

Chapter 1: Introduction

1.1 Project Overview

In today's dynamic environment, managing societies—whether they are residential communities, professional organizations, or other types of groups—requires a sophisticated approach to ensure efficiency and transparency. Traditional methods of management, often reliant on manual processes and disparate tools, can lead to inefficiencies and challenges in communication and coordination.

To address these issues, this document proposes the development of a comprehensive Society Management System (SMS) software.

This software solution aims to streamline the administrative tasks of societies, enhancing the management of memberships, financial transactions, event planning, and communication within the community.

The Society Management System will serve as an integrated platform designed to simplify and automate routine operations, offering a central hub for all management activities. By leveraging modern technology, the system will provide features such as automated billing, real-time communication channels, event management, and detailed reporting capabilities. This document outlines the detailed requirements and specifications for the Society Management System software, including functional and non-functional requirements, system constraints, and user interactions. The goal is to ensure that the developed system meets the needs of its users effectively, providing a scalable and user-friendly solution for managing societal functions.

1.2 Purpose

The purpose of a Society Management System (SMS) is to streamline and automate the management of residential communities, enhancing the overall efficiency and effectiveness of operations for administrators, residents, and facility staff. This system serves as a centralized platform that addresses various administrative tasks, thereby reducing the manual workload and improving communication among stakeholders.

1.2.1 Problem statement

Managing residential societies involves numerous administrative, financial, and operational challenges, often leading to inefficiencies, miscommunication, and dissatisfaction among residents. Current management practices are largely manual, disorganized, and lack the technological infrastructure needed for streamlined operations. This results in time-consuming tasks such as maintenance tracking, utility billing, resident communication, and record-keeping, often prone to errors and delays.

1.2.2 Project aim

A Society Management System that integrates digital tools and automation can address these issues by providing a centralized platform for managing all aspects of society operations, improving communication, and enhancing resident satisfaction.

The Society Management System will serve as an integrated platform designed to simplify and automate routine operations, offering a central hub for all management activities.

1.2.3 Project objectives

- Manual management of society operations is time-consuming, inefficient, and prone to errors.
- Lack of a centralized system leads to miscommunication and conflicts among residents and management.
- Tracking visitors, deliveries, and staff movements manually poses security risks.
- Residents have limited access to real-time updates about society events and activities.
- Difficulty in storing and retrieving important records and transactions in a traditional setup.
- Managing complaints, vendors, and services manually leads to delays and inefficiencies.
- Absence of automated processes results in unnecessary consumption of time, money, and energy

Chapter 2: Project Scope

The scope of a Society Management System (SMS) includes various functionalities designed to enhance the management and operational efficiency of residential communities. Below are the key points outlining the scope:

- **Administrative Automation:** Streamlines routine tasks such as billing, complaint registration, and event management, reducing manual workloads for administrators.
- **Resident Engagement:** Provides a platform for residents to interact, share information, and participate in community governance through features like online voting and event proposals.
- **Complaint Handling:** Facilitates easy submission and tracking of complaints or suggestions by residents, ensuring timely responses from the administration.
- **Event Management:** Offers tools for planning, scheduling, and managing community events, enhancing social interaction among residents.
- **Visitor Management:** Enhances security through a systematic approach to managing visitor check-ins and tracking access to the premises.
- **Accessibility:** Ensures that the system is accessible from multiple devices, allowing users to engage with the platform conveniently.
- **Sustainability Practices:** Encourages eco-friendly initiatives by minimizing paper usage through digital documentation and communication.

2.1 Impact

- **Operational Efficiency:** By automating routine administrative tasks such as billing, complaint tracking, and event management, the SMS reduces the workload on committee members and staff. This allows them to focus on strategic initiatives rather than mundane tasks.
- **Enhanced Communication:** The SMS provides a centralized platform for communication among residents and administrators, ensuring that important announcements and updates are disseminated effectively. This improves overall community connectivity.
- **Data-Driven Decision Making:** The system collects and analyzes data related to community operations, enabling administrators to make informed decisions about resource allocation and community needs. This leads to better management practices.

2.1.1 Significance

- **Transparency and Accountability:** The SMS enhances transparency in financial transactions by allowing residents to access billing information and payment histories. This fosters trust between residents and the management committee.
- **Community Engagement:** By facilitating online voting for committee elections and allowing residents to propose events or initiatives, the SMS encourages active participation in community governance, strengthening the sense of belonging among residents.
- **Improved Quality of Life:** With automated processes, residents can enjoy a more organized living environment. The time saved from administrative tasks can be redirected towards community- building activities that enrich residents' lives.

2.1.2 Contribution

- **Resource Optimization:** The SMS helps in efficiently allocating resources based on community needs, minimizing waste and ensuring that facilities are utilized effectively.
- **Sustainability Initiatives:** By promoting eco-friendly practices through digital documentation and communication, the SMS contributes to sustainability efforts within the community.
- **Security Enhancements:** Features like visitor management and secure user authentication improve the safety of residents, contributing to a more secure living environment.

2.2 Summary

In summary, the Society Management System plays a crucial role in modernizing the management of residential societies. Its impact on operational efficiency, communication, and resident engagement is significant, making it an essential tool for fostering a well-organized and harmonious community. Through enhanced transparency, accountability, and resource optimization, the SMS not only improves the quality of life for residents but also supports sustainable practices within the community.

CHAPTER 3: FEASIBILITY ANALYSIS

3.1 Technical Feasibility

- **Technology Stack:** The project utilizes widely available and proven technologies, including Java, JavaScript, HTML, MySQL, and CSS, ensuring compatibility and support.
- **Infrastructure:** The hardware and software requirements (e.g., 8GB RAM, 256GB SSD, and SQL databases) are standard and accessible.
- **Development Tools:** Tools like Visual Studio Code, AWS for deployment, and GitHub for version control support efficient development and collaboration.

3.2 Time Schedule Feasibility

- **Defined Timeline:** The project plan breaks the development process into 12 weeks, including analysis, design, development, testing, deployment, and training.
- **Phased Development:** Clear segmentation (e.g., backend development in Weeks 7–8, testing in Week 9) indicates a manageable timeline.

3.3 Operational Feasibility

- **User-Centric Design:** The system aims to simplify operations for society managers and residents with features like real-time alerts, visitor management, vendor and services.
- **Accessibility:** Mobile and web-based platforms ensure that users can interact with the system conveniently.
- **Engagement & Support:** Features such as Helpdesk, directories, and emergency contact integration cater to operational needs, promoting adoption among users.

3.4 Implementation Feasibility

- **Modular Design:** A modular codebase enables easier updates and scalability for future enhancements.
- **Training:** Deployment includes preparing user manuals, ensuring that users understand and effectively adopt the system.

3.5 Economic Feasibility

- **Cost Efficiency:** The use of open-source tools and cloud hosting (AWS) minimizes upfront and operational costs.
- **Scalability:** The system can handle up to 10,000 concurrent users, making it a cost-effective long-term solution for various communities.
- **Savings:** Automation reduces manual administrative tasks, saving time and money for society management.

Chapter 4: Hardware and Software Requirements

4.1 Hardware Requirements:

Development Machines:

- **RAM:** Minimum 8GB
- **Storage:** 256GB SSD
- **Processor:** Intel i3 or equivalent.
- **Devices:** Android and iOS smartphones and Laptop.

4.2 Software Requirements:

The software requirements for a Society Management System typically include the following:

Operating System

- **Windows:** Windows 7 or higher is commonly recommended for desktop applications.

Development Environment

- **IDE:** Visual Studio for web application.
- **Languages Used:** CSS, HTML, NodeJS and JavaScript.

Database

- **SQL Database:** Microsoft SQL Server is often used to store member details and transaction records.
- **Web Server:** IIS (Internet Information Services) for hosting web applications.
- **Cloud Services:** For cloud-based systems, services like AWS may be utilized to host applications and databases.

CHAPTER 5: PROCESS MODEL

5.1 Agile Methodology

- **Plan:** This initial phase involves defining the project goals, requirements, and scope. Planning sets the foundation for the project and helps in determining what needs to be achieved.
- **Design:** In this phase, the overall architecture and design of the project are created. It involves creating detailed specifications and blueprints for the development team to follow.
- **Develop:** This phase is where the actual coding and development take place. The development team works on building the software according to the design specifications.
- **Test:** After development, the software undergoes rigorous testing to identify and fix any bugs or issues. This ensures the quality and functionality of the software.
- **Deploy:** Once testing is complete and the software is deemed ready, it is deployed to a live environment. This means the software is made available for use by the end-users.
- **Review:** In this phase, the deployed software is reviewed to gather feedback from users and stakeholders. This helps in identifying any further improvements or changes needed.
- **Launch:** Finally, after successful reviews and iterations, the software is fully launched for all users. This marks the completion of the development cycle.

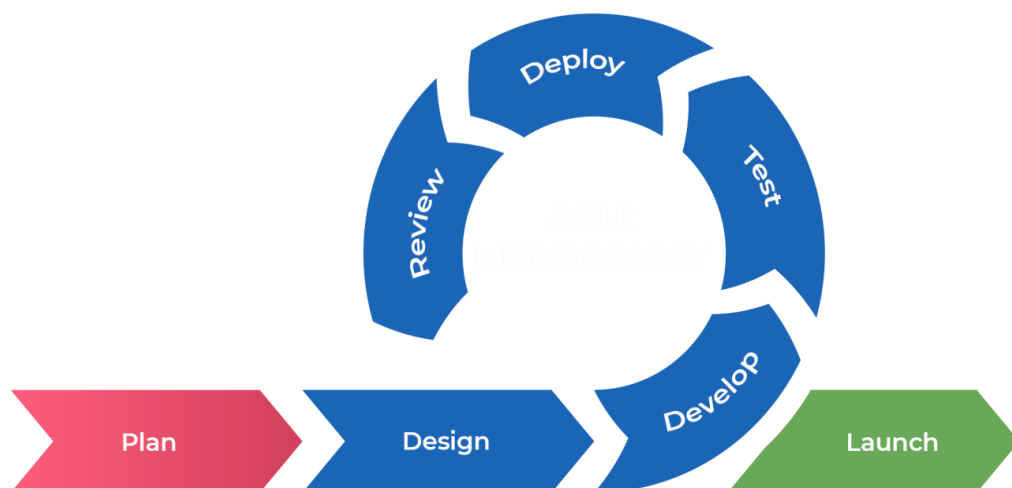


Figure 5.1 Agile Process Model

5.2 Agile Process for Project

1. Plan

Sprint Planning:

- Break the project into 2-week sprints, each focusing on delivering incremental functionalities.
- Prioritize features in the product backlog based on importance and dependencies.

Product Backlog: Create a backlog with prioritized features

- User authentication and role-based access.
- Resident and staff database management.
- Visitor management system with notification features.
- Billing and payment tracking.
- Alerts and notification system for events and emergencies.
- Admin panel for managing users, vendors, and staff.
- Helpdesk for complaint tracking and resolution.

2. Design

- **UI/UX Design:** Create wireframes and mockups for key interfaces, including login screens, dashboards, and notification layouts.
- **System Architecture:** Define the architecture for backend (NodeJS, JavaScript, MySQL) and frontend (HTML, CSS).
- **API Design:** Custom APIs for user authentication, visitor tracking, and alerts.

3. Develop

Sprint Breakdown

- **Sprint 1:**
 - Backend setup (Java, MySQL, AWS).
 - Implement user authentication and role-based access.
- **Sprint 2:**
 - Resident profile management (CRUD operations).
 - Notifications for events and emergencies.
- **Sprint 3:**
 - Visitor management system (check-in/check-out workflow with notifications).
 - Staff entry-exit tracking module.
- **Sprint 4:**
 - Billing and payment tracking system.
 - Helpdesk for complaint tracking.
- **Sprint 5:**
 - Admin panel for managing users, vendors, and staff.
 - Vendor and service management features.

- **Sprint 6:**
 - Security enhancements (multi-factor authentication and encryption).
 - Reports and analytics module for administrators.

4. Test

- Unit Testing: Write and execute tests for individual components.
- Integration Testing: Ensure all components (frontend and backend) interact correctly.

5. Deploy

- Continuous Deployment: Set up CI/CD pipeline for automatic deployment of updates.
- Initial Deployment: Deploy the MVP to a live environment for early user access and feedback.

6. Launch: Deploy the fully tested and completed platform.

Chapter 6: PROJECT PLAN

The project is broken down into the following phases:

- 1. Week 1-2: Requirement Analysis**
 - Defining the scope and objectives.
- 2. Week 3-4: System Design**
 - Create diagrams (Use Case, ERD, DFD, Sequence, Class).
 - Database schema finalized.
- 3. Week 5-6: Front-End Development**
 - Build UI components.
- 4. Week 7-8: Back-End Development**
 - Set up MySQL/MongoDB databases.
 - Implement API in Java.
 - Develop service management.
- 5. Week 9: Testing Phase 1**
 - Unit and integration tests.
 - Debugging front-end and back-end component.
- 6. Week 11: Deployment Preparation**
 - Setup cloud infrastructure.
 - Finalize app store submissions.
- 7. Week 12: Deployment and User Training**
 - Launch the app on Google Play and Apple Store.
 - Prepare user manuals for customers and service providers

Chapter 7: SYSTEM DESIGN

Diagrams:

7.1. Use Case Diagram:

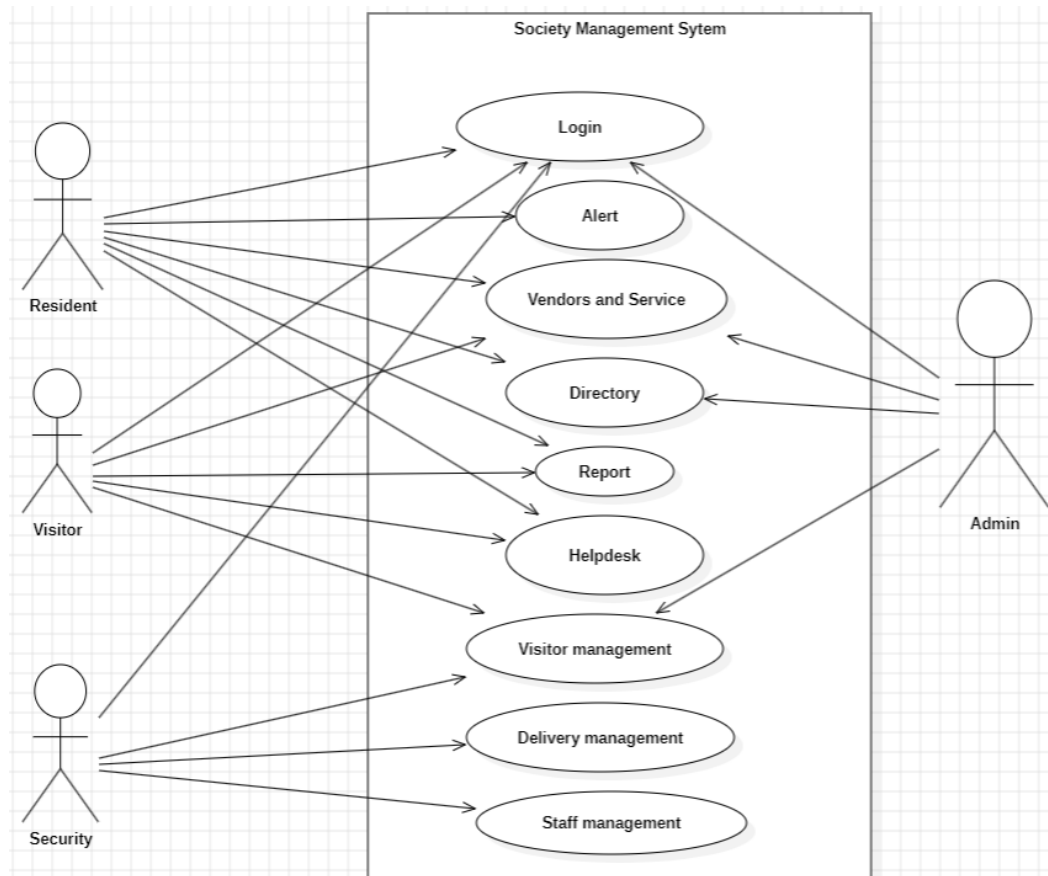


Figure 7.1 Use Case Diagram

The Use Case Diagram for the society management system illustrates the interactions between different types of users and the system's features. It includes actors such as Residents, Society Administrators, and Security Staff. Residents perform actions like logging in, accessing alerts, interacting with vendors and services, viewing the directory, submitting reports, and using the helpdesk. Society Administrators oversee functionalities such as vendor and service management and visitor management. Security Staff focus on managing visitors, deliveries, and staff. The diagram highlights how these actions relate to the system's core functionalities, ensuring seamless integration and role-specific access to various features

7.2 . Activity Diagram:

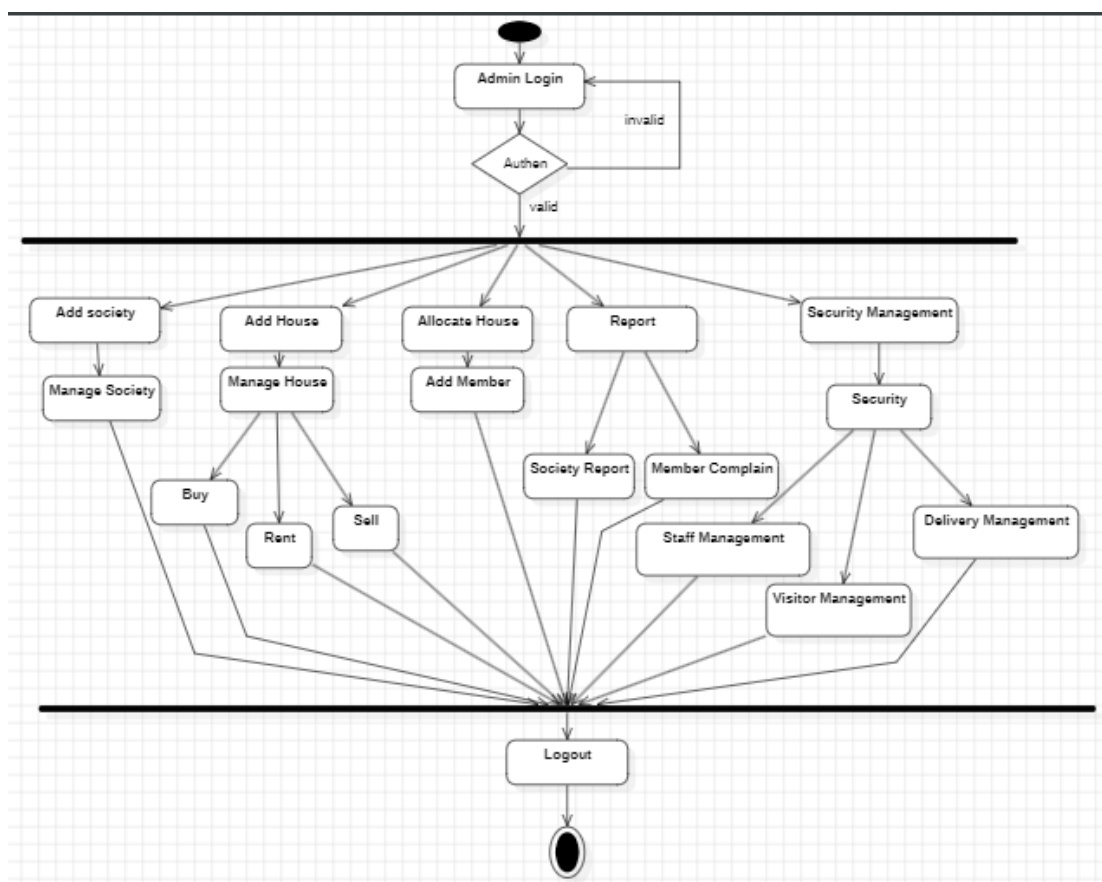


Figure 7.2 Activity Diagram

The Activity Diagram illustrates the workflow for managing society administration in the system. It begins with the Admin logging in, followed by authentication to verify access. Once authenticated, the process moves through key activities such as adding new societies and houses, allocating houses to residents, and overseeing security management. Decision points, such as successful login and available houses, guide the flow of actions. The diagram also highlights how Security Staff manage visitor and delivery tracking, ensuring compliance with safety protocols. This visualization clarifies the step-by-step process and the interconnection of actions, ensuring efficient management of society operations.

