 SRS standard helped us structure our documentation and gather clearer, more precise functional and nonfunctional requirements.

3. **Collaborative Software Design**
   Through use case diagrams, class diagrams, and sequence diagrams, we learned how crucial UML is for visualizing system architecture before writing a single line of code. This step saved us time during implementation and helped align all team members on the design.

4. **Agile Project Management**
   Working in sprints taught us the value of breaking work into manageable parts. Sprint backlogs, product backlog, and our Trello board helped us stay organized. Daily standups and weekly meetings enhanced our accountability and collaboration.

5. **Testing is Not Optional**
   We underestimated the role of BlackBox and WhiteBox testing initially. But through practice, we learned how testing ensures system reliability and helps catch bugs early. This significantly improved the quality of our final product.

6. **Version Control & Collaboration Tools Matter**
   GitHub wasn't just a tool—it became our backbone for managing versions, handling conflicts, and tracking contributions. We learned how vital clean commits and pull requests are to collaborative coding.

7. **Communication is Key**
   Whether during documentation or coding sprints, continuous communication helped us avoid misunderstandings and improve feature integration across different modules.

8. **Responsibility and Ownership**
   Each of us learned to take full ownership of our modules, while also supporting each other when needed. It was a valuable lesson in teamwork and responsibility.