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 **Community Management System (CMS)** is a web-based application designed to streamline property management, public service requests, digital voting, 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 **Next.js (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 **Next.js**.
- 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
- ### Book Parking & Reserve Facility

User Story: As a resident, I want to book parking spots and recreational facilities so that I can use them when needed.

- **Acceptance Criteria:**
    - Display available parking spots and facilities.
    - Allow residents to book and cancel reservations.
    - Prevent overbooking by limiting reservations per slot.
    - Send confirmation upon successful booking.
- ### Submit Maintenance Request

User Story: As a resident, I want to submit a maintenance request so that my property issues can be addressed.

- **Acceptance Criteria:**
    - Enter request description and category.
    - Upload images or documents for proof.
    - Track request status ("Pending," "In Progress," "Completed").
    - Receive notifications about status updates.
- ### Schedule Infrastructure Maintenance

User Story: As an admin, I want to schedule and oversee infrastructure maintenance so that public facilities remain operational.

- **Acceptance Criteria:**

  - Schedule maintenance tasks.
  - Notify residents about upcoming maintenance.
  - Track maintenance progress and completion.
  - Log past maintenance records for reference.

- ## RSVP & Manage Events

User Story (Resident): As a resident, I want to RSVP for events so that I can attend community activities.

- **Acceptance Criteria:**
  - Display a list of upcoming events.
  - Allow residents to RSVP and update their responses.
  - Prevent exceeding event capacity.
  - Send confirmation upon RSVP submission.

User Story (Admin): As an admin, I want to create and manage community events so that residents can participate.

- **Acceptance Criteria:**
  - Allow admins to create, edit, and delete events.
  - Display event details (date, time, location, description).
  - Track attendance and RSVPs.

- ## Send & Receive Notifications

User Story (Admin): As an admin, I want to send notifications to residents so that they stay informed about important updates.

- **Acceptance Criteria:**
  - Compose and send messages.
  - Deliver notifications via email, SMS, or the dashboard.
  - Categorize notifications (e.g., urgent, informational).

User Story (Resident): As a resident, I want to receive notifications so that I stay updated on society matters.

- **Acceptance Criteria:**
  - Display notifications on the dashboard.

- Allow marking notifications as read.
- Store notification history for reference.

---

5. Nonfunctional Requirements

5.1 Product Requirements

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

5.2 Organizational Requirements

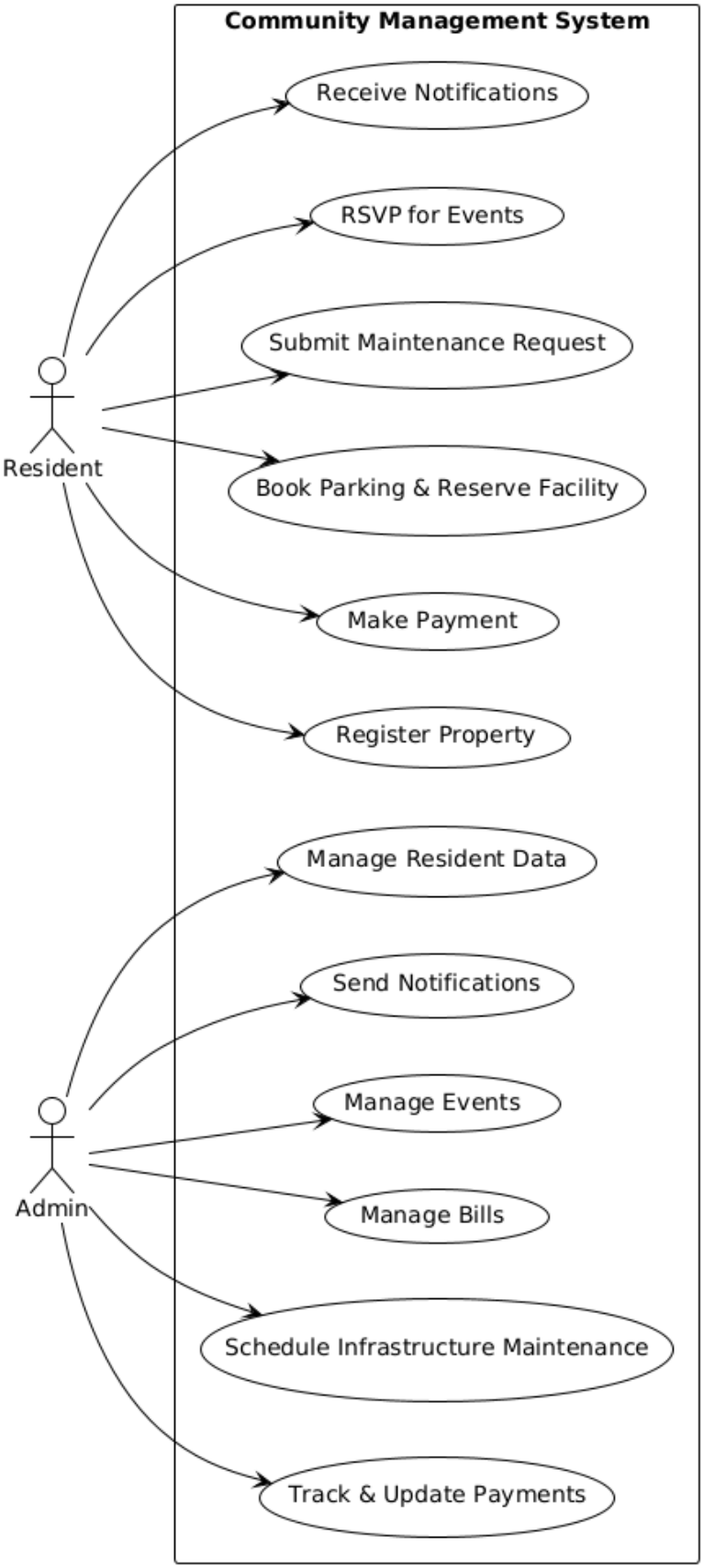
- Uses **Agile methodology** for iterative development.