7.3. Class Diagram:

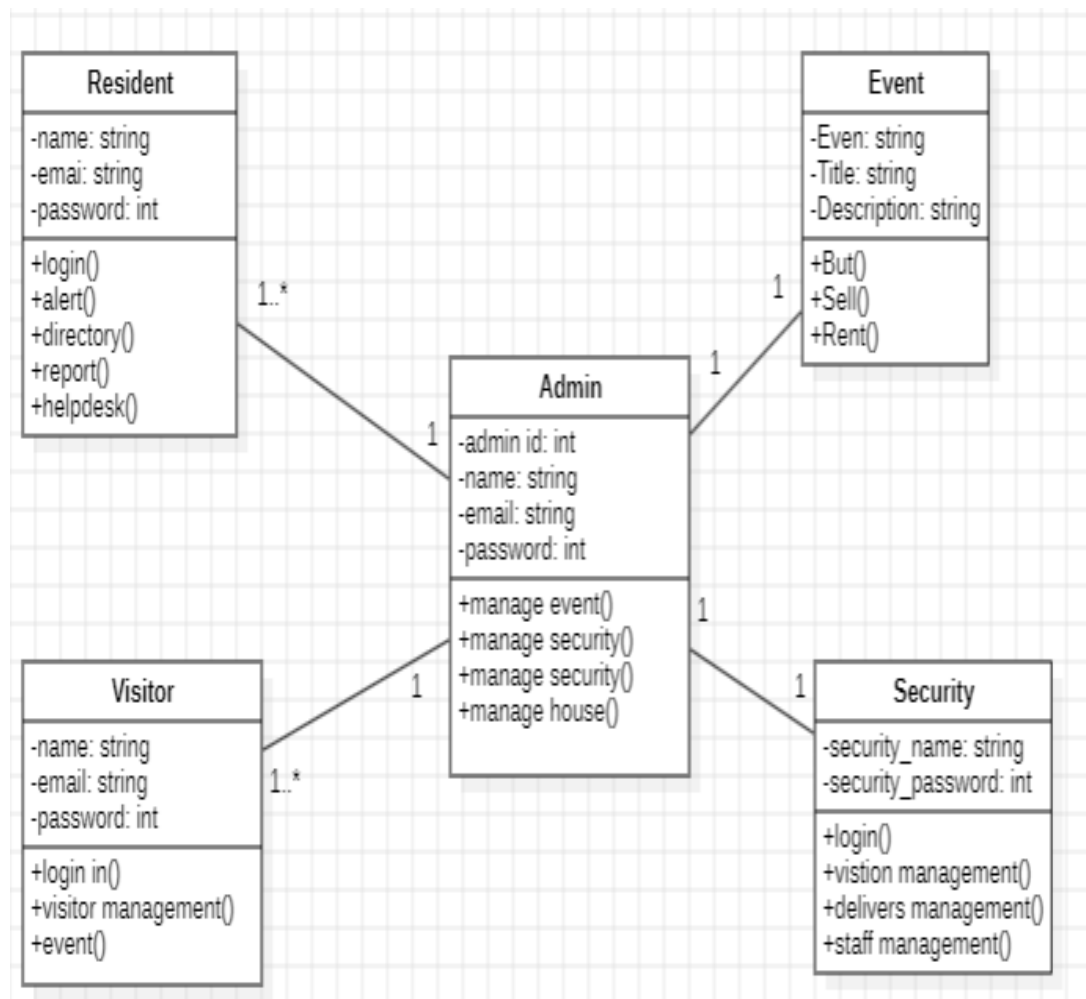


Figure 7.3. Class Diagram

The Class Diagram provides a structured view of the society management system's data model. It defines several key classes:

Admin: Oversees the overall operations of the society, including managing security, events, and house allocations.

Resident: Represents society residents, enabling activities like login, viewing the directory, accessing the helpdesk, and receiving alerts.

Event: Manages events within the society, including functionalities for buying, selling, and renting properties.

Visitor: Handles visitor-related activities, such as logging in and attending events.

Security: Manages staff operations, including delivery tracking and visitor management.

The diagram illustrates relationships such as associations (e.g., an Admin interacts with Security and Event classes) and dependencies (e.g., Resident activities depend on Event and Visitor management). This helps in understanding the system's static structure and how its components interact.

7.4. Sequence Diagram:

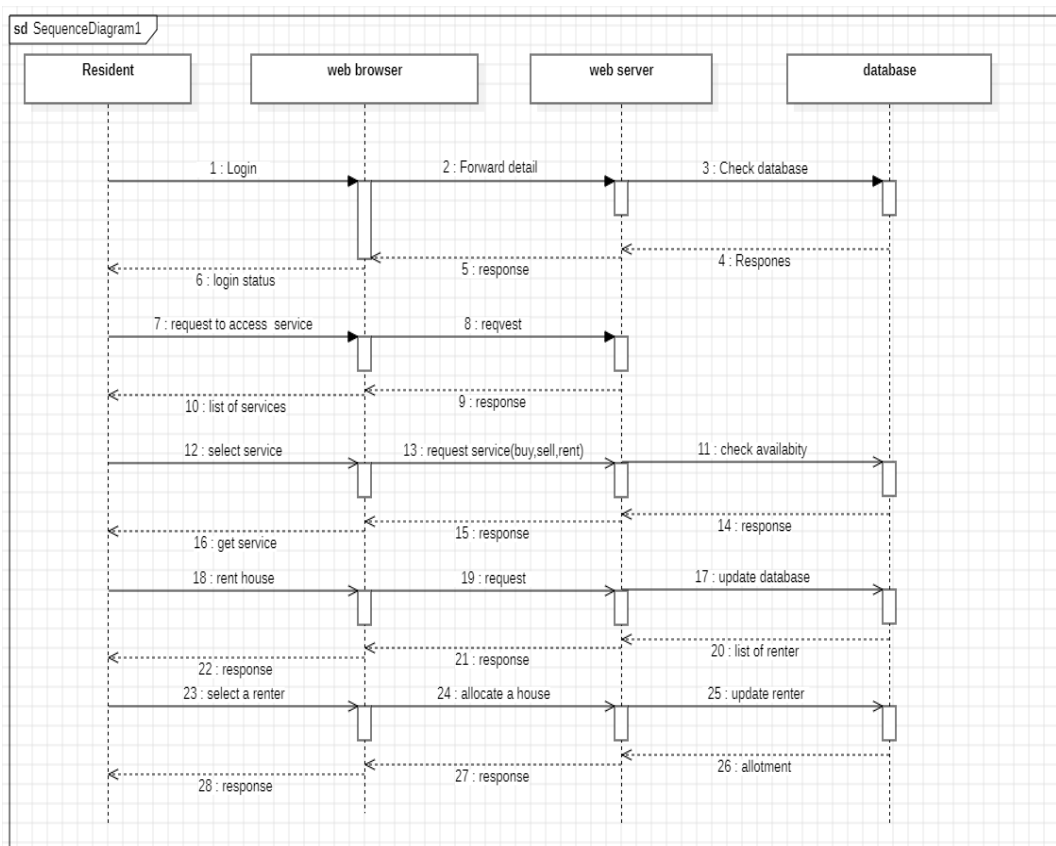


Figure 7.4. Sequence Diagram

The Sequence Diagram shows the interactions between a Resident, web browser, web server, and database in the society management system. It starts with the Resident logging in, followed by authentication through the web server and database. Once authenticated, the Resident requests services, selects one (e.g., buying, selling, or renting), and the server checks availability in the database. The system updates the database and confirms the transaction, providing a clear view of the step-by-step process and message flow.

7.5 ER Diagram:

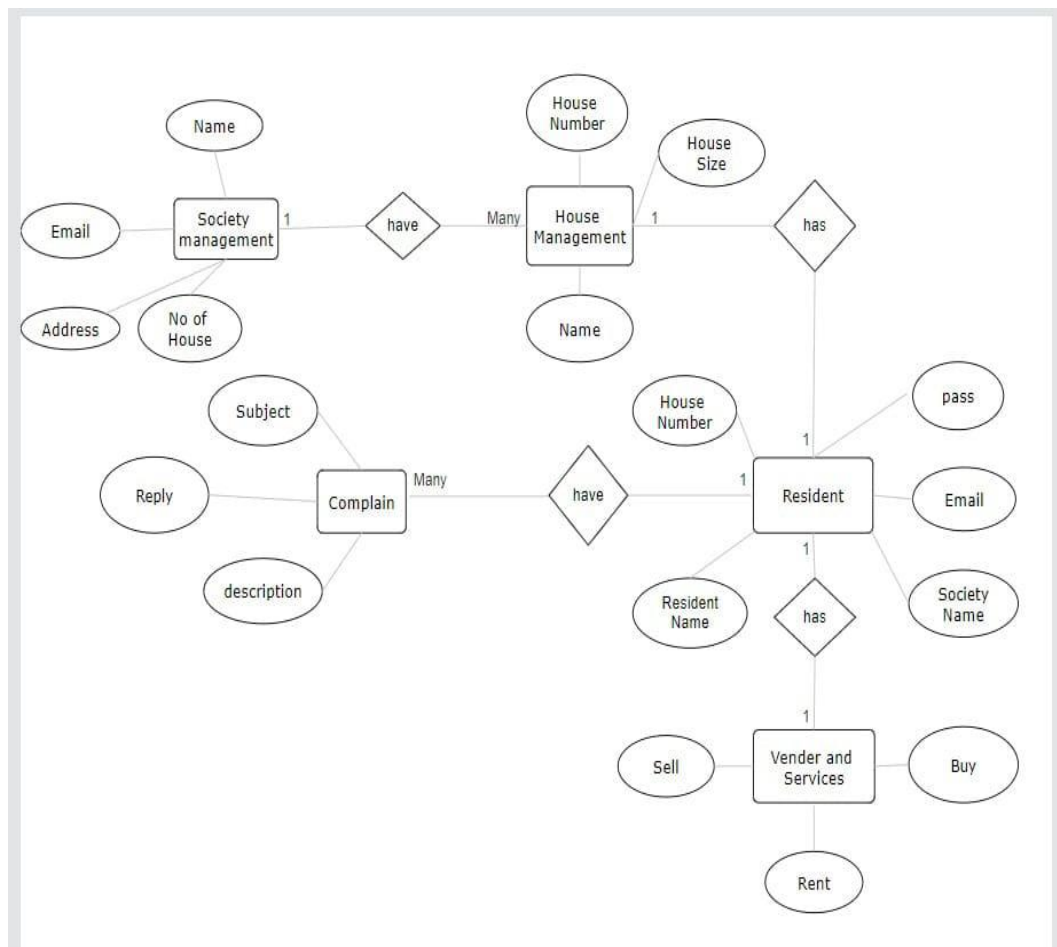


Figure 7.5 ER Diagram

This Entity-Relationship (ER) diagram represents the structure of a Society Management System, highlighting its main entities and their relationships. The system consists of entities such as Society Management, House Management, Resident, Complain, and Vendor and Services, each with their respective attributes like Name, Email, Address, House Number, and Subject.

The diagram shows relationships, such as:

- A Society Management entity manages multiple houses.
- A House Management entity connects with residents and tracks house details like size and number.
- Residents can raise multiple complaints, each linked with a subject, description, and reply.
- A Resident interacts with vendors and services for activities like buying, selling, or renting.
- This diagram helps understand the system's data organization, highlighting how various components interact to facilitate effective society management.

7.6. DFD:

7.6.1 LEVEL 0:

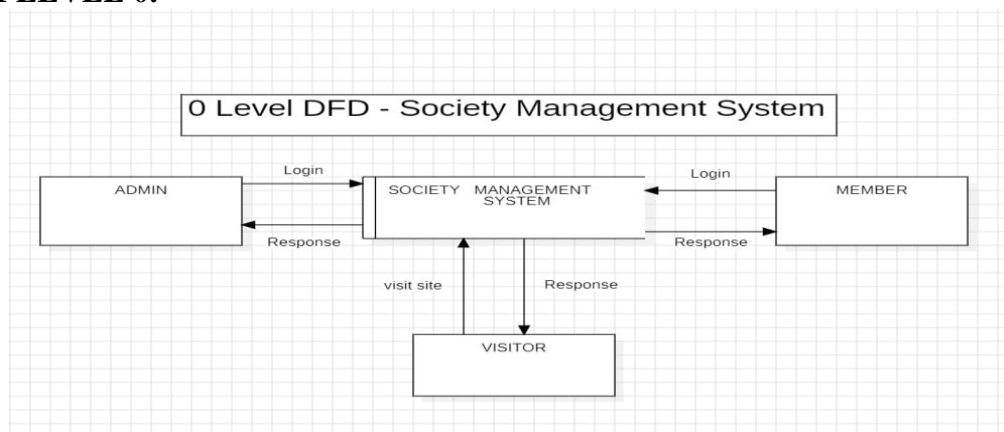


Figure 7.6 LEVEL 0

The Level 0 DFD of the Society Management System illustrates the interactions between the system and external entities: Admin, Member, and Visitor. The system handles login requests from Admin and Member, providing responses, while the Visitor simply visits the site without logging in. This diagram provides a high-level overview of how each entity communicates with the system.

7.6.2 LEVEL 1:

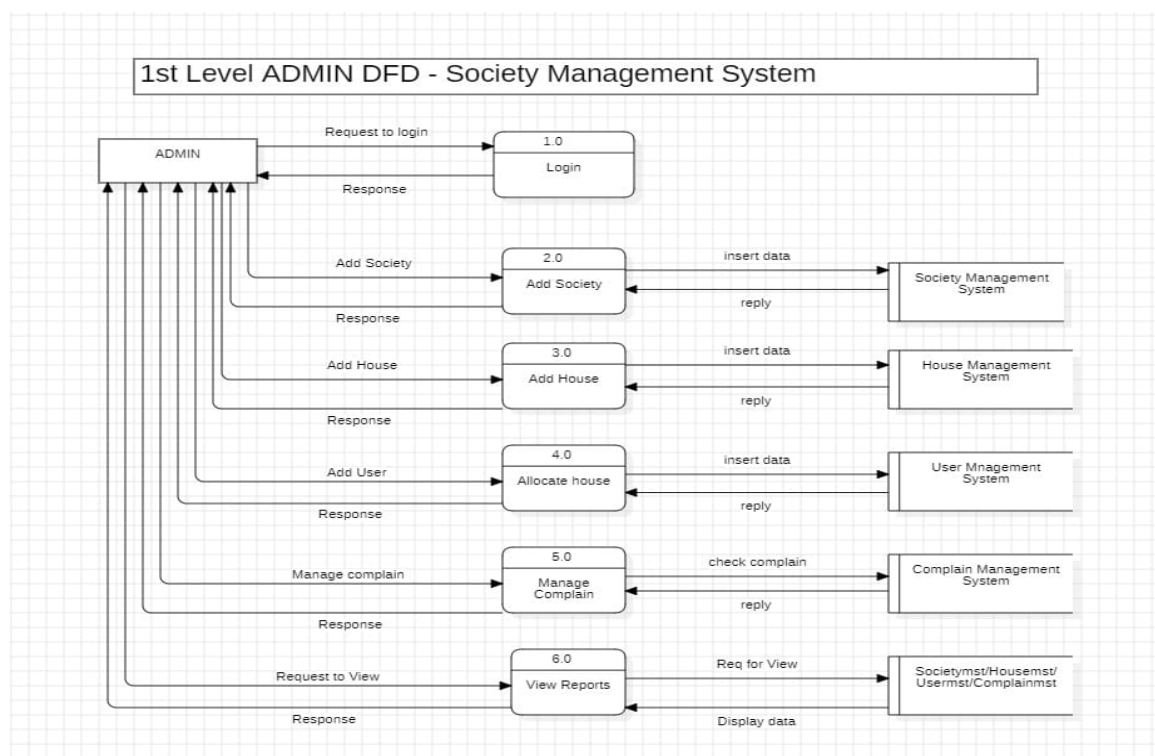


Figure 7.7 LEVEL 1

The Level 1 DFD for the Admin in the Society Management System provides a detailed breakdown of the admin's interactions with various system subprocesses. It shows how the admin logs in, adds societies, houses, and users, allocates houses, manages complaints, and views reports. Each subprocess, such as Add Society, Add House, and Manage Complain, involves inserting data and receiving responses from associated

systems like the Society Management System, House Management System, and Complaint Management System. This diagram illustrates the flow of data between the admin and the system's different components, offering a more detailed view of the internal processes and their interactions.

7.6.3 LEVEL 2:

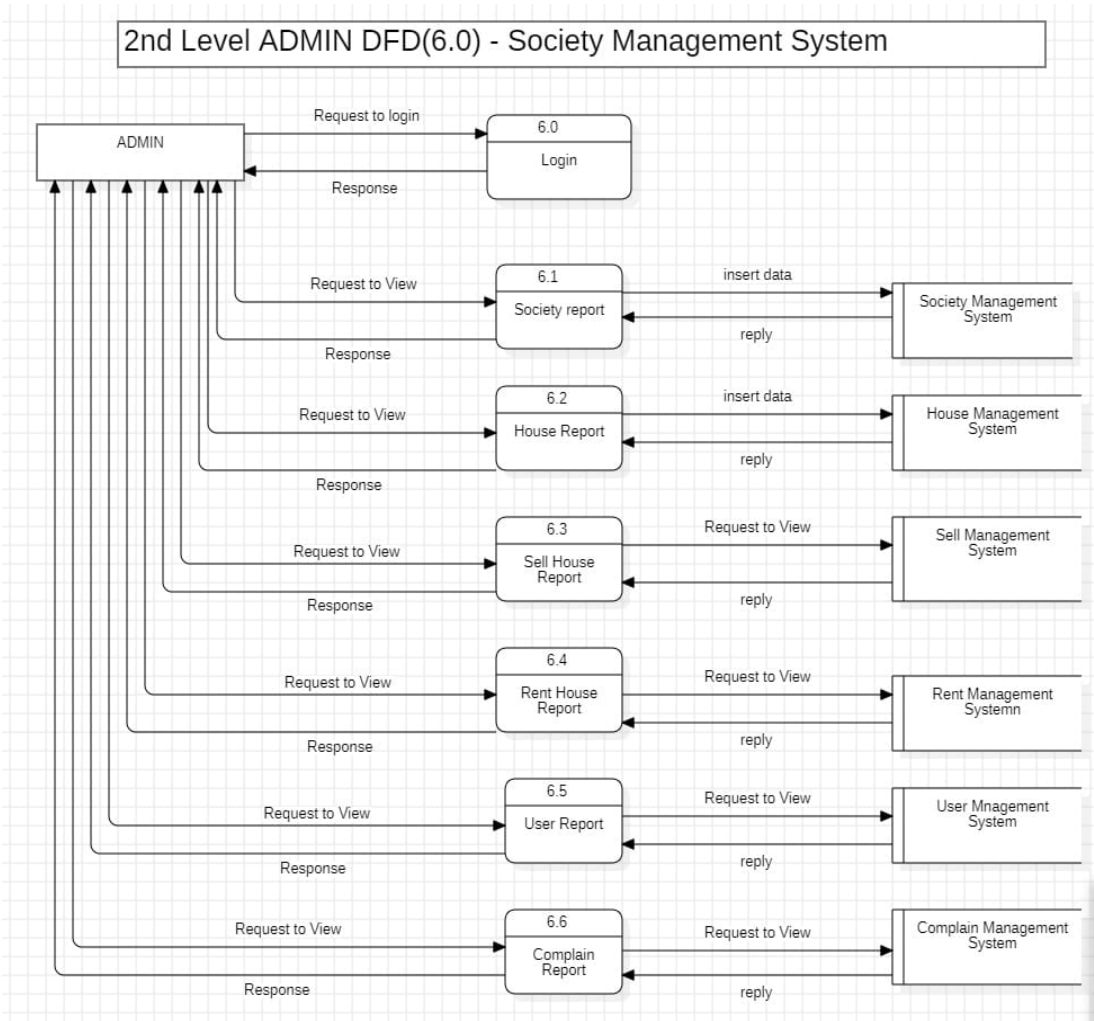


Figure 7.8 LEVEL 2

The Level 2 Data Flow Diagram (DFD) for the Society Management System provides an in-depth view of the administrative functions, focusing on various reports and management processes. It outlines the interaction between the Admin and the system's key subsystems, such as Society Management, House Management, Sell Management, Rent Management, User Management, and Complain Management. Each subsystem handles specific operations, including generating reports, inserting data, and replying to user queries. The diagram clearly depicts the data flow, requests, and responses between the Admin and these components, illustrating how they collectively support the system's functionality. This detailed visualization helps in understanding the intricate workings and data exchanges within the system's administrative module.

7.7 DATABASE TABLES

7.7.1 Users Details Table:

| Column Name | Data Type | Description |
|-------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| username | VARCHAR(255) | User's full name |
| password | VARCHAR(255) | Password |

7.1 Users Details Table

7.7.2 Staff Details Table:

| Column Name | Data Type | Description |
|-------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| name | VARCHAR(255) | Staff's full name |
| email | VARCHAR(255) | Staff's email address |
| phone | VARCHAR(20) | Contact number |
| role | ENUM | Role of staff |

7.2 Staff Details Table

7.7.3 Tickets Details Table:

| Column Name | Data Type | Description |
|-------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| Subject | Varchar(255) | Issue related lines |
| Description | Text | Brief Description |
| Priority | Enum | Priority of issue |
| Status | Enum | Status of issue |
| Created at | Timestamp | Time |

7.3 Tickets Details Table

7.7.4 Visitor Details Table:

| Column Name | Data Type | Description |
|---------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| name | VARCHAR(100) | Visitor's full name |
| phone | VARCHAR(20) | Visitor's phone no. |
| visiting date | DATETIME | Visiting date |
| meeting with | VARCHAR(100) | Resident name |
| purpose | VARCHAR(50) | Purpose to meet |
| Created at | TIMESTAMP | Time of visit |

7.4 Visitor Details Table

7.7.5 Deliveries Details Table:

| Column Name | Data Type | Description |
|------------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| recipient_name | VARCHAR(255) | full name |
| package_details | TEXT | Package Description |
| expected_arrival | Date | Date of package arrival |

*7.5 Deliveries Details Table***7.7.6 Directory-Contact Detail Table:**

| Column Name | Data Type | Description |
|-------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| name | VARCHAR(100) | full name |
| role | VARCHAR(100) | Role |
| phone | VARCHAR(20) | Phone Number |
| email | VARCHAR(100) | Email ID |

*7.6 Directory-Contact Details Table***7.7.7 Properties Detail Table:**

| Column Name | Data Type | Description |
|----------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| property_type | VARCHAR(20) | property name |
| Owner_name | VARCHAR(100) | Owner name |
| Contact_number | VARCHAR(20) | Phone Number |
| description | TEXT | Description of property |
| Created_at | TIMESTAMP | Time |

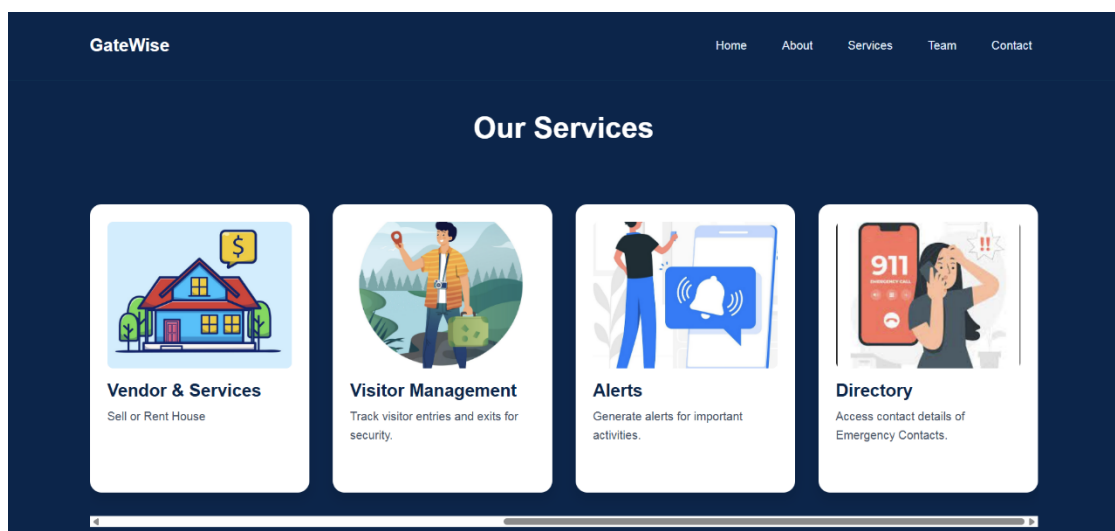
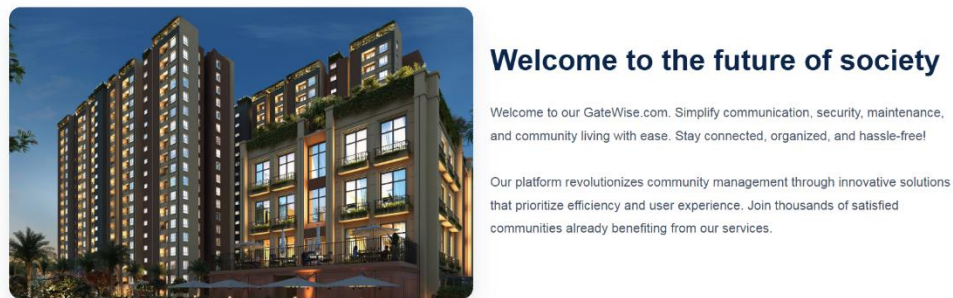
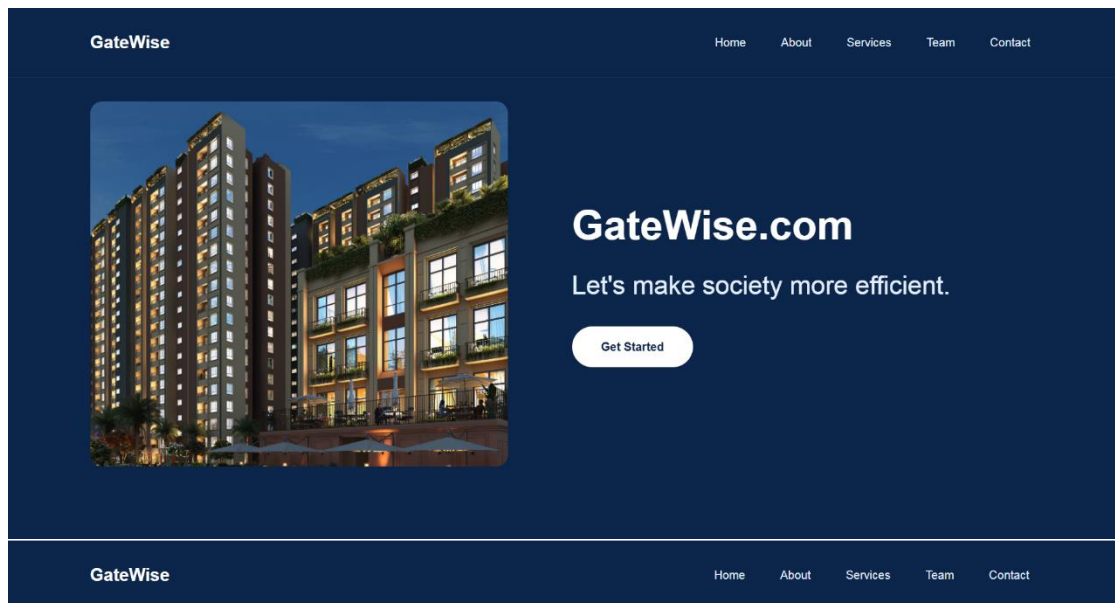
*7.7 Properties Details Table***7.7.8 Alerts Detail Table:**

| Column Name | Data Type | Description |
|-------------|--------------|--------------------------------|
| id | INT | Primary key, unique identifier |
| Event_name | VARCHAR(255) | Name of the event |
| Description | Text | Detail of the event |
| Event_date | Date | Date of the event |
| Created_at | TIMESTAMP | Time |

7.8 Alerts Details Table

7.8 User Interface:

7.8.1 Home Page



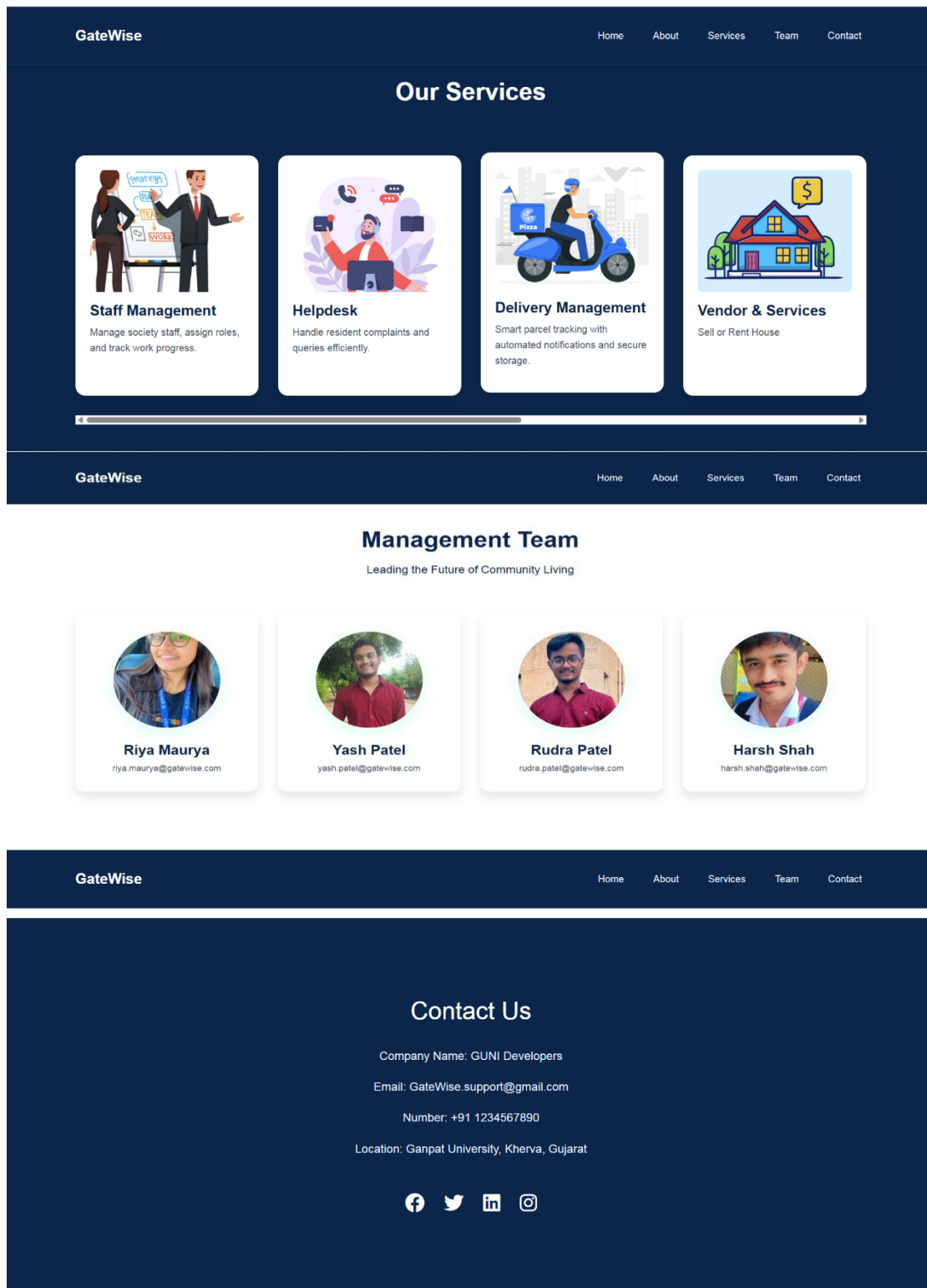


Fig. 7.9 Home Page

The GateWise Community Dashboard provides a key community updates and services. It displays scheduled maintenance alerts, such as elevator downtime, and notices from residents. The dashboard tracks helpdesk activity. It also monitors visitor data. Delivery updates include pending and delivered items like an Amazon parcel, furniture, and documents. The sidebar menu allows navigation to sections like Helpdesk, Staff, Visitor, Delivery, and more. Users can add or remove notices, toggle between light/dark modes, and access their profiles or log out easily.

7.8.2 Sign-Up / Sign-In Page:

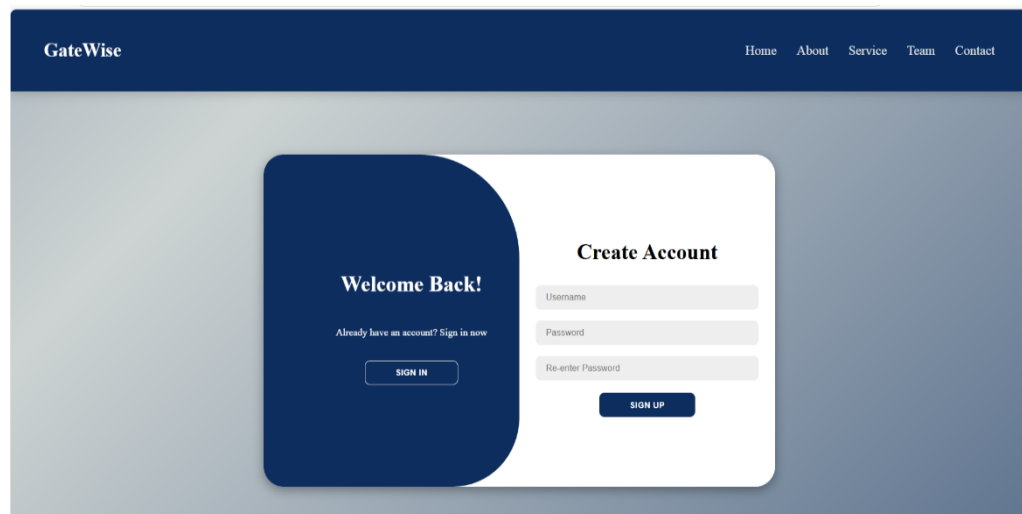


Fig. 7.10 Sign-Up

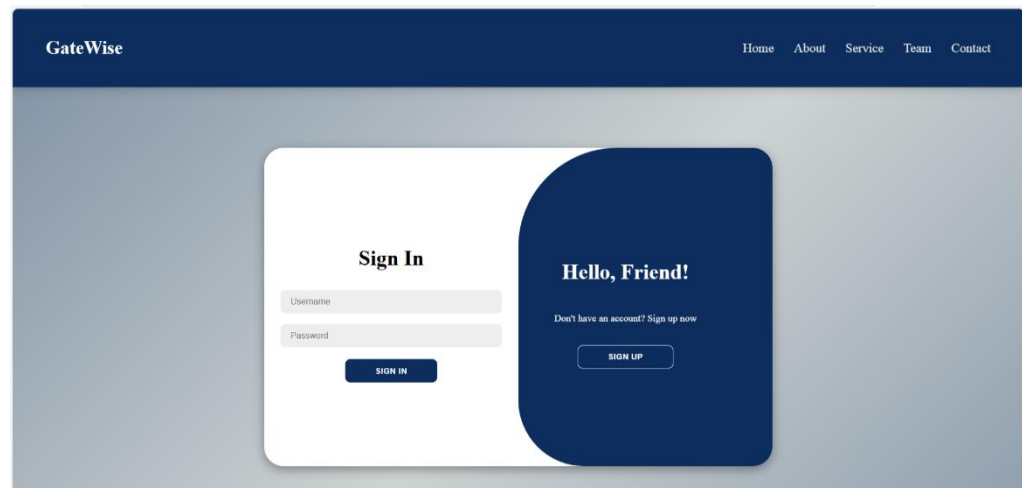


Fig. 7.11 Sign-In

The Signup Page presents a seamless and secure registration experience through its thoughtfully designed user interface. Prominently displaying the signup form with fields for user name, phone number, password, and the page guides users through the account creation process with clear instructions and inline validation

7.8.3 Dashboard

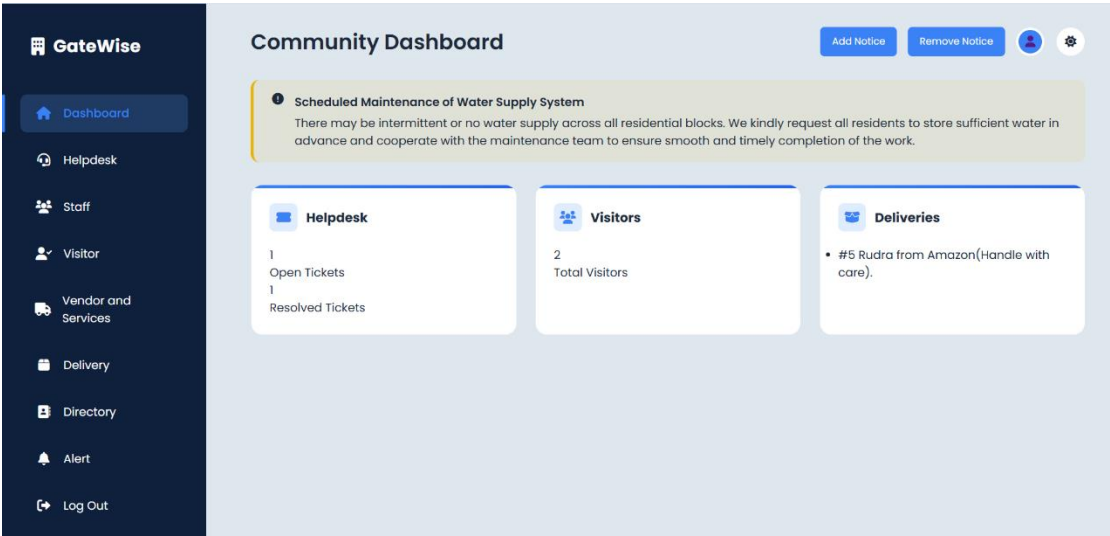


Fig. 7.12 Dashboard

The dashboard of the Society Management Application Admin Panel serves as the central interface, providing administrators with a comprehensive overview of the system's critical statistics and functionalities. The Helpdesk, Manage user, Alert, Directory, Delivery Management, Staff Management, Vendor And Services, Visitor Management and ensuring efficient oversight of society operations.

7.8.4 Profile

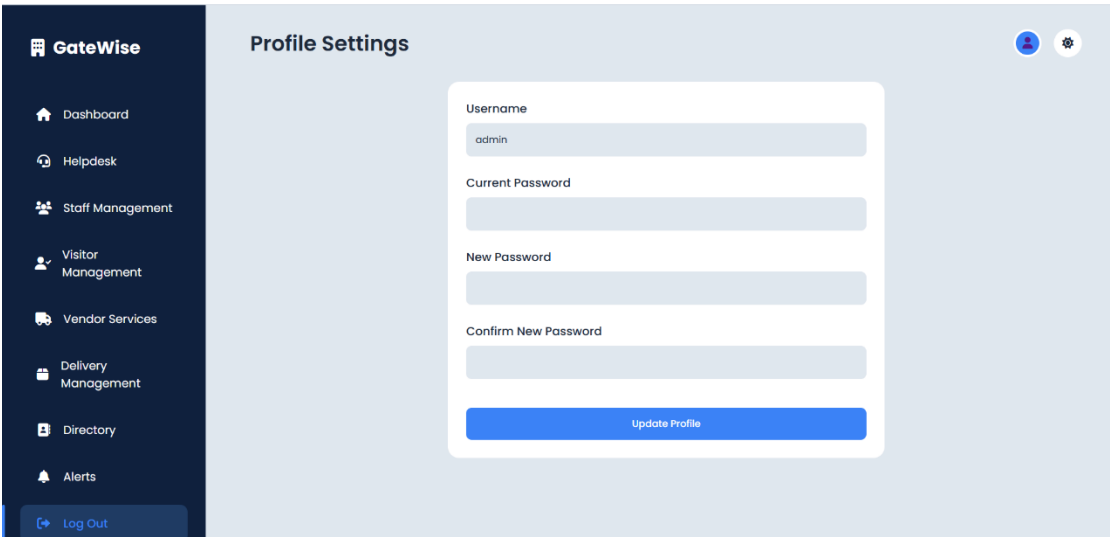


Fig. 7.13 Admin Module

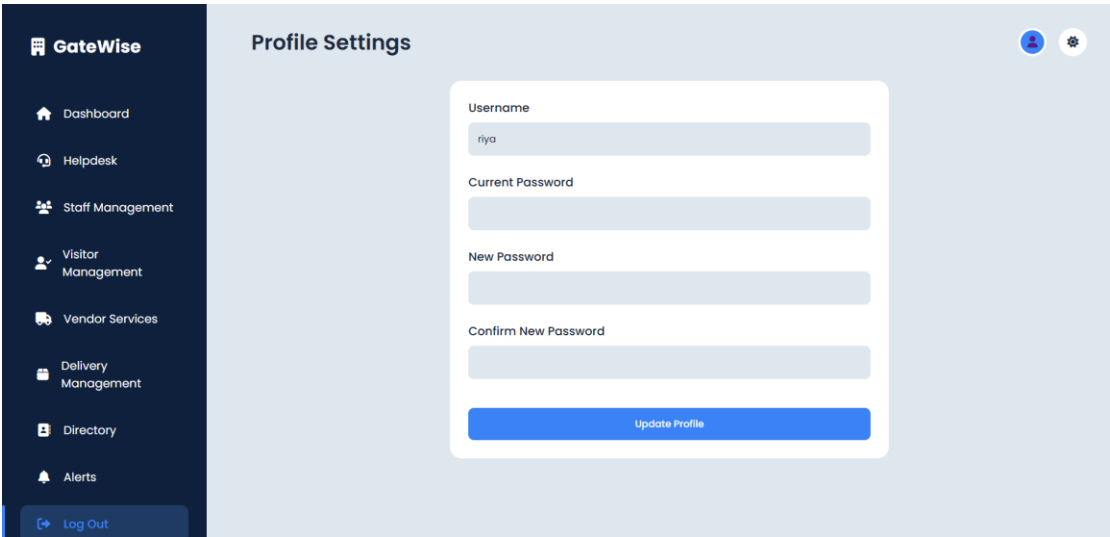


Fig. 7.14 User Module

The Profile feature in a Society Management System web application enables users to manage their personal information. Residents can edit their profile photo, update their name, email, and address, ensuring accurate and up-to-date records for effective communication and system functionality.

7.8.5 Alert

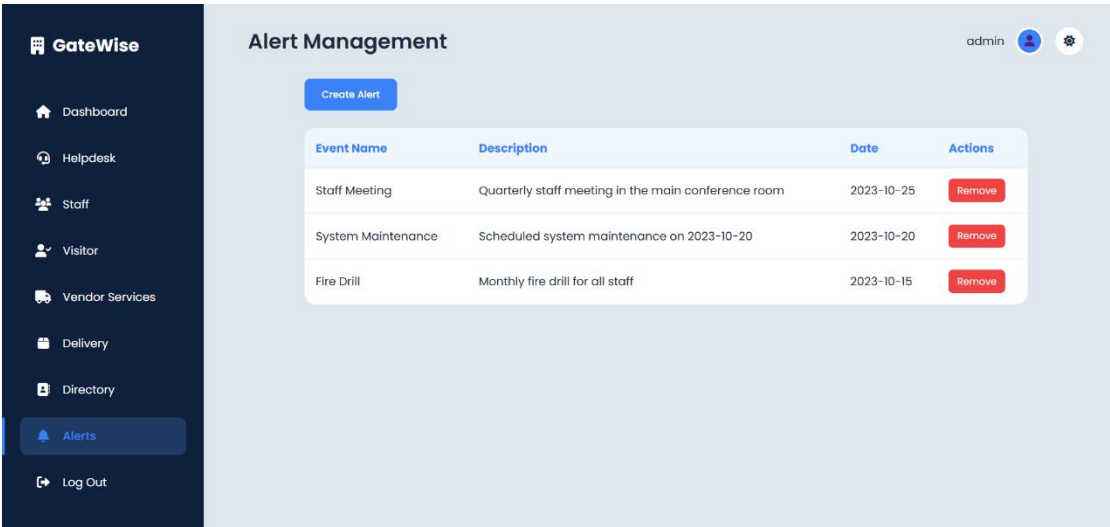


Fig. 7.15 Admin Module

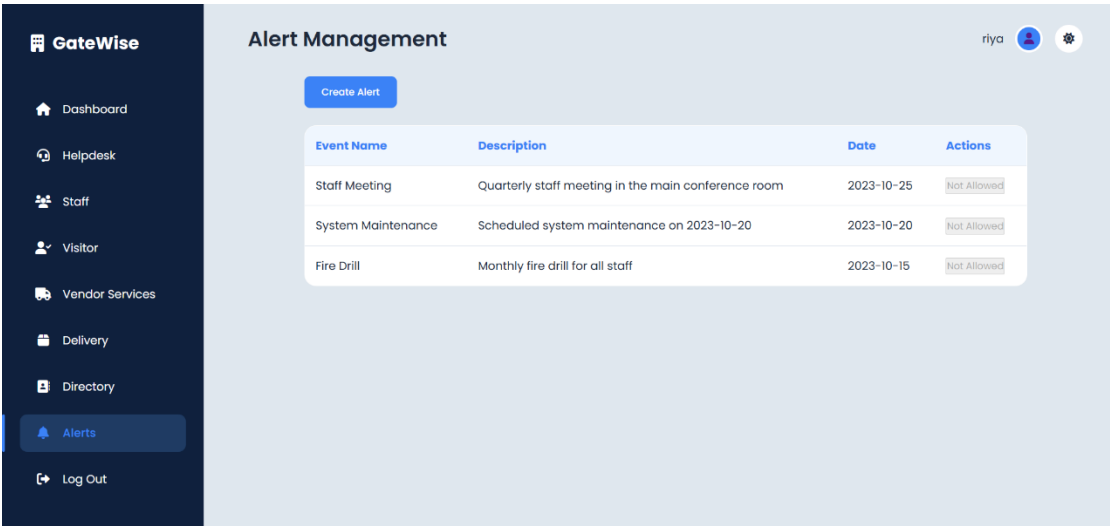


Fig. 7.16 User Module

The Alert feature in a Society Management System web application allows administrators to send emergency messages or notifications about society meetings. This ensures timely communication of important updates, enhancing resident safety and keeping them informed about key events or emergencies within the society.