5.3 External Requirements

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

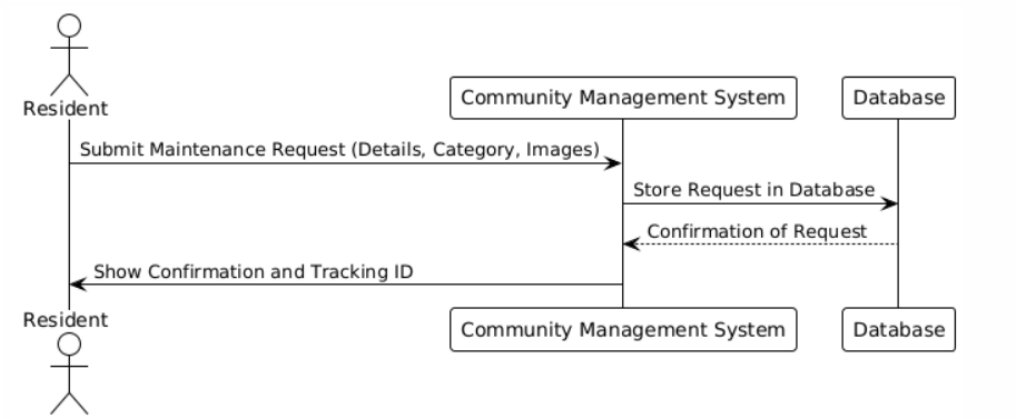
---

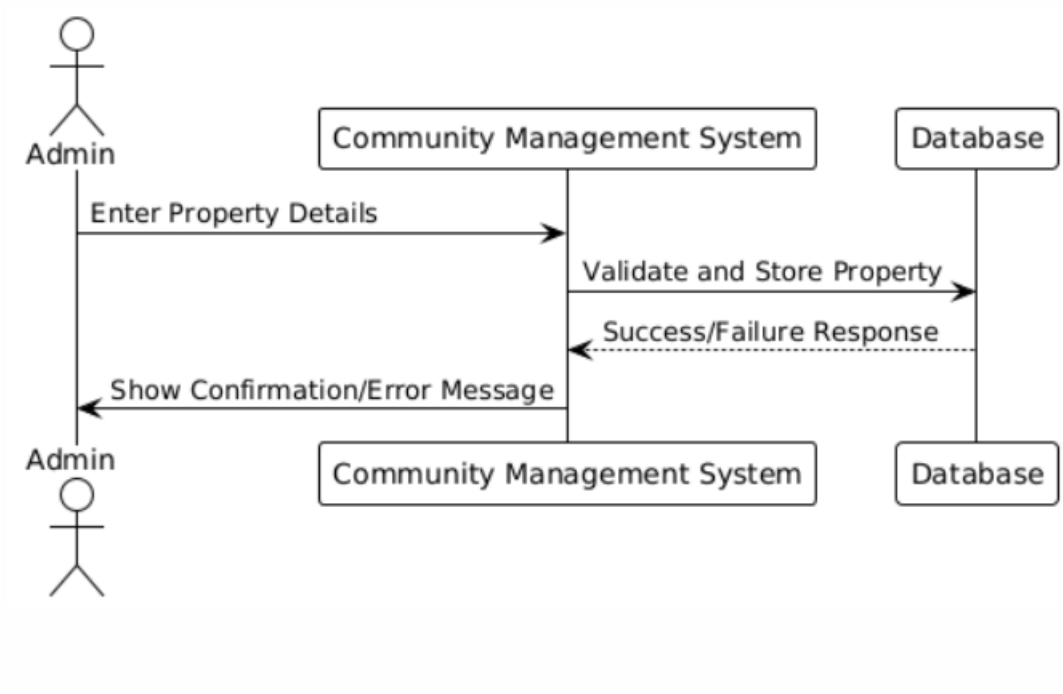
6. Use Case Diagram & User Stories

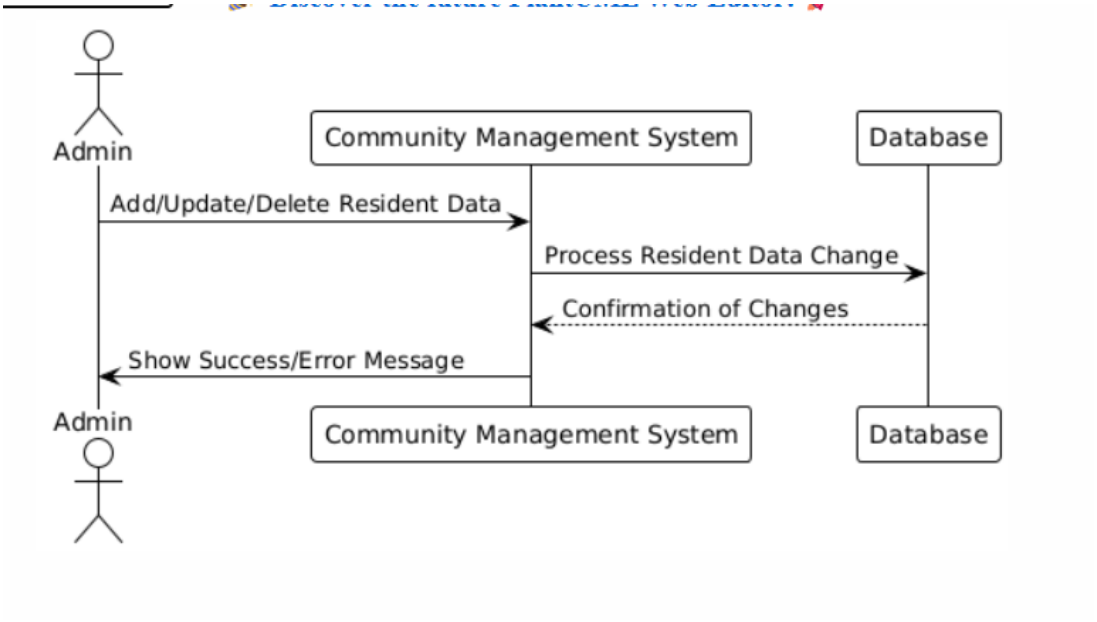

Community Management System
Receive Notifications
RSVP for Events
Submit Maintenance Request
Book Parking & Reserve Facility
Make Payment
Register Property
Manage Resident Data
Send Notifications
Manage Events
Manage Bills
Schedule Infrastructure Maintenance