7.8.6 Vendors and Services

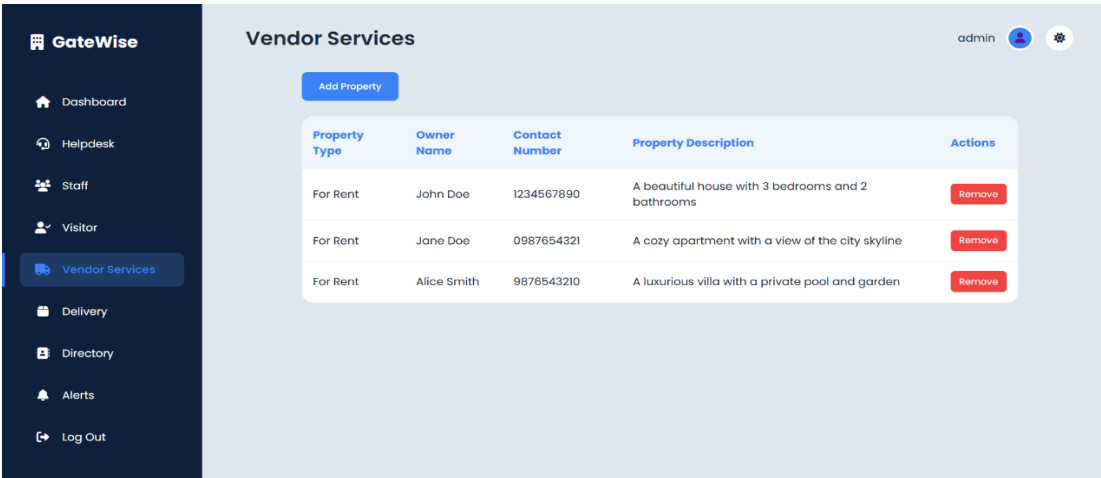


Fig. 7.17 Admin Module

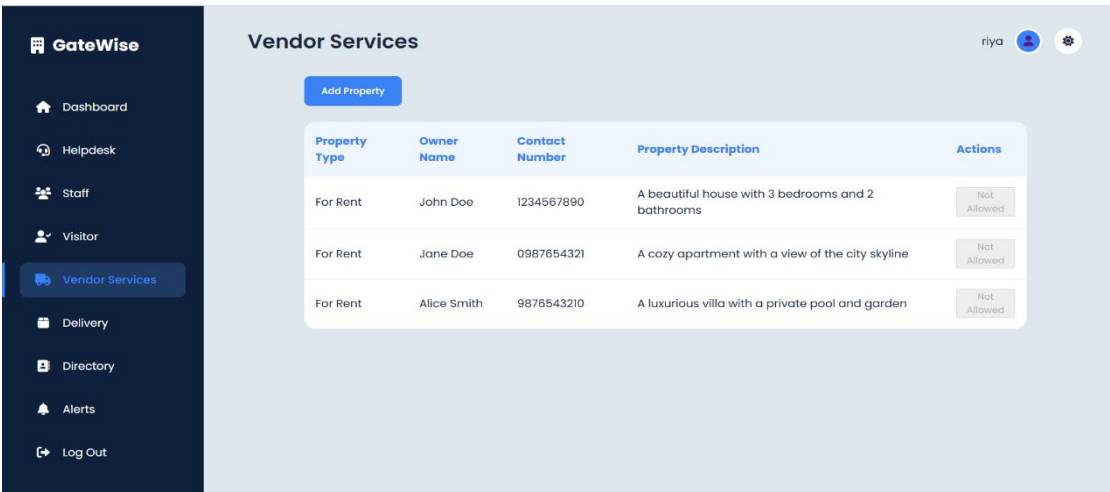


Fig. 7.18 User Module

The Vendor and Services feature in a Society Management System web application allows users to add, remove, or update the status of their property, such as buying, selling, or renting. This function helps residents manage their housing needs efficiently and provides a platform for updating property status, enhancing communication between residents and vendors or service providers.

7.8.7 Directory

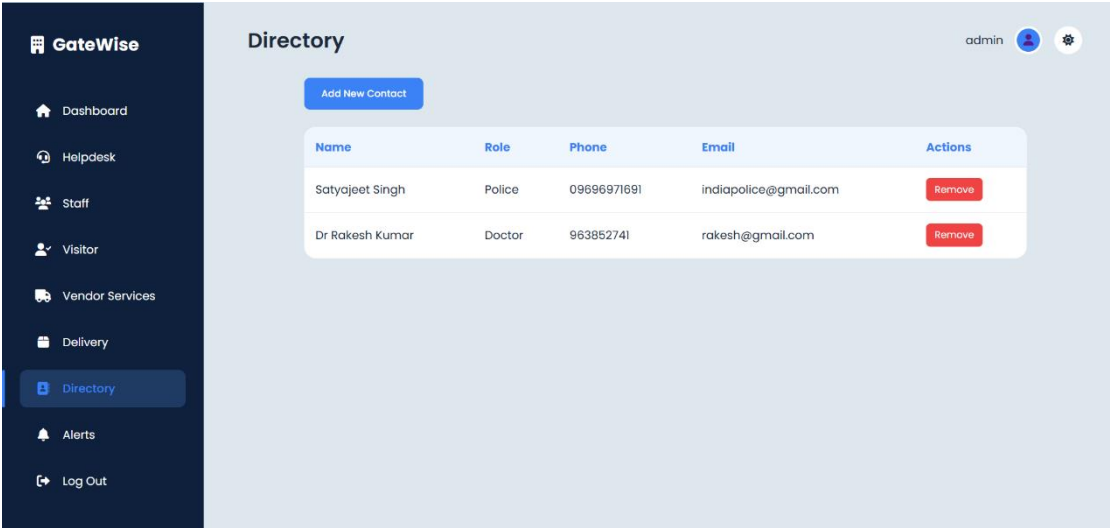


Fig. 7.19 Admin Module

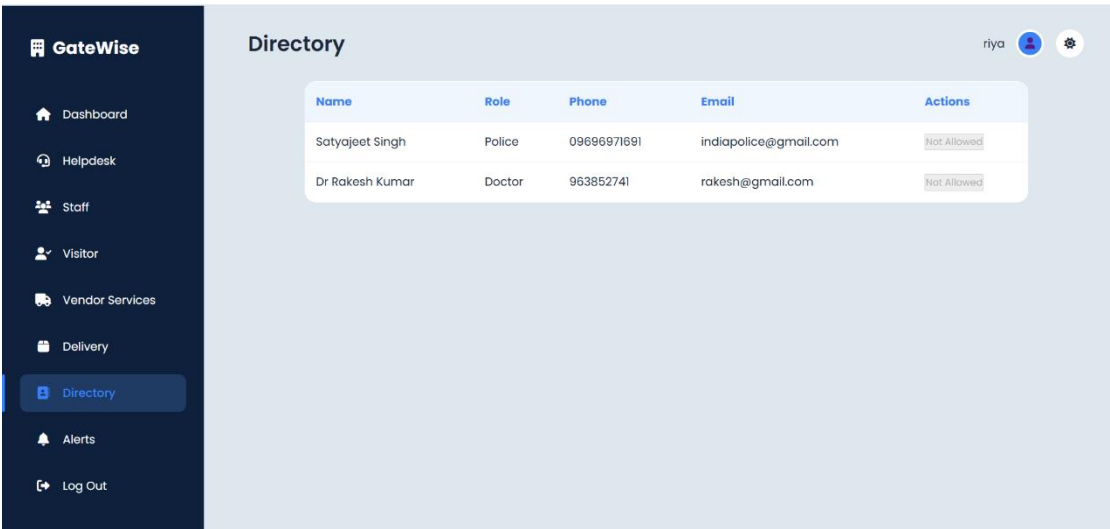


Fig. 7.20 User Module

The Directory feature in a Society Management System web application allows residents to add or edit emergency contacts. This ensures that up-to-date and critical contact information is available for quick access in case of emergencies, improving safety and communication within the society.

7.8.8 Helpdesk

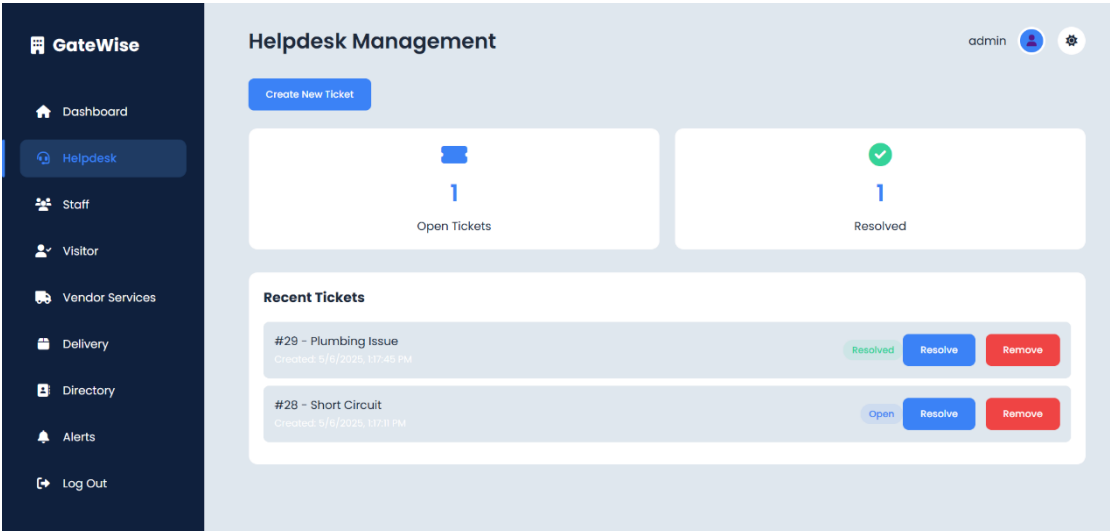


Fig. 7.21 Admin Module

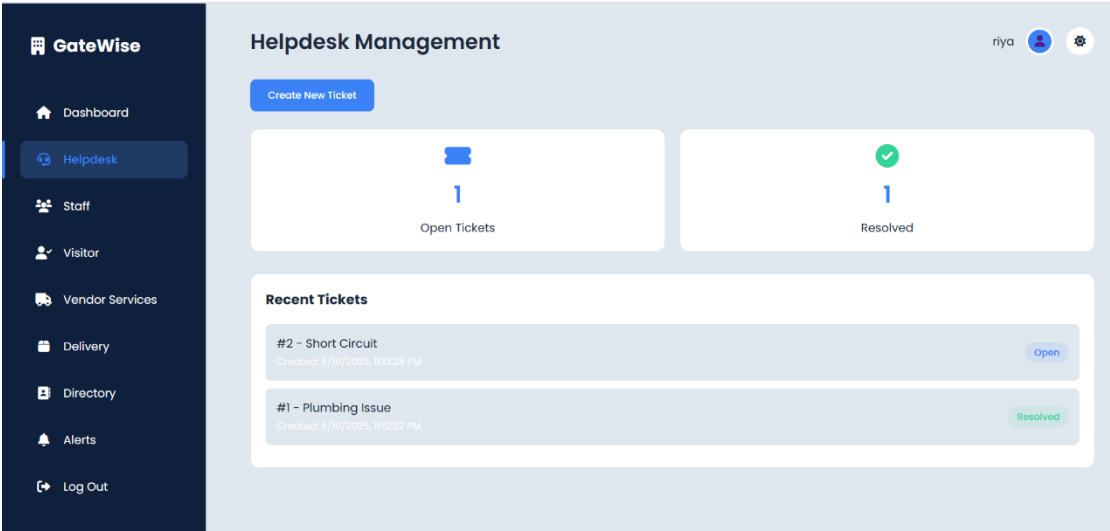


Fig. 7.22 User Module

The Helpdesk in a Society Management System allows residents to log complaints, such as water leakage or broken gates, for efficient resolution. It streamlines issue tracking, task assignment, and progress monitoring, ensuring prompt maintenance and better communication between residents and management.

7.8.9 Visitor Management

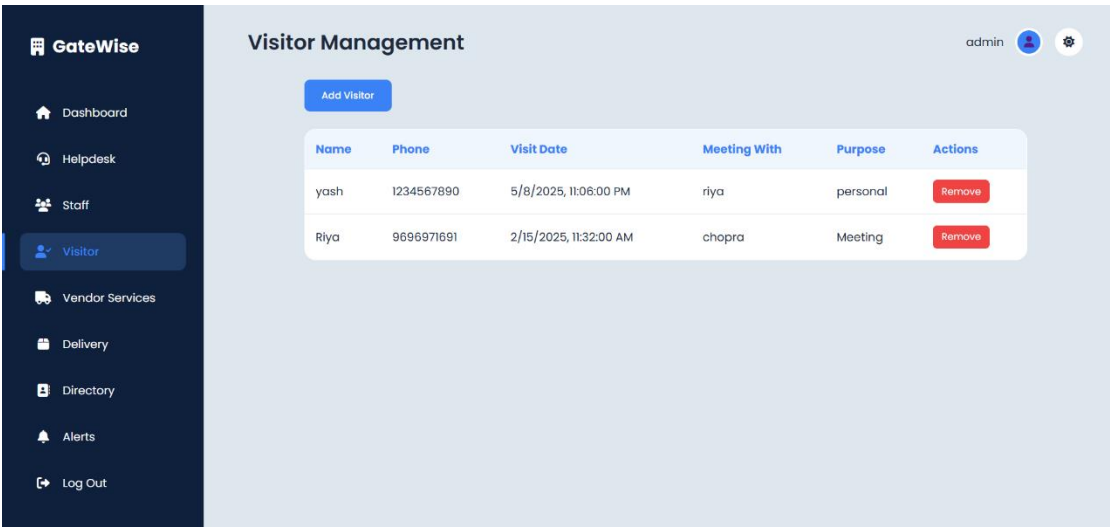


Fig. 7.23 Admin Module

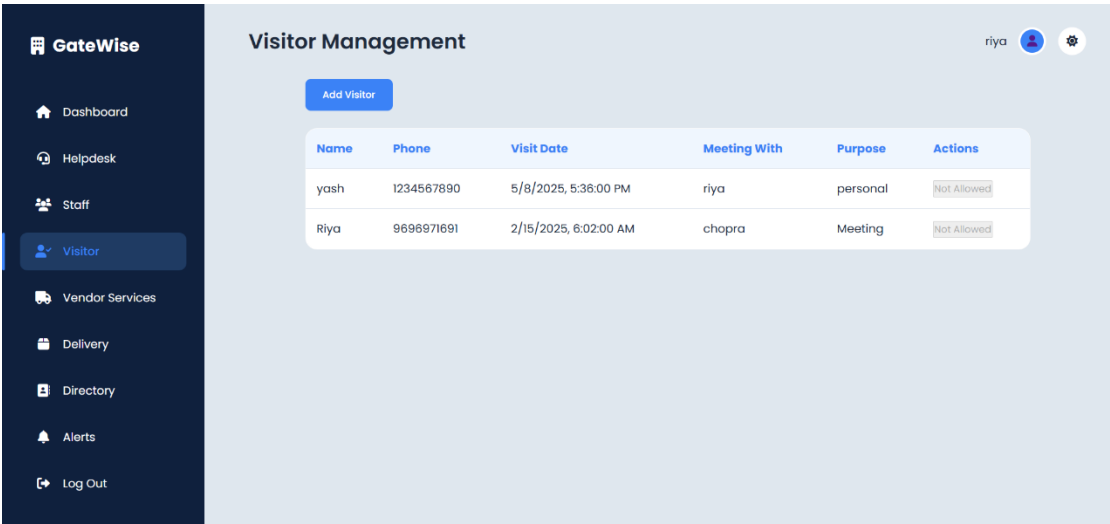


Fig. 7.24 User Module

The Visitor Management feature in a Society Management System web application allows administrators to add, remove, or update the status of upcoming visitors. This ensures proper tracking and coordination of visitor access, improving security and communication between residents and the management team.

7.8.10 Delivery Management

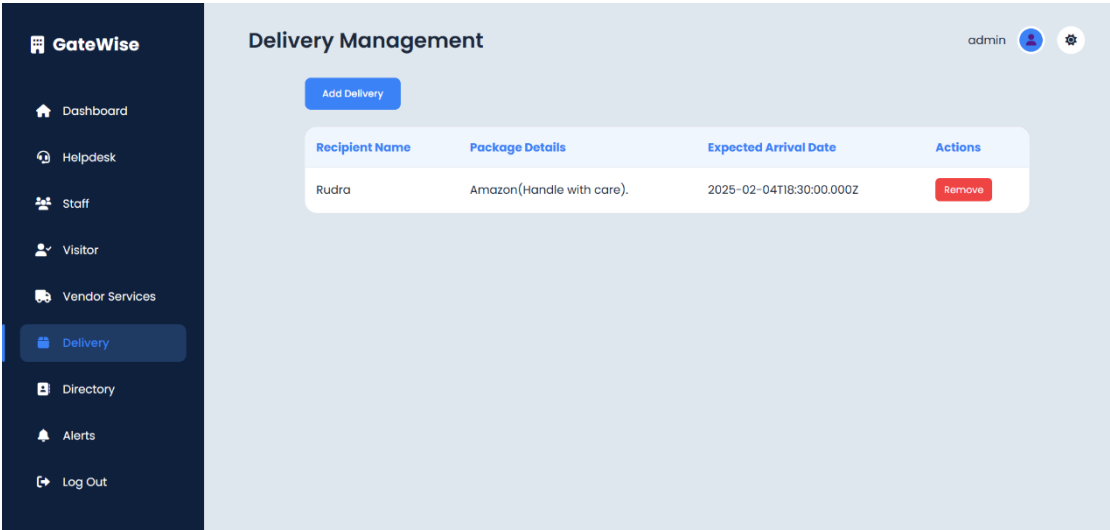


Fig. 7.25 Admin Module

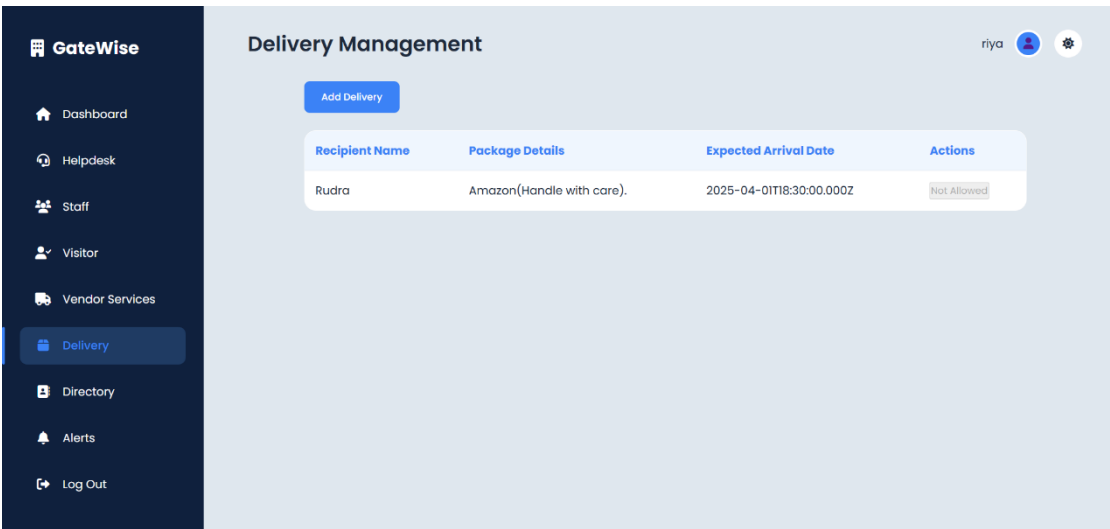


Fig. 7.26 User Module

The Delivery Management feature in a Society Management System web application allows administrators to add, remove, or update the status of deliveries. This ensures efficient tracking of packages and parcels, keeping residents informed about the status of their deliveries and improving overall delivery management within the society.

7.8.11 Staff Management

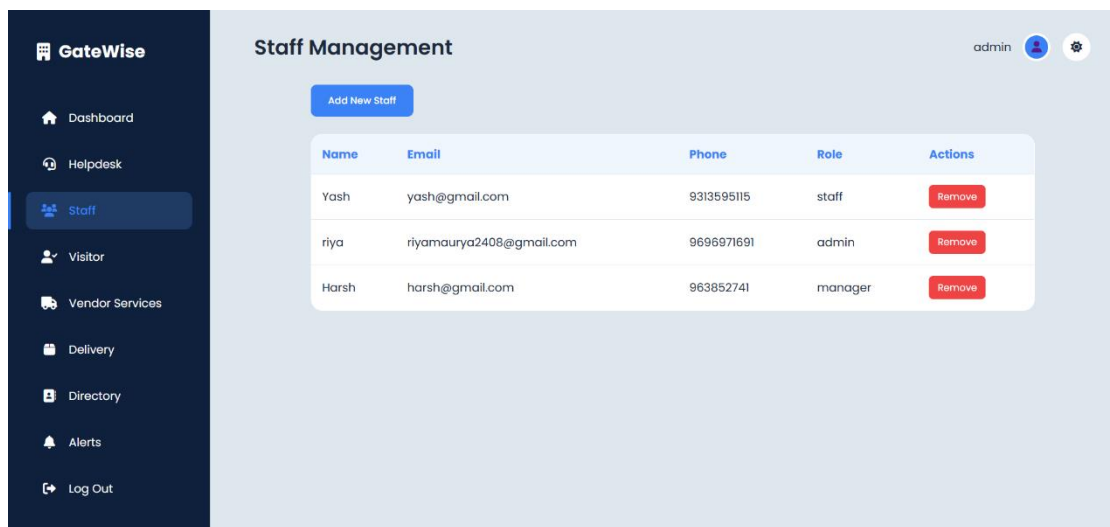


Fig. 7.27 Admin Module

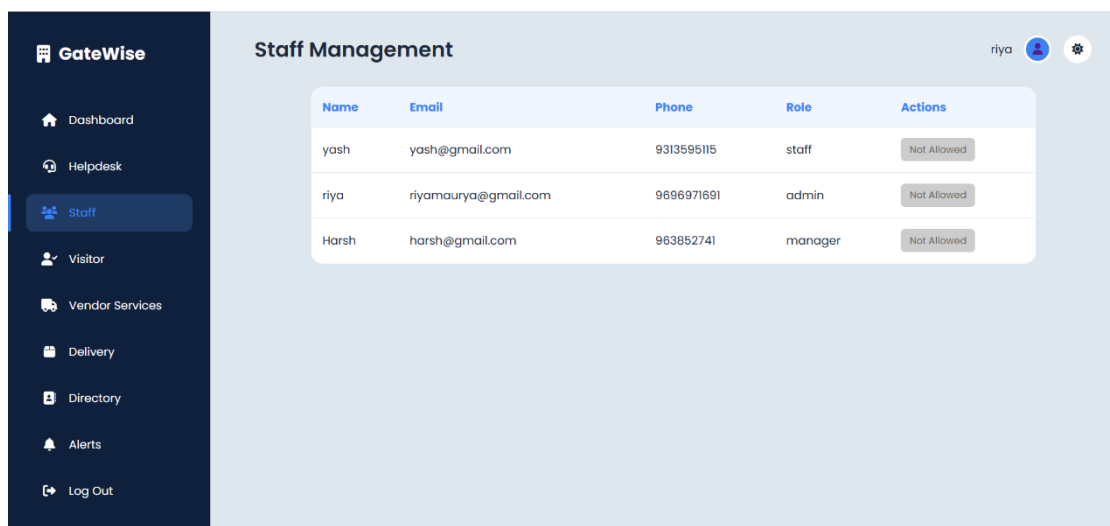


Fig. 7.28 User Module

The Staff Management feature in a Society Management System web application allows administrators to efficiently add or remove staff members. This function helps maintain an up-to-date roster of personnel, ensuring smooth operations and effective management of tasks within the society.

Chapter 8: Implementation Details

8.1 Flowchart of Implementation

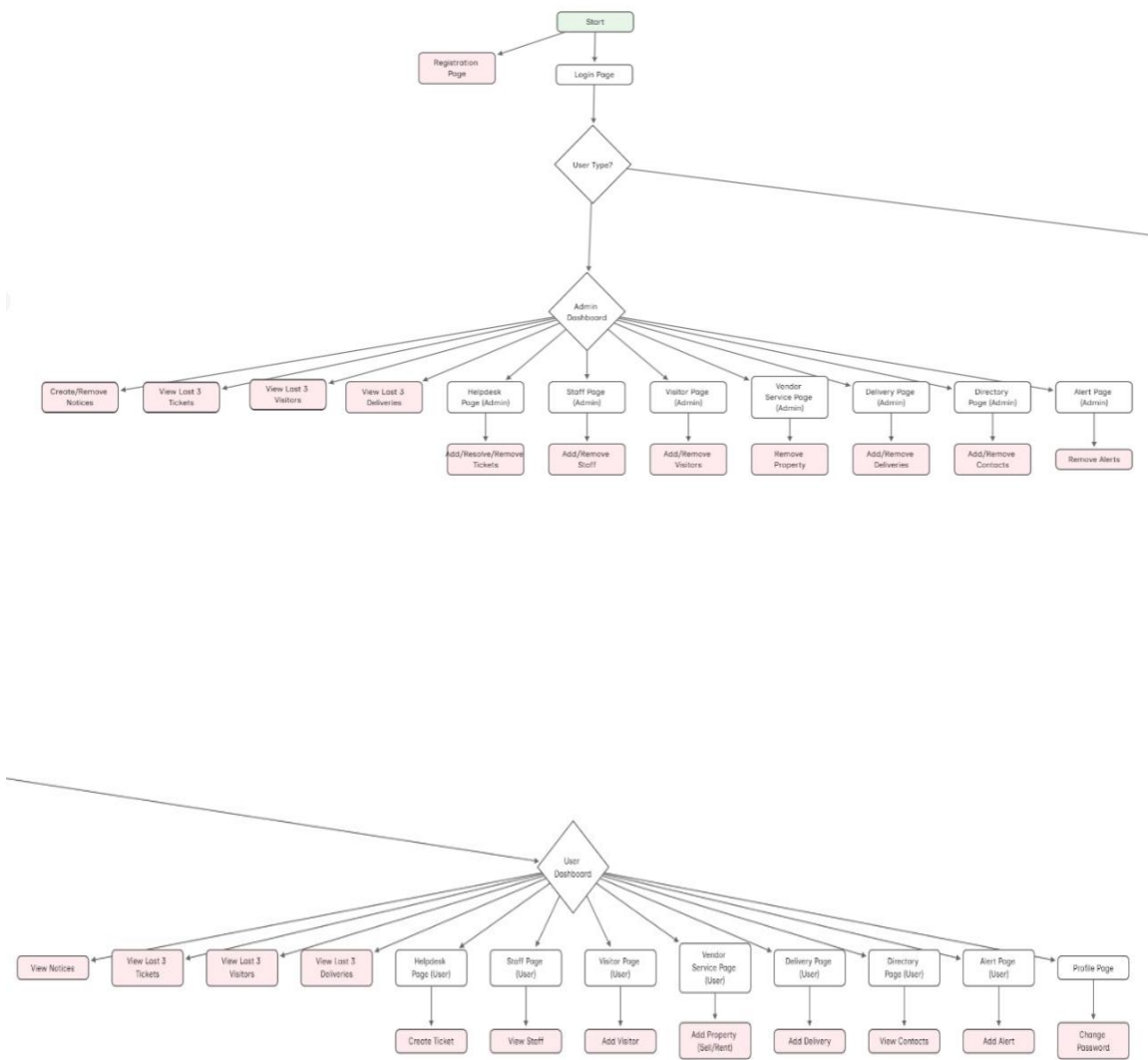


Fig. 8.1 Flowchart

8.2 Actual Program Code

8.2.1 Login

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css">
  <link rel="stylesheet" href="style.css">
  <title>GateWise Login</title>
</head>
<body>
  <nav>
    <div class="logo">
      <h1>GateWise</h1>
    </div>
    <div class="nav-buttons">
      <a href="h10.html">Home</a>
      <a href="h10.html">About</a>
      <a href="h10.html">Service</a>
      <a href="h10.html">Team</a>
      <a href="h10.html">Contact</a>
    </div>
  </nav>

  <div class="container" id="container">
    <div class="form-container sign-up">
      <form id="signupForm">
        <h1>Create Account</h1>
        <input type="text" id="signupUsername" placeholder="Username"
required>
        <input type="password" id="signupPassword" placeholder="Password"
required>
        <input type="password" id="confirmPassword" placeholder="Re-enter
Password" required>
        <button type="submit">Sign Up</button>
      </form>
    </div>

    <div class="form-container sign-in">
      <form id="loginForm">
        <h1>Sign In</h1>
        <input type="text" id="loginUsername" placeholder="Username" required>
        <input type="password" id="loginPassword" placeholder="Password"
required>
        <button type="submit">Sign In</button>
      </form>
    </div>
  </div>

```



```

<div class="toggle-container">
  <div class="toggle">
    <div class="toggle-panel toggle-left">
      <h1>Welcome Back!</h1>
      <p>Already have an account? Sign in now</p>
      <button class="hidden" id="login">Sign In</button>
    </div>
    <div class="toggle-panel toggle-right">
      <h1>Hello, Friend!</h1>
      <p>Don't have an account? Sign up now</p>
      <button class="hidden" id="register">Sign Up</button>
    </div>
  </div>
</div>
</div>
</div>

<script>
  const container = document.getElementById("container");
  const registerBtn = document.getElementById("register");
  const loginBtn = document.getElementById("login");

  registerBtn.addEventListener("click", () => {
    container.classList.add("active");
  });

  loginBtn.addEventListener("click", () => {
    container.classList.remove("active");
  });

  // Function to Save User to Session Storage
  function saveUserToSession(username) {
    sessionStorage.setItem("user", JSON.stringify({ username }));
  }

  // Login Function
  document.getElementById("loginForm").addEventListener("submit", async
(event) => {
    event.preventDefault();
    const username = document.getElementById("loginUsername").value;
    const password = document.getElementById("loginPassword").value;

    try {
      const response = await fetch("http://localhost:5001/login", {

        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ username, password })
      });

      const data = await response.json();
    }
  });

```

```

        if (data.success) {
            saveUserToSession(username); // Store user in session
            alert("Login successful!");
            window.location.href = "dashboard.html"; // Redirect after login
        } else {
            alert(data.message);
        }
    } catch (error) {
        console.error("Login error:", error);
    }
});

// Signup Function
document.getElementById("signupForm").addEventListener("submit", async
(event) => {
    event.preventDefault();

    const username = document.getElementById("signupUsername").value;
    const password = document.getElementById("signupPassword").value;
    const confirmPassword =
document.getElementById("confirmPassword").value;

    if (password !== confirmPassword) {
        alert("Passwords do not match!");
        return;
    }

    try {
        const response = await fetch("http://localhost:5001/register", {

            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ username, password })
        });

        const data = await response.json();
        alert(data.message);

        if (data.success) {
            saveUserToSession(username); // Store user in session
            window.location.href = "dashboard.html";
        }
    } catch (error) {
        console.error("Signup error:", error);
    }
});

// Auto Redirect Logged-In Users
document.addEventListener("DOMContentLoaded", () => {
    if (sessionStorage.getItem("user")) {
        window.location.href = "dashboard.html";
    }
});

```

```

    </script>
  </body>

</html>

```

8.2.2 Dashboard

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Smart Society Dashboard</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700
&display=swap" rel="stylesheet">
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
      font-family: 'Poppins', sans-serif;
    }
    :root {
      --primary-color: #60a5fa;
      --secondary-color: #3b82f6;
      --accent-color: #ef4444;
      --text-primary: #f8fafc;
      --text-secondary: #ffffff;
      --sidebar-bg: #1e293b;
      --main-bg: #0f172a;
      --card-bg: #1e293b;
      --border-color: #334155;
    }

    [data-theme="light"] {
      --primary-color: #3b82f6;
      --secondary-color: #2563eb;
      --accent-color: #ef4444;
      --text-primary: #1e293b;
      --text-secondary: white;
      --sidebar-bg: #0f203d;
      --main-bg: #dfe7ee;
      --card-bg: #ffffff;
      --border-color: #e2e8f0;
    }

    body {
      background: var(--main-bg);
      color: var(--text-primary);

```

```
    transition: background 0.3s ease, color 0.3s ease;
  }

.container {
  display: grid;
  grid-template-columns: 280px 1fr;
  min-height: 100vh;
  gap: 20px;
}

.sidebar {
  background: var(--sidebar-bg);
  padding: 25px;
  position: relative;
  z-index: 2;
  border-right: 1px solid var(--border-color);
  display: flex;
  flex-direction: column;
  gap: 12px;
}

.logo {
  display: flex;
  align-items: center;
  gap: 12px;
  margin-bottom: 20px;
  padding: 15px;
  color: var(--text-secondary);
  font-weight: 700;
  font-size: 1.5rem;
}

.menu-item {
  display: flex;
  align-items: center;
  padding: 14px 20px;
  border-radius: 10px;
  cursor: pointer;
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
  color: var(--text-secondary);
  position: relative;
}

.menu-item:hover {
  background: rgba(96,165,250,0.1);
  color: var(--primary-color);
  transform: translateX(8px);
}

.menu-item.active {
  background: rgba(96,165,250,0.2);
  color: var(--primary-color);
  font-weight: 500;
}
```

```
}

.menu-item.active::before {
  content: "";
  position: absolute;
  left: -25px;
  top: 0;
  height: 100%;
  width: 4px;
  background: var(--primary-color);
  border-radius: 2px;
}

.icon {
  width: 24px;
  text-align: center;
  margin-right: 15px;
  font-size: 1.1rem;
}

.main-content {
  padding: 30px 40px;
}

.header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 30px;
  gap: 20px;
}

.header h1 {
  font-size: 1.8rem;
  font-weight: 600;
  white-space: nowrap;
}

.header-controls {
  display: flex;
  gap: 15px;
  align-items: center;
  height: 40px;
}

.profile-btn {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 40px;
  height: 40px;
  border-radius: 50%;
  background: var(--card-bg);
```

```
border: 1px solid var(--border-color);
cursor: pointer;
transition: all 0.3s ease;
}

.profile-icon {
width: 32px;
height: 32px;
border-radius: 50%;
background: var(--primary-color);
display: flex;
align-items: center;
justify-content: center;
}

.theme-toggle {
cursor: pointer;
width: 40px;
height: 40px;
border-radius: 50%;
background: var(--card-bg);
border: 1px solid var(--border-color);
display: flex;
align-items: center;
justify-content: center;
}

.admin-btn {
padding: 8px 16px;
background: var(--primary-color);
border: none;
border-radius: 6px;
color: white;
cursor: pointer;
transition: opacity 0.3s ease;
height: 40px;
line-height: 24px;
}

.cards-container {
display: grid;
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
gap: 25px;
margin-top: 20px;
}

.card {
background: var(--card-bg);
padding: 25px;
border-radius: 16px;
transition: transform 0.3s ease, box-shadow 0.3s ease;
position: relative;
overflow: hidden;
```

```
        border: 1px solid var(--border-color);
    }

    .card:hover {
        transform: translateY(-5px);
        box-shadow: 0 8px 32px rgba(0,0,0,0.1);
    }

    .card::before {
        content: "";
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 4px;
        background: linear-gradient(90deg, var(--primary-color), var(--secondary-
color));
    }

    .card h3 {
        margin-bottom: 20px;
        font-size: 1.1rem;
        display: flex;
        align-items: center;
        gap: 12px;
    }

    .card-icon {
        width: 38px;
        height: 38px;
        background: rgba(96,165,250,0.2);
        border-radius: 8px;
        display: flex;
        align-items: center;
        justify-content: center;
        color: var(--primary-color);
    }

    .progress-bar {
        height: 6px;
        background: var(--border-color);
        border-radius: 3px;
        overflow: hidden;
        margin: 15px 0;
    }

    .progress-fill {
        height: 100%;
        width: 65%;
        background: linear-gradient(90deg, var(--primary-color), var(--secondary-
color));
        border-radius: 3px;
        transition: width 0.5s ease;
    }
```

```
}

.notice {
  background: rgba(234, 179, 8, 0.1);
  padding: 18px 25px;
  border-radius: 12px;
  margin-bottom: 30px;
  display: flex;
  align-items: flex-start;
  gap: 15px;
  border-left: 4px solid #eab308;
  position: relative;
}

.notice-content {
  flex: 1;
}

.notice-title {
  font-weight: 600;
  margin-bottom: 5px;
}

.modal {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0,0,0,0.5);
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

.modal-content {
  background: var(--card-bg);
  padding: 25px;
  border-radius: 12px;
  width: 90%;
  max-width: 500px;
  border: 1px solid var(--border-color);
}

.modal-content h3 {
  margin-bottom: 15px;
}

.modal-input {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
}
```



```
background: var(--main-bg);
border: 1px solid var(--border-color);
color: var(--text-primary);
border-radius: 8px;
}

.modal-actions {
  display: flex;
  gap: 10px;
  margin-top: 15px;
}

.modal-btn {
  flex: 1;
  padding: 10px;
  border: none;
  border-radius: 8px;
  cursor: pointer;
}

.modal-btn.primary {
  background: var(--primary-color);
  color: white;
}

.modal-btn.secondary {
  background: var(--border-color);
  color: var(--text-primary);
}

.delete-btn {
  margin-left: auto;
  padding: 4px 8px;
  background: var(--accent-color);
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

@keyframes fadeIn {
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
}

.card {
  animation: fadeIn 0.6s ease forwards;
}

@media (max-width: 768px) {
  .container {
    grid-template-columns: 1fr;
  }
}
```

```

.sidebar {
  display: none;
}
.header {
  flex-wrap: wrap;
}
.header-controls {
  gap: 8px;
}
.admin-btn {
  padding: 6px 12px;
  font-size: 0.9rem;
}
.profile-btn,
.theme-toggle {
  width: 36px;
  height: 36px;
}
}
</style>
</head>
<body>
  <div class="container">
    <aside class="sidebar">
      <div class="logo">
        <i class="fas fa-building"></i>
        GateWise
      </div>
      <div class="menu-item active">
        <i class="fas fa-home icon"></i>
        Dashboard
      </div>
      <div class="menu-item" onclick="window.location.href='help.html';">
        <i class="fas fa-headset icon"></i>
        Helpdesk
      </div>
      <div class="menu-item" onclick="window.location.href='staff.html';">
        <i class="fas fa-users icon"></i>
        Staff
      </div>
      <div class="menu-item" onclick="window.location.href='visitor.html';">
        <i class="fas fa-user-check icon"></i>
        Visitor
      </div>
      <div class="menu-item" onclick="window.location.href='vendor.html';">
        <i class="fas fa-truck icon"></i>
        Vendor and Services
      </div>
      <div class="menu-item" onclick="window.location.href='delivery.html';">
        <i class="fas fa-box icon"></i>
        Delivery
      </div>
      <div class="menu-item" onclick="window.location.href='directory.html';">

```

```

        <i class="fas fa-address-book icon"></i>
        Directory
    </div>
    <div class="menu-item" onclick="window.location.href='alert.html';">
        <i class="fas fa-bell icon"></i>
        Alert
    </div>
    <div class="menu-item" onclick="window.location.href='h10.html';">
        <i class="fas fa-sign-out-alt icon"></i>
        Log Out
    </div>
</aside>

<main class="main-content">
    <div class="header">
        <h1>Community Dashboard</h1>
        <div class="header-controls">
            <button class="admin-btn" id="addNoticeBtn">Add Notice</button>
            <button class="admin-btn" id="removeNoticeBtn">Remove
Notice</button>
            <a href="profile.html">
                <div class="profile-btn">
                    <div class="profile-icon">
                        <i class="fas fa-user text-white"></i>
                    </div>
                </div>
            </a>
            <div class="theme-toggle" id="themeToggle">
                <i class="fas fa-moon" id="themeIcon"></i>
            </div>
        </div>
    </div>

    <div class="notices-container" id="noticesContainer">
        <!-- Notices will be dynamically added here -->
    </div>

    <div class="cards-container">
        <div class="card">
            <h3><div class="card-icon"><i class="fas fa-ticket-
alt"></i></div>Helpdesk</h3>
            <div class="stats">
                <div>
                    <div class="stat-number" id="openTicketsCount">0</div>
                    <div class="stat-label">Open Tickets</div>
                </div>
                <div>
                    <div class="stat-number" id="resolvedTicketsCount">0</div>
                    <div class="stat-label">Resolved Tickets</div>
                </div>
            </div>
            <!-- <div class="progress-bar">
                <div class="progress-fill"></div>
            </div>

```

```

        </div> -->
    </div>

    <div class="card">
        <h3><div class="card-icon"><i class="fas fa-
users"></i></div>Visitors</h3>
        <div class="stats">
            <div>
                <div class="stat-number" id="totalVisitorsCount">0</div>
                <div class="stat-label">Total Visitors</div>
            </div>
            <!-- <div>
                <div class="stat-number">9</div>
                <div class="stat-label">Checked-in</div>
            </div> -->
        </div>
        <!-- <div class="progress-bar">
            <div class="progress-fill" style="width: 50%"></div>
        </div> -->
    </div>

    <div class="card">
        <h3><div class="card-icon"><i class="fas fa-box-
open"></i></div>Deliveries</h3>
        <ul class="delivery-list" id="deliveryList">
            <!-- Last 3 deliveries will be dynamically added here -->
        </ul>
    </div>
</div>
</main>
</div>

<!-- Modals
<div id="passwordModal" class="modal">
    <div class="modal-content">
        <h3>Admin Verification</h3>
        <input type="password" id="adminPassword" class="modal-input"
placeholder="Enter admin password">
        <div class="modal-actions">
            <button class="modal-btn secondary"
onclick="closeModal('passwordModal')">Cancel</button>
            <button class="modal-btn primary"
id="verifyPasswordBtn">Verify</button>
        </div>
    </div>
</div>
</div> -->

<div id="addNoticeModal" class="modal">
    <div class="modal-content">
        <h3>Create New Notice</h3>
        <input type="text" id="noticeTitle" class="modal-input" placeholder="Enter
notice title">
        <textarea id="noticeText" class="modal-input" rows="4" placeholder="Enter