Track & Update Payments
Resident
Admin

7. Sequence Diagrams

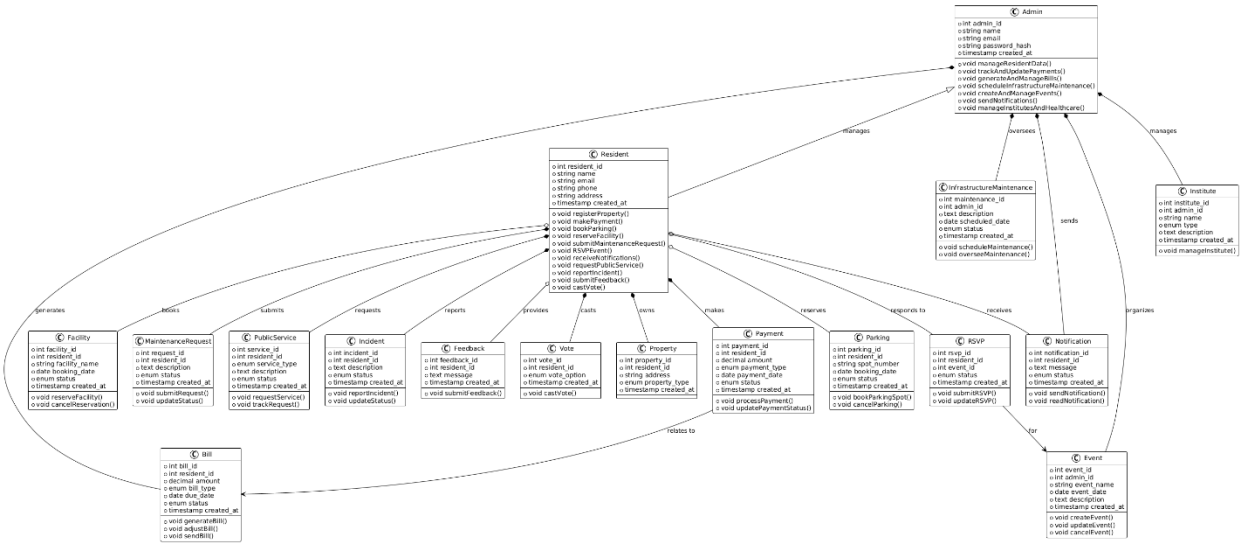
- **Three key activities visualized.**



- 



8. Class Diagram



9. 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**

10. 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, Security	March 16: Backend, 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

-

11. Version Control & Contribution Evidence

- **GitHub Link & Contribution Screenshots**
- https://github.com/Haziq739/SE-Assignment-01
-