```

```

notice content"></textarea>
    <div class="modal-actions">
        <button class="modal-btn secondary"
onclick="closeModal('addNoticeModal')">Cancel</button>
        <button class="modal-btn primary" id="submitNoticeBtn">Publish
Notice</button>
    </div>
</div>
</div>

<script>
// Theme Management
const body = document.body;
const themeToggle = document.getElementById('themeToggle');
const themeIcon = document.getElementById('themeIcon');
const savedTheme = localStorage.getItem('theme') || 'dark';
body.setAttribute('data-theme', savedTheme);
themeIcon.className = savedTheme === 'light' ? 'fas fa-sun' : 'fas fa-moon';

themeToggle.addEventListener('click', () => {
    const currentTheme = body.getAttribute('data-theme');
    const newTheme = currentTheme === 'light' ? 'dark' : 'light';
    body.setAttribute('data-theme', newTheme);
    themeIcon.className = newTheme === 'light' ? 'fas fa-sun' : 'fas fa-moon';
    localStorage.setItem('theme', newTheme);
});

// Notice Management
const ADMIN_PASSWORD = 'admin123';
let currentAction = null;
let notices = JSON.parse(localStorage.getItem('notices')) || [
    {
        id: Date.now(),
        title: "Scheduled Maintenance",
        content: "Saturday 10:00 AM - 12:00 PM. Elevators will be temporarily
unavailable."
    }
];

// Modal Functions
function showModal(modalId) {
    document.getElementById(modalId).style.display = 'flex';
}

function closeModal(modalId) {
    document.getElementById(modalId).style.display = 'none';
}

// Password Verification
document.getElementById('verifyPasswordBtn').addEventListener('click', () => {
    const password = document.getElementById('adminPassword').value;
    if (password === ADMIN_PASSWORD) {
        closeModal('passwordModal');
    }
});

```

```

        if (currentAction === 'add') showModal('addNoticeModal');
        if (currentAction === 'remove') enableDeleteMode();
    } else {
        alert('Incorrect admin password!');
    }
    document.getElementById('adminPassword').value = "";
});

// Notice Creation
document.getElementById('addNoticeBtn').addEventListener('click', () => {
    currentAction = 'add';
    showModal('passwordModal');
});

document.getElementById('submitNoticeBtn').addEventListener('click', () => {
    const title = document.getElementById('noticeTitle').value;
    const content = document.getElementById('noticeText').value;
    if (title && content) {
        notices.push({
            id: Date.now(),
            title: title,
            content: content
        });
        localStorage.setItem('notices', JSON.stringify(notices));
        renderNotices();
        closeModal('addNoticeModal');
        document.getElementById('noticeTitle').value = "";
        document.getElementById('noticeText').value = "";
    }
});

// Notice Deletion
document.getElementById('removeNoticeBtn').addEventListener('click', () => {
    currentAction = 'remove';
    showModal('passwordModal');
});

function enableDeleteMode() {
    document.querySelectorAll('.notice').forEach(notice => {
        const deleteBtn = document.createElement('button');
        deleteBtn.className = 'delete-btn';
        deleteBtn.innerHTML = '<i class="fas fa-trash"></i>';
        deleteBtn.onclick = () => {
            if (confirm('Are you sure you want to delete this notice?')) {
                const noticeId = parseInt(notice.dataset.id);
                notices = notices.filter(n => n.id !== noticeId);
                localStorage.setItem('notices', JSON.stringify(notices));
                renderNotices();
            }
        };
        notice.appendChild(deleteBtn);
    });
}

```

```

// Notice Rendering
function renderNotices() {
  const container = document.getElementById('noticesContainer');
  container.innerHTML = "";

  notices.forEach(notice => {
    const noticeEl = document.createElement('div');
    noticeEl.className = 'notice';
    noticeEl.dataset.id = notice.id;
    noticeEl.innerHTML = `
      <i class="fas fa-exclamation-circle"></i>
      <div class="notice-content">
        <div class="notice-title">${notice.title}</div>
        ${notice.content}
      </div>
    `;
    container.appendChild(noticeEl);
  });
}

// Initial Render
renderNotices();

// Close Modals on Outside Click
window.onclick = function(event) {
  if (event.target.className === 'modal') {
    event.target.style.display = 'none';
  }
};
</script>

<script>
// Check user role on page load
document.addEventListener("DOMContentLoaded", async () => {
  const userData = sessionStorage.getItem("user");
  console.log(userData);
  if (!userData) {
    window.location.href = "index.html";
    return;
  }

  const user = JSON.parse(userData);
  console.log(user);
  // Hide admin buttons if not admin
  if (user.username !== "admin") {
    document.getElementById("addNoticeBtn").style.display = "none";
    document.getElementById("removeNoticeBtn").style.display = "none";
  }

  // Load notices from server
  await loadNotices();

```

```

// Add event listeners for add and remove buttons
document.getElementById("addNoticeBtn").addEventListener("click", () => {
  document.getElementById("addNoticeModal").style.display = "flex";
});

document.getElementById("removeNoticeBtn").addEventListener("click", ()
=> {
  enableDeleteMode();
});
});

async function loadNotices() {
  try {
    const response = await fetch("http://localhost:5001/notices");
    const notices = await response.json();

    const container = document.getElementById("noticesContainer");
    container.innerHTML = "";

    notices.forEach(notice => {
      const noticeEl = document.createElement("div");
      noticeEl.className = "notice";
      noticeEl.dataset.id = notice.id;
      noticeEl.innerHTML = `
        <i class="fas fa-exclamation-circle"></i>
        <div class="notice-content">
          <div class="notice-title">${notice.title}</div>
          ${notice.content}
        </div>
      `;
      container.appendChild(noticeEl);
    });
  } catch (error) {
    console.error("Error loading notices:", error);
  }
}

function closeModal(modalId) {
  document.getElementById(modalId).style.display = "none";
}

// Update your existing modal functions to use the server endpoints:

document.getElementById("submitNoticeBtn").addEventListener("click", async
() => {
  const title = document.getElementById("noticeTitle").value;
  const content = document.getElementById("noticeText").value;
  const userData = sessionStorage.getItem("user");
  if (!userData) {
    alert("User not logged in");
    return;
  }
}

```



```

const user = JSON.parse(userData);

if (title && content) {
  try {
    const response = await fetch("http://localhost:5001/add-notice", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ username: user.username, title, content })
    });

    const data = await response.json();

    if (data.success) {
      await loadNotices();
      closeModal("addNoticeModal");
      document.getElementById("noticeTitle").value = "";
      document.getElementById("noticeText").value = "";
    } else {
      alert("Failed to add notice");
    }
  } catch (error) {
    console.error("Error adding notice:", error);
    alert("Error adding notice");
  }
}

// Update delete functionality
async function enableDeleteMode() {
  const notices = document.querySelectorAll(".notice");
  const userData = sessionStorage.getItem("user");
  if (!userData) {
    alert("User not logged in");
    return;
  }
  const user = JSON.parse(userData);

  notices.forEach(notice => {
    const deleteBtn = document.createElement("button");
    deleteBtn.className = "delete-btn";
    deleteBtn.innerHTML = '<i class="fas fa-trash"></i>';

    deleteBtn.onclick = async () => {
      if (confirm("Are you sure you want to delete this notice?")) {
        const noticeId = notice.dataset.id;

        try {
          const response = await fetch(`http://localhost:5001/remove-notice/${noticeId}`, {
            method: "DELETE",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ username: user.username })
          });

```

```

        const data = await response.json();

        if (data.success) {
            await loadNotices();
        } else {
            alert("Failed to delete notice");
        }
    } catch (error) {
        console.error("Error deleting notice:", error);
        alert("Error deleting notice");
    }
}
};

notice.appendChild(deleteBtn);
});
}
</script>

<script>
// Fetch ticket counts and update dashboard
async function updateTicketCounts() {
    try {
        const response = await fetch("http://localhost:5001/ticket-counts");
        if (!response.ok) {
            console.error("Failed to fetch ticket counts");
            return;
        }
        const counts = await response.json();
        document.getElementById("openTicketsCount").textContent = counts.open
|| 0;
        document.getElementById("resolvedTicketsCount").textContent =
counts.resolved || 0;
    } catch (error) {
        console.error("Error fetching ticket counts:", error);
    }
}

document.addEventListener("DOMContentLoaded", () => {
    updateTicketCounts();
});
</script>

<script>
// Fetch total visitors count and update dashboard
async function updateVisitorsCount() {
    try {
        const response = await fetch("http://localhost:5001/visitors");
        if (!response.ok) {
            console.error("Failed to fetch visitors");
            return;
        }
    }

```

```

        const visitors = await response.json();
        document.getElementById("totalVisitorsCount").textContent =
visitors.length || 0;
    } catch (error) {
        console.error("Error fetching visitors:", error);
    }
}

document.addEventListener("DOMContentLoaded", () => {
    updateVisitorsCount();
});
</script>

<script>
// Fetch last 3 deliveries and update dashboard
async function updateDeliveries() {
    try {
        const response = await fetch("http://localhost:5001/deliveries");
        if (!response.ok) {
            console.error("Failed to fetch deliveries");
            return;
        }
        const deliveries = await response.json();
        const deliveryList = document.getElementById("deliveryList");
        deliveryList.innerHTML = "";
        // Sort deliveries by id or date descending to get last 3
        const last3 = deliveries.slice(-3).reverse();
        last3.forEach(delivery => {
            const li = document.createElement("li");
            li.className = "delivery-item";
            li.innerHTML = `
                <span>#${delivery.id}  ${delivery.recipient_name}  from
${delivery.package_details}</span>
                `;
            deliveryList.appendChild(li);
        });
    } catch (error) {
        console.error("Error fetching deliveries:", error);
    }
}

document.addEventListener("DOMContentLoaded", () => {
    updateDeliveries();
});
</script>
</body>
</html>

```

8.2.3 Profile

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Profile - GateWise</title>
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;500;600;700
&display=swap" rel="stylesheet">
  <style>
    /* Previous CSS remains the same */
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
      font-family: 'Poppins', sans-serif;
    }

    :root {
      --primary-color: #60a5fa;
      --secondary-color: #3b82f6;
      --accent-color: #ef4444;
      --text-primary: #f8fafc;
      --text-secondary: #ffffff;
      --sidebar-bg: #1e293b;
      --main-bg: #0f172a;
      --card-bg: #1e293b;
      --border-color: #334155;
    }

    [data-theme="light"] {
      --primary-color: #3b82f6;
      --secondary-color: #2563eb;
      --accent-color: #ef4444;
      --text-primary: #1e293b;
      --text-secondary: white;
      --sidebar-bg: #0f203d;
      --main-bg: #dfe7ee;
      --card-bg: #ffffff;
      --border-color: #e2e8f0;
    }

    body {
      background: var(--main-bg);
      color: var(--text-primary);
      transition: background 0.3s ease, color 0.3s ease;
    }

    .container {

```

```
    display: grid;
    grid-template-columns: 280px 1fr;
    min-height: 100vh;
    gap: 20px;
  }

/* Unified Sidebar Styling */
.sidebar {
  background: var(--sidebar-bg);
  padding: 25px;
  position: relative;
  z-index: 2;
  border-right: 1px solid var(--border-color);
  display: flex;
  flex-direction: column;
  gap: 12px;
}

.logo {
  display: flex;
  align-items: center;
  gap: 12px;
  margin-bottom: 20px;
  padding: 15px;
  color: var(--text-secondary);
  font-weight: 700;
  font-size: 1.5rem;
}

.menu-item {
  display: flex;
  align-items: center;
  padding: 14px 20px;
  border-radius: 10px;
  cursor: pointer;
  transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
  color: var(--text-secondary);
  position: relative;
  font-size: 1rem;
}

.menu-item:hover {
  background: rgba(96,165,250,0.1);
  color: var(--primary-color);
  transform: translateX(8px);
}

.menu-item.active {
  background: rgba(96,165,250,0.2);
  color: var(--primary-color);
  font-weight: 500;
}
```

```
.menu-item.active::before {
  content: "";
  position: absolute;
  left: -25px;
  top: 0;
  height: 100%;
  width: 4px;
  background: var(--primary-color);
  border-radius: 2px;
}

.icon {
  width: 24px;
  text-align: center;
  margin-right: 15px;
  font-size: 1.1rem;
}

.main-content {
  padding: 30px 40px;
}

/* Profile Page Specific Styles */
.profile-section {
  max-width: 600px;
  margin: 0 auto;
}

.profile-form {
  background: var(--card-bg);
  padding: 25px;
  border-radius: 16px;
  border: 1px solid var(--border-color);
}

.form-group {
  margin-bottom: 1.5rem;
}

.form-label {
  display: block;
  margin-bottom: 0.5rem;
  color: var(--text-primary);
  font-weight: 500;
}

.form-input {
  width: 100%;
  padding: 0.75rem;
  border: 1px solid var(--border-color);
  border-radius: 8px;
  background: var(--main-bg);
  color: var(--text-primary);
}
```

```
        transition: all 0.3s ease;
    }

    .btn-primary {
        background: var(--primary-color);
        color: white;
        padding: 12px 24px;
        border: none;
        border-radius: 8px;
        cursor: pointer;
        transition: all 0.3s ease;
        font-weight: 500;
        width: 100%;
        margin-top: 15px;
    }

    .btn-primary:hover {
        background: var(--secondary-color);
    }

    /* Header Styling */
    .header {
        display: flex;
        justify-content: space-between;
        align-items: center;
        margin-bottom: 30px;
        gap: 20px;
    }

    .header h1 {
        font-size: 1.8rem;
        font-weight: 600;
        white-space: nowrap;
    }

    .header-controls {
        display: flex;
        gap: 15px;
        align-items: center;
        height: 40px;
    }

    .profile-btn {
        display: flex;
        align-items: center;
        justify-content: center;
        width: 40px;
        height: 40px;
        border-radius: 50%;
        background: var(--card-bg);
        border: 1px solid var(--border-color);
        cursor: pointer;
        transition: all 0.3s ease;
    }
```

```

    }

    .profile-icon {
        width: 32px;
        height: 32px;
        border-radius: 50%;
        background: var(--primary-color);
        display: flex;
        align-items: center;
        justify-content: center;
    }

    .theme-toggle {
        cursor: pointer;
        width: 40px;
        height: 40px;
        border-radius: 50%;
        background: var(--card-bg);
        border: 1px solid var(--border-color);
        display: flex;
        align-items: center;
        justify-content: center;
    }

    @media (max-width: 768px) {
        .container {
            grid-template-columns: 1fr;
        }
        .sidebar {
            display: none;
        }
        .profile-form {
            padding: 20px;
        }
    }
</style>
</head>
<body>
<div class="container">
<aside class="sidebar">
<div class="logo">
    <i class="fas fa-building"></i>
    GateWise
</div>
<div class="menu-item" onclick="window.location.href='dashboard.html';">
    <i class="fas fa-home icon"></i>
    Dashboard
</div>
<div class="menu-item" onclick="window.location.href='help.html';">
    <i class="fas fa-headset icon"></i>
    Helpdesk
</div>
<div class="menu-item" onclick="window.location.href='staff.html';">

```



```

        <i class="fas fa-users icon"></i>
        Staff Management
    </div>
    <div class="menu-item" onclick="window.location.href='visitor.html';">
        <i class="fas fa-user-check icon"></i>
        Visitor Management
    </div>
    <div class="menu-item" onclick="window.location.href='vendor.html';">
        <i class="fas fa-truck icon"></i>
        Vendor Services
    </div>
    <div class="menu-item" onclick="window.location.href='delivery.html';">
        <i class="fas fa-box icon"></i>
        Delivery Management
    </div>
    <div class="menu-item" onclick="window.location.href='directory.html';">
        <i class="fas fa-address-book icon"></i>
        Directory
    </div>
    <div class="menu-item" onclick="window.location.href='alerts.html';">
        <i class="fas fa-bell icon"></i>
        Alerts
    </div>
    <div class="menu-item active">
        <i class="fas fa-sign-out-alt icon"></i>
        Log Out
    </div>
</aside>

<main class="main-content">
    <div class="header">
        <h1>Profile Settings</h1>
        <div class="header-controls">
            <a href="profile.html">
                <div class="profile-btn">
                    <div class="profile-icon">
                        <i class="fas fa-user text-white"></i>
                    </div>
                </div>
            </a>
            <div class="theme-toggle" id="themeToggle">
                <i class="fas fa-moon" id="themeIcon"></i>
            </div>
        </div>
    </div>

    <div class="profile-section">
        <form class="profile-form" id="profileForm">
            <div class="form-group">
                <label class="form-label">Username</label>
                <input type="text" class="form-input" id="username" required readonly>
            </div>
            <div class="form-group">

```

```

        <label class="form-label">Current Password</label>
        <input type="password" class="form-input" id="currentPassword"
required>
    </div>
    <div class="form-group">
        <label class="form-label">New Password</label>
        <input type="password" class="form-input" id="newPassword"
required>
    </div>
    <div class="form-group">
        <label class="form-label">Confirm New Password</label>
        <input type="password" class="form-input" id="confirmPassword"
required>
    </div>
    <button type="submit" class="btn-primary">Update Profile</button>
</form>
</div>
</main>
</div>

<script>
    // Theme Management
    const themeToggle = document.getElementById('themeToggle');
    const themeIcon = document.getElementById('themeIcon');
    const body = document.body;

    const savedTheme = localStorage.getItem('theme') || 'dark';
    body.setAttribute('data-theme', savedTheme);
    themeIcon.className = savedTheme === 'light' ? 'fas fa-sun' : 'fas fa-moon';

    themeToggle.addEventListener('click', () => {
        const currentTheme = body.getAttribute('data-theme');
        const newTheme = currentTheme === 'light' ? 'dark' : 'light';

        body.setAttribute('data-theme', newTheme);
        themeIcon.className = newTheme === 'light' ? 'fas fa-sun' : 'fas fa-moon';
        localStorage.setItem('theme', newTheme);
    });

    // Fetch Username from Session Storage
    function fetchUsername() {
        const user = JSON.parse(sessionStorage.getItem('user'));
        if (user) {
            document.getElementById('username').value = user.username;
        } else {
            // Redirect to login page if no user is found in session
            window.location.href = 'index.html';
        }
    }

    // Profile Form Handling
    document.getElementById('profileForm').addEventListener('submit', async
function(e) {

```

```
e.preventDefault();

const currentPassword = document.getElementById('currentPassword').value;
const newPassword = document.getElementById('newPassword').value;
const confirmPassword = document.getElementById('confirmPassword').value;

if (newPassword !== confirmPassword) {
    alert('New passwords do not match!');
    return;
}

if (newPassword.length < 8) {
    alert('Password must be at least 8 characters!');
    return;
}

try {
    const user = JSON.parse(sessionStorage.getItem('user'));
    const response = await fetch('http://localhost:5001/update-password', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            username: user.username,
            currentPassword,
            newPassword
        })
    });
    const data = await response.json();
    alert(data.message || 'Password updated successfully!');
    if (data.success) this.reset();
} catch (error) {
    console.error('Error updating password:', error);
}

// Fetch and display username when the page loads
fetchUsername();
</script>
</body>
</html>
```

8.2.4 Server

```
require("dotenv").config();
const express = require("express");
const mysql = require("mysql");
const session = require("express-session");
const bodyParser = require("body-parser");
const cors = require('cors');
const PORT = 5001;

const app = express();
app.use(cors()); // Correctly placed after app initialization

// Serve static files
// app.use(express.static(__dirname));
app.use(express.static(__dirname));
app.use(express.json());

app.use(cors());
app.use(bodyParser.json());
app.use(
  session({
    secret: "secret_key",
    resave: false,
    saveUninitialized: true,
    cookie: { secure: false } // Change to true if using HTTPS
  })
);

// Database Connection
const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "root",
  database: "gatewise_db"
});

db.connect((err) => {
  if (err) {
    console.error("Database connection error:", err);
  } else {
    console.log("MySQL Connected...");
  }
});

// Register User (Plain Text Password)
app.post("/register", (req, res) => {
  const { username, password } = req.body;

  db.query("INSERT INTO users (username, password) VALUES (?, ?)", [username, password], (err) => {
    if (err) return res.status(500).json({ message: "User already exists or database error" });
  });
});
```

```
    res.json({ success: true, message: "User registered successfully" });
  });
});

// Login User (Plain Text Password)
app.post("/login", (req, res) => {
  const { username, password } = req.body;

  db.query("SELECT * FROM users WHERE username = ?", [username], (err, results)
=> {
    if (err) return res.status(500).json({ message: "Database error" });

    if (results.length === 0) {
      return res.status(401).json({ message: "Invalid credentials" });
    }

    const user = results[0];

    if (password !== user.password) {
      return res.status(401).json({ message: "Invalid credentials" });
    }

    // Include role
    req.session.user = { id: user.id, username: user.username, role: user.role };

    // Send full user details to client
    res.json({
      success: true,
      message: "Login successful",
      user: { id: user.id, username: user.username, role: user.role }
    });
  });
});

// Update Password (No Server Session)
app.post("/update-password", (req, res) => {
  const { username, currentPassword, newPassword } = req.body;

  db.query("SELECT password FROM users WHERE username = ?", [username],
(err, results) => {
    if (err) return res.status(500).json({ success: false, message: "Database error" });

    if (results.length === 0) return res.status(400).json({ success: false, message: "User
not found" });

    if (currentPassword !== results[0].password) {
      return res.status(401).json({ success: false, message: "Current password is
incorrect" });
    }

    db.query("UPDATE users SET password = ? WHERE username = ?",
[newPassword, username], (err) => {
      if (err) return res.status(500).json({ success: false, message: "Error updating
```

```
password" });

    res.json({ success: true, message: "Password updated successfully" });
  });
});

app.get("/staff", (req, res) => {
  db.query("SELECT * FROM staff", (err, results) => {
    if (err) {
      console.error("Error fetching staff:", err);
      return res.status(500).json({ error: "Failed to fetch staff data" });
    }
    // console.log("Fetched staff data:", results);

    res.json(results);
  });
});

// Add Staff Endpoint
app.post("/add-staff", (req, res) => {
  const { name, email, phone, role } = req.body;

  // Validate input
  if (!name || !email || !phone || !role) {
    return res.status(400).json({ message: "All fields are required" });
  }

  // Insert into the database
  const query = "INSERT INTO staff (name, email, phone, role) VALUES (?, ?, ?, ?)";
  db.query(query, [name, email, phone, role], (err, results) => {
    if (err) {
      console.error("Error adding staff:", err);
      return res.status(500).json({ message: "Failed to add staff member" });
    }

    res.json({ message: "Staff member added successfully", id: results.insertId });
  });
});

app.delete("/remove-staff/:id", (req, res) => {
  const staffId = req.params.id;

  db.query("DELETE FROM staff WHERE id = ?", [staffId], (err, results) => {
    if (err) {
      console.error("Error removing staff:", err);
      return res.status(500).json({ message: "Error removing staff" });
    }

    if (results.affectedRows === 0) {
      return res.status(404).json({ message: "Staff member not found" });
    }
  });
});
```

```
    res.json({ message: "Staff member removed successfully" });
  });
});

// Submit a New Ticket
app.post("/submit-ticket", (req, res) => {
  const { subject, description, priority } = req.body;
  const status = "Open"; // Default status for new tickets

  const query = "INSERT INTO tickets (subject, description, priority, status,
created_at) VALUES (?, ?, ?, ?, NOW())";
  db.query(query, [subject, description, priority, status], (err, result) => {
    if (err) {
      console.error("Error inserting ticket: ", err);
      return res.status(500).json({ message: "Error submitting ticket" });
    }
    res.json({ message: "Ticket submitted successfully!" });
  });
});

// Get Ticket Counts (Open, In Progress, Resolved)
app.get("/ticket-counts", (req, res) => {
  const query = "SELECT status, COUNT(*) AS count FROM tickets GROUP BY
status";
  db.query(query, (err, results) => {
    if (err) {
      console.error("Error fetching ticket counts: ", err);
      return res.status(500).json({ message: "Error fetching counts" });
    }
    const counts = { open: 0, in_progress: 0, resolved: 0 };
    results.forEach(row => {
      if (row.status === "Open") counts.open = row.count;
      if (row.status === "In Progress") counts.in_progress = row.count;
      if (row.status === "Resolved") counts.resolved = row.count;
    });
    res.json(counts);
  });
});

// Get Recent Tickets
app.get("/recent-tickets", (req, res) => {
  const query = "SELECT * FROM tickets ORDER BY created_at DESC LIMIT 10";
  db.query(query, (err, results) => {
    if (err) {
      console.error("Error fetching tickets: ", err);
      return res.status(500).json({ message: "Error fetching tickets" });
    }
    res.json(results);
  });
});

// Admin: Update Ticket Status
app.put("/update-ticket/:id", (req, res) => {
```

```
const { id } = req.params;
const { status } = req.body;

const validStatuses = ["Open", "In Progress", "Resolved"];
if (!validStatuses.includes(status)) {
  return res.status(400).json({ message: "Invalid status" });
}

const query = "UPDATE tickets SET status = ? WHERE id = ?";
db.query(query, [status, id], (err, result) => {
  if (err) {
    console.error("Error updating ticket: ", err);
    return res.status(500).json({ message: "Error updating ticket" });
  }
  res.json({ message: "Ticket status updated successfully!" });
});
// Remove Ticket
app.delete("/remove-ticket/:id", (req, res) => {
  const { id } = req.params;

  const query = "DELETE FROM tickets WHERE id = ?";
  db.query(query, [id], (err, result) => {
    if (err) {
      console.error("Error removing ticket: ", err);
      return res.status(500).json({ message: "Error removing ticket" });
    }

    if (result.affectedRows === 0) {
      return res.status(404).json({ message: "Ticket not found" });
    }

    res.json({ message: "Ticket removed successfully!" });
  });
});
// Visitor Management Endpoints - Updated with better error handling

// Get all visitors
app.get("/visitors", (req, res) => {
  db.query("SELECT * FROM visitors ORDER BY visit_date DESC", (err, results)
=> {
    if (err) {
      console.error("Database error fetching visitors:", err);
      return res.status(500).json({
        error: "Failed to fetch visitors",
        details: err.message
      });
    }
    res.setHeader('Content-Type', 'application/json');
    res.json(results);
  });
});
```



```

// Add new visitor with enhanced validation
app.post("/add-visitor", (req, res) => {
  const { name, phone, visit_date, meeting_with, purpose } = req.body;

  // Validate input
  if (!name || !phone || !visit_date || !meeting_with || !purpose) {
    console.log("Validation failed - missing fields");
    return res.status(400).json({
      message: "All fields are required",
      received: req.body
    });
  }

  // Phone number validation
  if (!/^\d{10,15}$/.test(phone)) {
    console.log("Validation failed - invalid phone:", phone);
    return res.status(400).json({
      message: "Phone number must be 10-15 digits"
    });
  }

  try {
    // Convert visit_date to MySQL compatible format
    const formattedDate = new Date(visit_date).toISOString().slice(0, 19).replace('T',
    ');

    const query = "INSERT INTO visitors (name, phone, visit_date, meeting_with,
    purpose) VALUES (?, ?, ?, ?, ?)";
    db.query(query, [name, phone, formattedDate, meeting_with, purpose], (err,
    results) => {
      if (err) {
        console.error("Database error adding visitor:", err);
        return res.status(500).json({
          message: "Failed to add visitor",
          error: err.message,
          sql: err.sql
        });
      }

      console.log("Visitor added successfully:", results.insertId);
      res.json({
        success: true,
        message: "Visitor added successfully",
        visitorId: results.insertId
      });
    } catch (error) {
      console.error("Error processing visitor data:", error);
      res.status(500).json({
        message: "Error processing visitor data",
        error: error.message
      });
    }
  }

```

```
});

// Remove visitor
app.delete("/remove-visitor/:id", (req, res) => {
  const visitorId = req.params.id;

  if (!visitorId || isNaN(visitorId)) {
    return res.status(400).json({
      message: "Invalid visitor ID",
      received: visitorId
    });
  }

  db.query("DELETE FROM visitors WHERE id = ?", [visitorId], (err, results) => {
    if (err) {
      console.error("Database error removing visitor:", err);
      return res.status(500).json({
        message: "Error removing visitor",
        error: err.message
      });
    }

    if (results.affectedRows === 0) {
      return res.status(404).json({
        message: "Visitor not found"
      });
    }

    res.json({
      success: true,
      message: "Visitor removed successfully"
    });
  });
});

// Get all deliveries
app.get("/deliveries", (req, res) => {
  db.query("SELECT * FROM deliveries", (err, results) => {
    if (err) {
      console.error("Error fetching deliveries:", err);
      return res.status(500).json({ error: "Failed to fetch deliveries" });
    }
    res.json(results);
  });
});

// Add new delivery
app.post("/add-delivery", (req, res) => {
  const { recipient_name, package_details, expected_arrival } = req.body;

  if (!recipient_name || !package_details || !expected_arrival) {
    return res.status(400).json({ message: "All fields are required" });
  }
});
```

```

    const query = "INSERT INTO deliveries (recipient_name, package_details,
expected_arrival) VALUES (?, ?, ?)";
    db.query(query, [recipient_name, package_details, expected_arrival], (err, results)
=> {
        if (err) {
            console.error("Error adding delivery:", err);
            return res.status(500).json({ message: "Failed to add delivery" });
        }
        res.json({
            message: "Delivery added successfully",
            delivery: {
                id: results.insertId,
                recipient_name,
                package_details,
                expected_arrival
            }
        });
    });
});

// Remove delivery
app.delete("/remove-delivery/:id", (req, res) => {
    const deliveryId = req.params.id;

    db.query("DELETE FROM deliveries WHERE id = ?", [deliveryId], (err, results) =>
    {
        if (err) {
            console.error("Error removing delivery:", err);
            return res.status(500).json({ message: "Error removing delivery" });
        }

        if (results.affectedRows === 0) {
            return res.status(404).json({ message: "Delivery not found" });
        }

        res.json({ message: "Delivery removed successfully" });
    });
});

// Get all directory contacts
app.get("/directory-contacts", (req, res) => {
    db.query("SELECT * FROM directory_contacts", (err, results) => {
        if (err) {
            console.error("Error fetching contacts:", err);
            return res.status(500).json({ message: "Failed to fetch contacts" });
        }
        res.json(results);
    });
});

// Add a new directory contact
app.post("/add-directory-contact", (req, res) => {
    const { name, role, phone, email } = req.body;

```

```

    if (!name || !role || !phone || !email) {
      return res.status(400).json({ message: "All fields are required" });
    }

    db.query(
      "INSERT INTO directory_contacts (name, role, phone, email) VALUES (?, ?, ?, ?)",
      [name, role, phone, email],
      (err, result) => {
        if (err) {
          console.error("Error adding contact:", err);
          return res.status(500).json({ message: "Failed to add contact" });
        }
        res.json({ message: "Contact added successfully!" });
      }
    );
  });

// Remove a contact
app.delete("/remove-directory-contact/:id", (req, res) => {
  const id = req.params.id;
  db.query("DELETE FROM directory_contacts WHERE id = ?", [id], (err, result) => {
    if (err) {
      console.error("Error deleting contact:", err);
      return res.status(500).json({ message: "Failed to delete contact" });
    }
    if (result.affectedRows === 0) {
      return res.status(404).json({ message: "Contact not found" });
    }
    res.json({ message: "Contact deleted successfully!" });
  });
});

// Add these property-related endpoints to your existing server.js

// Property Management Endpoints

// Get all properties
app.get("/properties", (req, res) => {
  db.query("SELECT * FROM properties", (err, results) => {
    if (err) {
      console.error("Error fetching properties:", err);
      return res.status(500).json({ error: "Failed to fetch properties" });
    }
    res.json(results);
  });
});

// Add new property
app.post("/add-property", (req, res) => {
  const { propertyType, ownerName, contactNumber, propertyDescription } =
    req.body;

```

```

    if (!propertyType || !ownerName || !contactNumber || !propertyDescription) {
      return res.status(400).json({ message: "All fields are required" });
    }

    const query = "INSERT INTO properties (property_type, owner_name,
    contact_number, description) VALUES (?, ?, ?, ?)";
    db.query(query, [propertyType, ownerName, contactNumber, propertyDescription],
    (err, results) => {
      if (err) {
        console.error("Error adding property:", err);
        return res.status(500).json({ message: "Failed to add property" });
      }
      res.json({
        message: "Property added successfully",
        property: {
          id: results.insertId,
          property_type: propertyType,
          owner_name: ownerName,
          contact_number: contactNumber,
          description: propertyDescription
        }
      });
    });
  });
});

// Remove property
app.delete("/remove-property/:id", (req, res) => {
  const propertyId = req.params.id;

  db.query("DELETE FROM properties WHERE id = ?", [propertyId], (err, results)
  => {
    if (err) {
      console.error("Error removing property:", err);
      return res.status(500).json({ message: "Error removing property" });
    }

    if (results.affectedRows === 0) {
      return res.status(404).json({ message: "Property not found" });
    }

    res.json({ message: "Property removed successfully" });
  });
});

// Get all alerts (sorted by date)
app.get("/alerts", (req, res) => {
  const query = `
    SELECT
      id,
      event_name AS eventName,
      description AS eventDescription,
      DATE_FORMAT(event_date, '%Y-%m-%d') AS eventDate

```

```
FROM alerts
ORDER BY event_date DESC
`;

db.query(query, (err, results) => {
  if (err) {
    console.error("Error fetching alerts:", err);
    return res.status(500).json({ error: "Failed to fetch alerts" });
  }
  res.json(results);
});

app.post("/add-alert", (req, res) => {
  const { eventName, eventDescription, eventDate } = req.body;

  if (!eventName || !eventDescription || !eventDate) {
    return res.status(400).json({ success: false, message: "All fields are required" });
  }

  const query = `
    INSERT INTO alerts
      (event_name, description, event_date)
    VALUES
      (?, ?, ?)
  `;

  db.query(query, [eventName, eventDescription, eventDate], (err, results) => {
    if (err) {
      console.error("Error adding alert:", err);
      return res.status(500).json({
        success: false,
        message: "Failed to add alert",
        error: err.message
      });
    }

    res.json({
      success: true,
      message: "Alert added successfully",
      alertId: results.insertId
    });
  });

  // Remove alert endpoint
  app.delete("/remove-alert/:id", (req, res) => {
    const alertId = req.params.id;

    if (!alertId || isNaN(alertId)) {
      return res.status(400).json({ success: false, message: "Invalid alert ID" });
    }
  });
});
```

```
db.query("DELETE FROM alerts WHERE id = ?", [alertId], (err, results) => {
  if (err) {
    console.error("Error removing alert:", err);
    return res.status(500).json({
      success: false,
      message: "Error removing alert",
      error: err.message
    });
  }

  if (results.affectedRows === 0) {
    return res.status(404).json({ success: false, message: "Alert not found" });
  }

  res.json({
    success: true,
    message: "Alert removed successfully"
  });
});

// Get all notices
app.get("/notices", (req, res) => {
  db.query("SELECT * FROM notices ORDER BY created_at DESC", (err, results)
=> {
    if (err) {
      console.error("Error fetching notices:", err);
      return res.status(500).json({ error: "Failed to fetch notices" });
    }
    res.json(results);
  });
});

// Add new notice (admin only)
app.post("/add-notice", (req, res) => {
  const { username, title, content } = req.body;

  if (!username || !title || !content) {
    return res.status(400).json({ message: "Username, title and content are required"
});
  }

  if (username !== "admin") {
    return res.status(403).json({ message: "Only admin can add notices" });
  }

  const query = "INSERT INTO notices (title, content, created_at) VALUES (?, ?,
NOW())";
  db.query(query, [title, content], (err, results) => {
    if (err) {
      console.error("Error adding notice:", err);
      return res.status(500).json({ message: "Failed to add notice" });
    }
  });
});
```

```
    }
    res.json({
      success: true,
      message: "Notice added successfully",
      noticeId: results.insertId
    });
  });
});

// Remove notice (admin only)
app.delete("/remove-notice/:id", (req, res) => {
  const noticeId = req.params.id;
  const { username } = req.body;

  if (!username) {
    return res.status(400).json({ message: "Username is required" });
  }

  if (username !== "admin") {
    return res.status(403).json({ message: "Only admin can remove notices" });
  }

  db.query("DELETE FROM notices WHERE id = ?", [noticeId], (err, results) => {
    if (err) {
      console.error("Error removing notice:", err);
      return res.status(500).json({ message: "Error removing notice" });
    }

    if (results.affectedRows === 0) {
      return res.status(404).json({ message: "Notice not found" });
    }

    res.json({ success: true, message: "Notice removed successfully" });
  });
});

// Start Server
app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```


Chapter 9: Testing

9.1 User Registration

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|------------------------------|-----------------------------|-------------------------------------|--------|
| TC001 | Register with valid details | Name, Password | "Registered successfully" message | Pass |
| TC002 | Register with existing email | Existing User name | " User name already exists" message | Fail |
| TC003 | Register with missing fields | Blank User name or Password | "All fields required" message | Fail |

Table 9.1 User Registration

9.2 User Login

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|-----------------------------------|----------------------------------|-------------------------------|--------|
| TC004 | Login with correct credentials | Valid User name & Password | Redirect to Dashboard | Pass |
| TC005 | Login with incorrect password | Valid User name & Wrong Password | "Invalid credentials" message | Fail |
| TC006 | Login with unregistered User name | Unregistered User name | "User not found" message | Fail |

Table 9.2 User Login

9.3 Dashboard Access

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|--------------------------------|-----------------------|-------------------------|--------|
| TC007 | Access dashboard after login | Authenticated session | Dashboard content loads | Pass |
| TC008 | Access dashboard without login | No session | Redirect to Login Page | Fail |

Table 9.3 Dashboard

9.4 Helpdesk

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|--|---|--|--------|
| TC009 | Create ticket with valid details | Subject, Description, Priority (Medium) | "Ticket submitted successfully" message | Pass |
| TC010 | Create ticket with missing subject | Blank Subject, Valid Description | "Subject is required" validation message | Fail |
| TC011 | Create ticket with missing description | Valid Subject, Blank Description | "Description is required" validation message | Fail |

Table 9.4 Helpdesk

9.5 Staff

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|--------------------------------------|--------------------------------------|--------------------------------------|--------|
| TC012 | Admin adds new staff with valid data | Name, Email, Phone, Role (Admin) | "Staff added successfully" message | Pass |
| TC013 | Admin adds staff with missing fields | Blank Name, Valid Email/Phone/Role | "All fields are required" message | Fail |
| TC014 | Admin removes staff member | Click "Remove" button + confirmation | "Staff removed successfully" message | Pass |

Table 9.5 Staff

9.6 Vendor Services

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|----------------------------------|---|---|--------|
| TC015 | Add property with valid data | Valid type, owner, contact, description | "Property added successfully" message | Pass |
| TC016 | Add property with missing fields | Blank owner name, other fields valid | Form rejects submission | Fail |
| TC017 | Remove property as admin | Click "Remove" + confirmation | "Property removed successfully" message | Pass |

Table 9.6 Vendor Services

9.7 Visitor

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|--------------------------------|--|--------------------------------------|--------|
| TC018 | Add visitor with valid data | Complete form with valid details | "Visitor added successfully" message | Pass |
| TC019 | Add visitor with missing name | Blank name, other fields valid | Form rejects submission | Fail |
| TC020 | Add visitor with invalid phone | Valid data except invalid phone format | Form rejects submission | Fail |

Table 9.7 Visitor

9.8 Delivery

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|-------------------------------------|--|---------------------------------------|--------|
| TC021 | Add delivery with valid data | Complete form with valid details | "Delivery added successfully" message | Pass |
| TC022 | Add delivery with missing recipient | Blank recipient name, other fields valid | Form rejects submission | Fail |

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|-----------------------------------|--|-------------------------|--------|
| TC023 | Add delivery with missing package | Valid recipient, blank package details | Form submission rejects | Fail |

Table 9.8 Delivery

9.9 Directory

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|------------------------------------|--|--------------------------------------|--------|
| TC024 | Admin adds contact with valid data | Complete form with valid details | "Contact added successfully" message | Pass |
| TC025 | Add contact with missing name | Blank name, other fields valid | Form submission rejects | Fail |
| TC026 | Add contact with invalid email | Valid data except invalid email format | Form submission rejects | Fail |

Table 9.9 Directory

9.10 Alert

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|---------------------------------------|--------------------------------------|--------------------------------------|--------|
| TC027 | Create alert with valid data | Complete form with valid details | "Alert created successfully" message | Pass |
| TC028 | Create alert with missing event name | Blank event name, other fields valid | Form submission rejects | Fail |
| TC029 | Create alert with missing description | Valid event name, blank description | Form submission rejects | Fail |

Table 9.10 Alert

9.11 Profile

| TC ID | Test Case Description | Input | Expected Output | Status |
|-------|--|--|---|--------|
| TC030 | Update password with valid data | Correct current password, matching new passwords | "Password updated successfully" message | Pass |
| TC031 | Update with incorrect current password | Wrong current password, valid new passwords | "Current password is incorrect" error | Fail |
| TC032 | New passwords don't match | Mismatched new/confirm passwords | "Passwords do not match" error | Fail |

Table 9.11 Profile

Chapter 10: User manual

10.1 Installation steps:

10.1.1 Node.js Installation Manual

Pre-requisite

1. Check System Requirements

- OS: Windows, macOS, Linux (Ubuntu/Debian/CentOS)
- RAM: Minimum 2 GB
- Internet Connection (for downloading Node.js)

2. Installation on Windows

Step 1: Download Installer

- Go to the official website: <https://nodejs.org>
- Download the **LTS (Long Term Support)** version for Windows (.msi file)

Step 2: Run Installer

- Double-click the downloaded .msi file
- Follow the installer steps:
 - Accept license
 - Choose install location
 - Ensure the checkbox for Add to PATH is selected

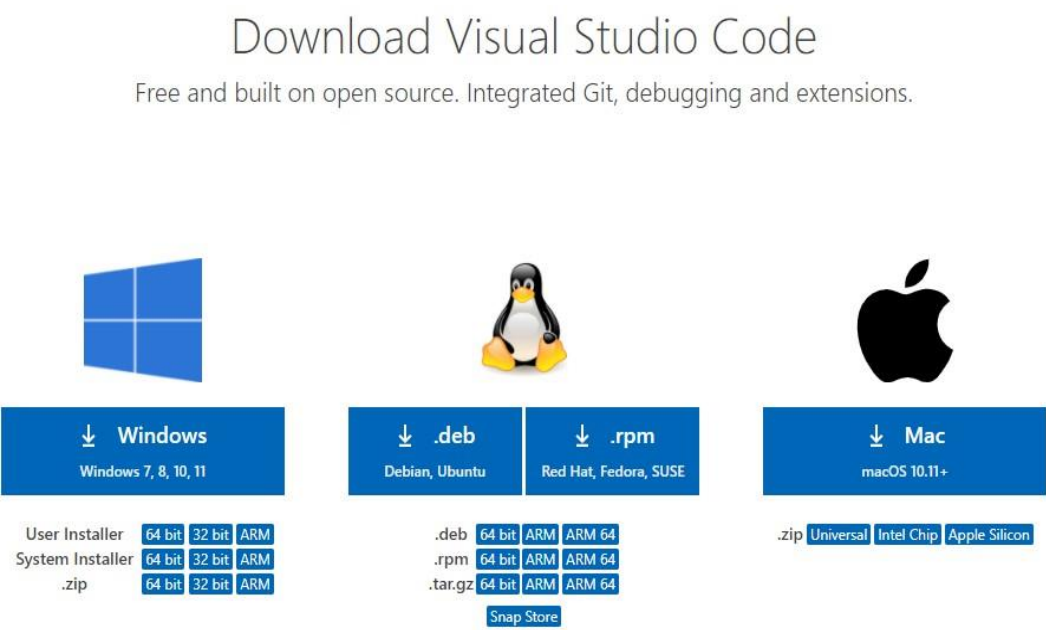
Step 3: Verify Installation

Open **Command Prompt** and run:

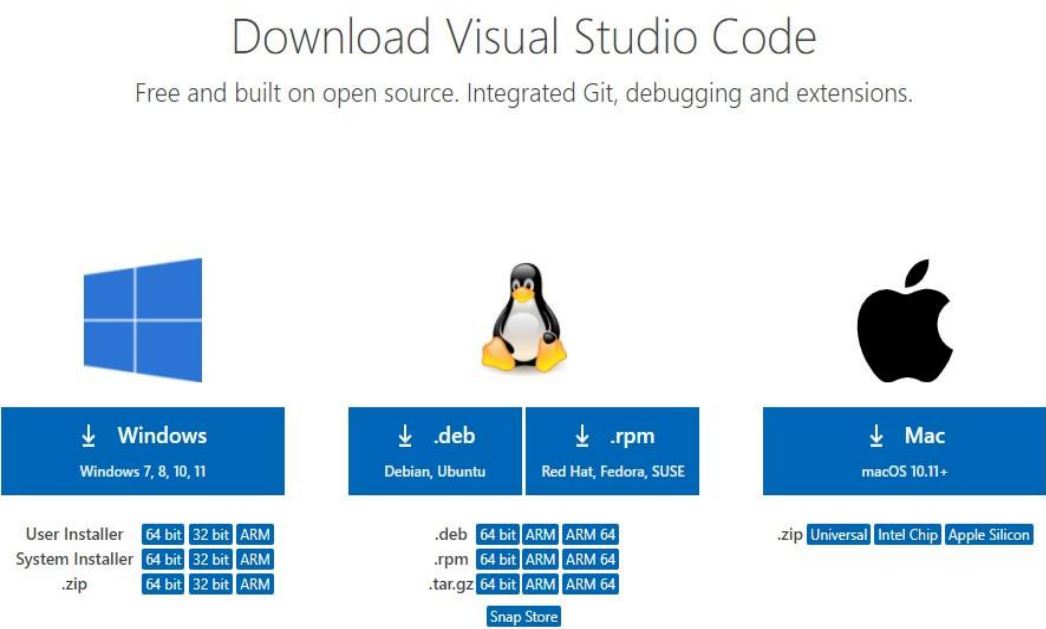
```
PS C:\Users\ASUS\OneDrive\Desktop> node -v
v22.13.1
PS C:\Users\ASUS\OneDrive\Desktop> npm -v
11.1.0
PS C:\Users\ASUS\OneDrive\Desktop> |
```

10.1.2 VS Code Installation Manual

Step 1: Visit the Official Website of the Visual Studio Code using any web browser like [Google Chrome](#), [Microsoft Edge](#), etc.



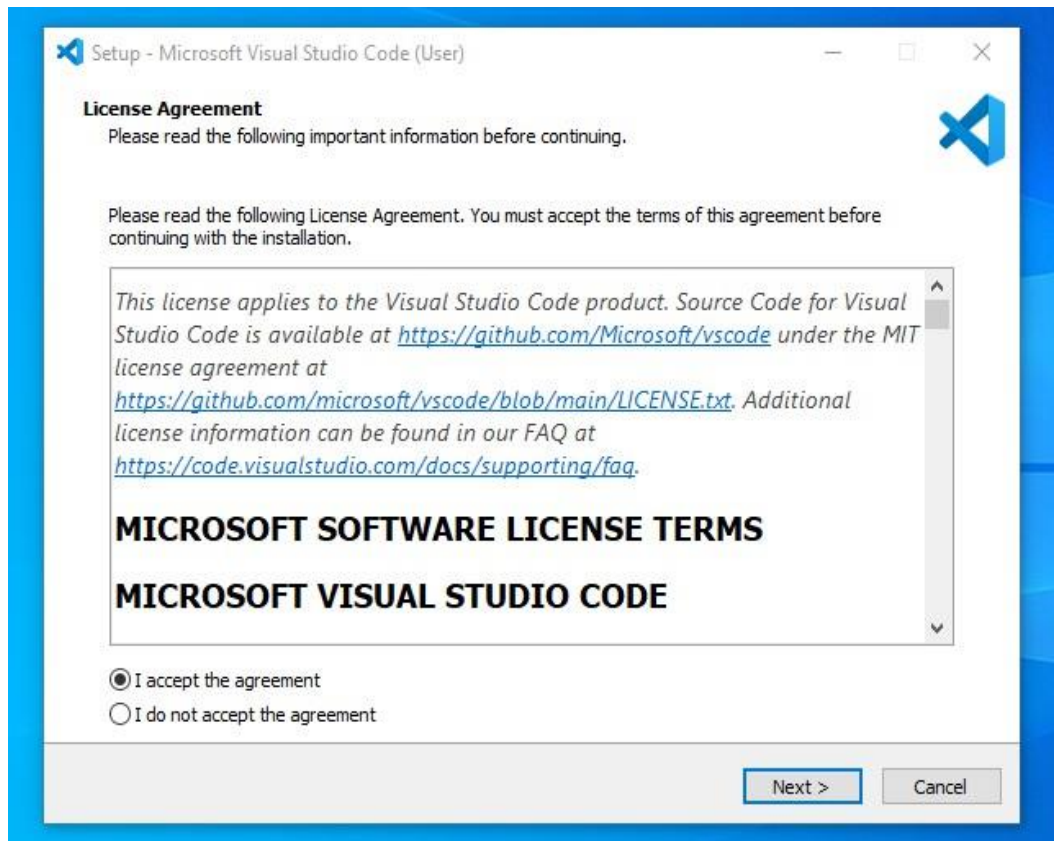
Step 2: Press the "Download for Windows" button on the website to start the download of the Visual Studio Code Application.



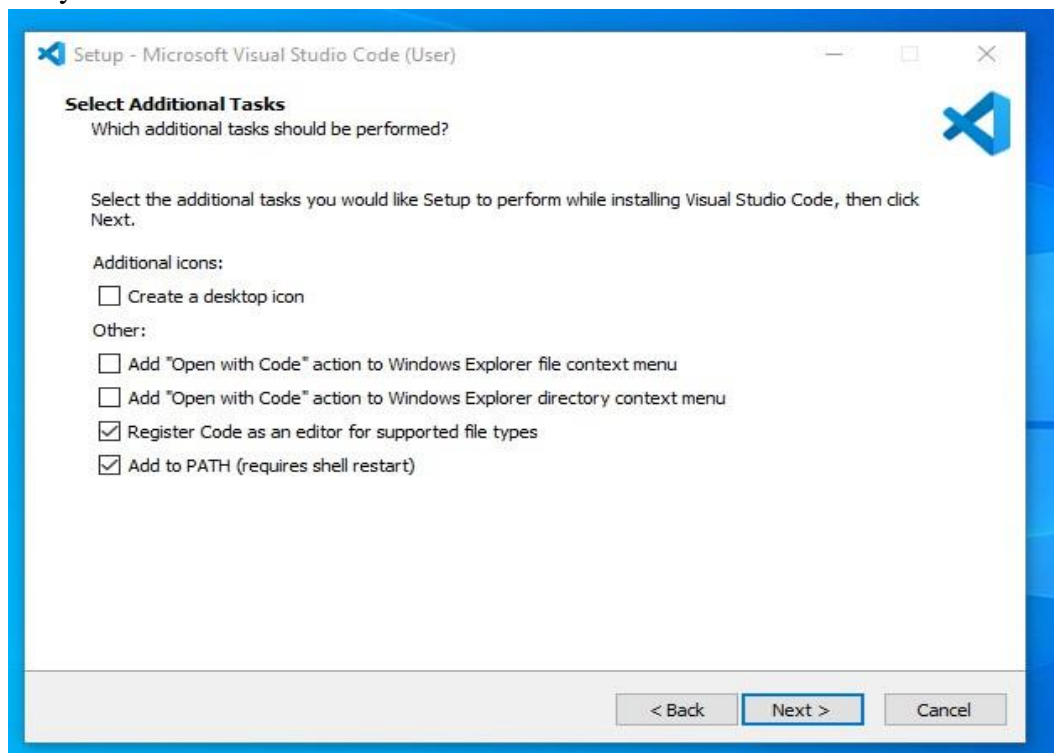
Step 3: When the download finishes, then the Visual Studio Code Icon appears in the downloads folder.

Step 4: Click on the Installer icon to start the installation process of the Visual Studio Code.

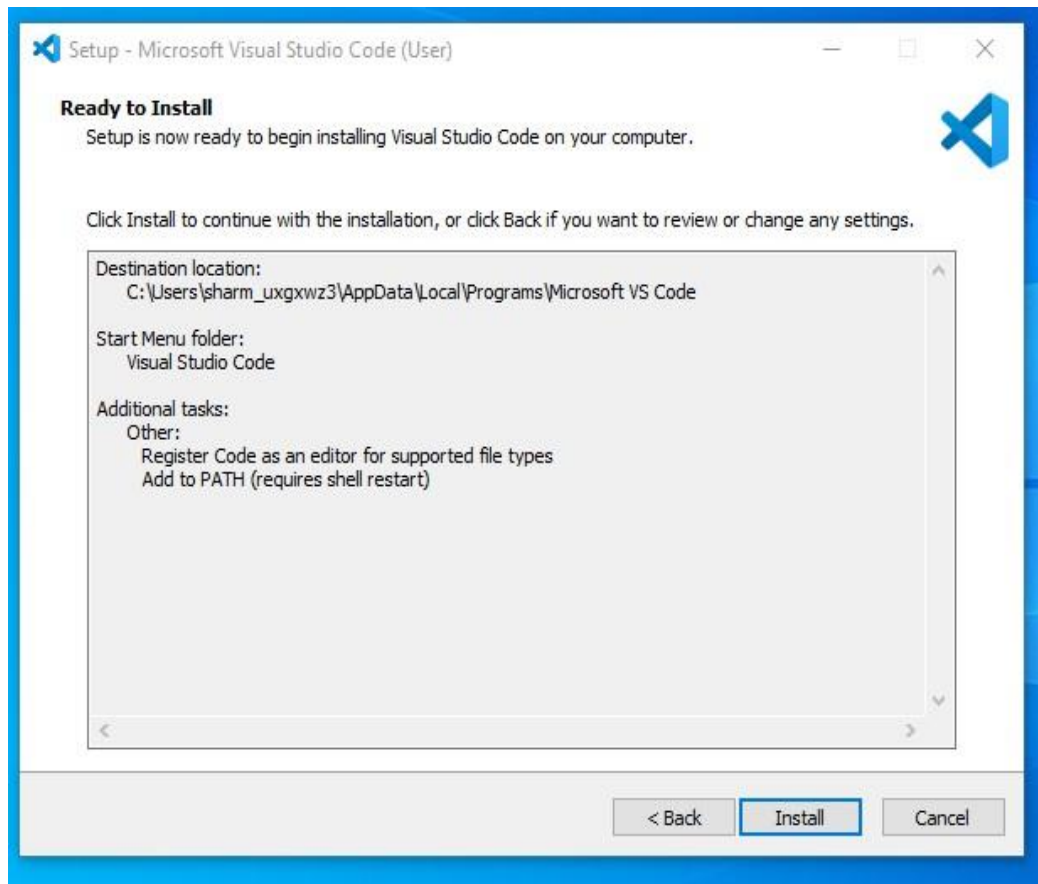
Step 5: After the Installer opens, it will ask you to accept the terms and conditions of the Visual Studio Code. Click on I accept the agreement and then click the Next button.



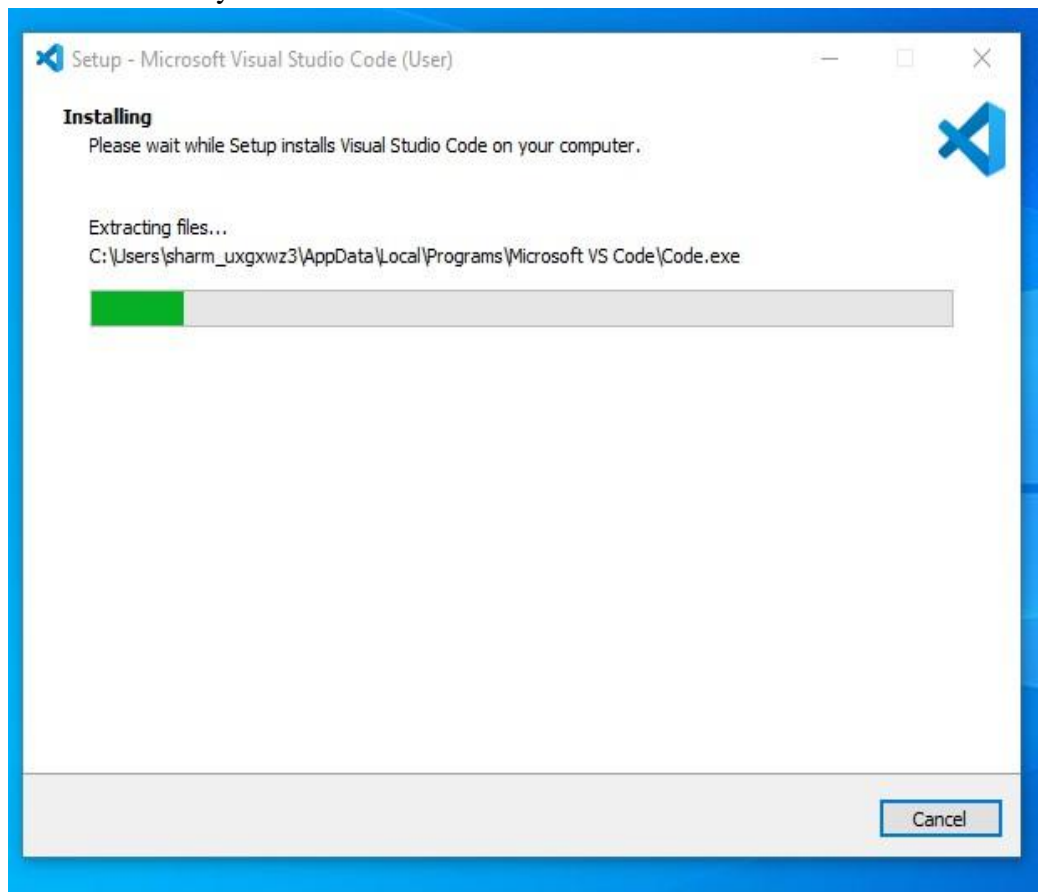
Step 6: Choose the location data for running the Visual Studio Code. It will then ask you to browse the location. Then click on the Next button.



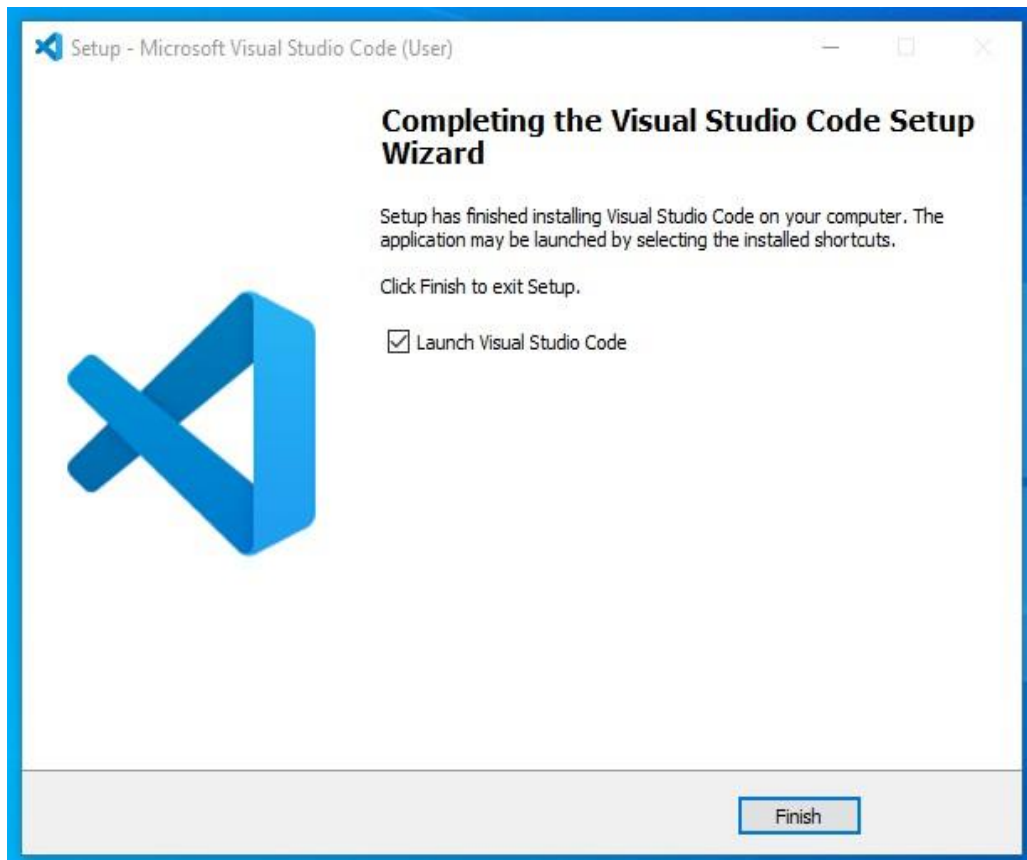
Step 7: Then it will ask to begin the installation setup. Click on the Install button.



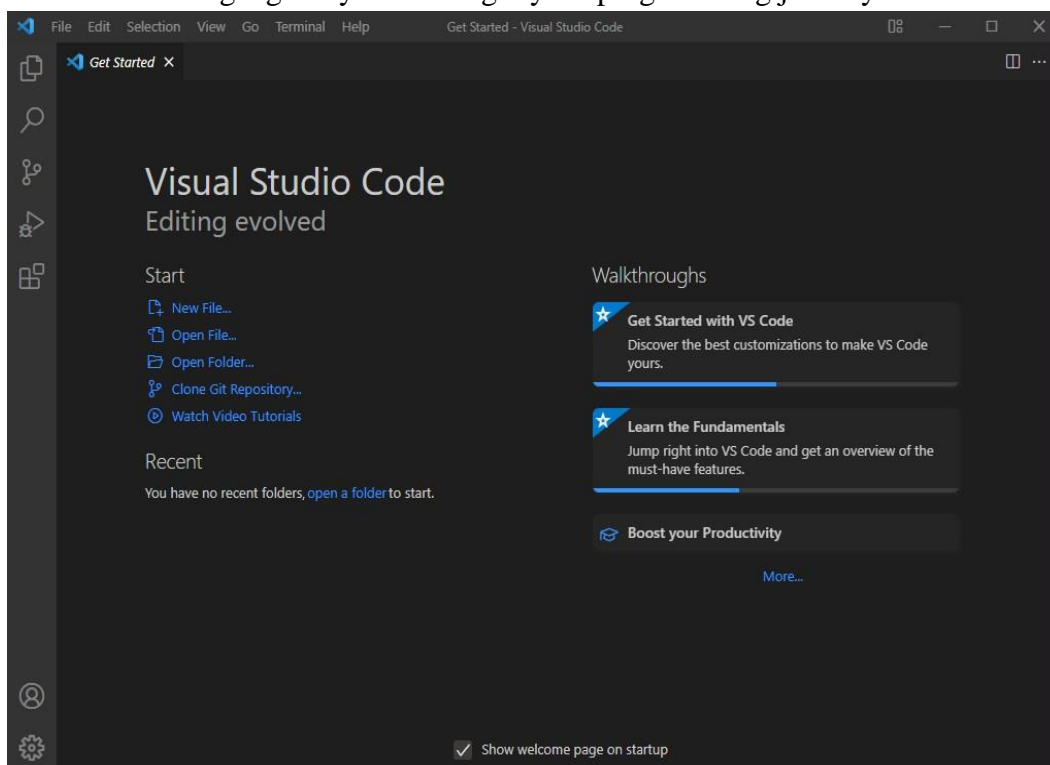
Step 8: After clicking on Install, it will take about 1 minute to install the Visual Studio Code on your device.



Step 9: After the Installation setup for Visual Studio Code is finished, it will show a window like this below. Tick the "Launch Visual Studio Code" checkbox and then click Next.



Step 10: After the previous step, the Visual Studio Code window opens successfully. Now you can create a new file in the Visual Studio Code window and choose a language of yours to begin your programming journey!



10.2 GateWise - User Manual

Brief Description:

In today's fast-paced and interconnected world, traditional methods of managing residential societies are no longer efficient. Manual tracking, poor coordination, and lack of transparency can lead to issues in community engagement and service management. The proposed Society Management System (SMS) software aims to digitalize and streamline the management of residential societies by integrating core functions like alerts, visitor logs, delivery tracking, and complaint resolution into a single platform accessible through smartphones and laptops.

System Requirements:

- Minimum RAM: 8 GB
- Storage: 256 GB SSD
- Processor: Intel i3 or equivalent
- Devices Supported: Android and iOS smartphones, Laptops

User Types:

1. Admin
2. Residents

Step-by-Step User Guide – GateWise Community Dashboard

Step 1: Log in to GateWise

- Open the GateWise web application in your browser.
- Enter your username and password.
- Click Login to access the dashboard.

Step 2: Navigate the Dashboard

Once logged in, you'll land on the Community Dashboard, which gives an overview of:

- Notices (top alert section)
- Helpdesk status
- Visitor count
- Delivery updates

Step 3: Use the Sidebar Menu

Use the left-hand sidebar to access the following modules:

| Section | Action You Can Take |
|-------------------|---|
| Dashboard | Return to the home screen overview. |
| Helpdesk | View, raise, or update support tickets. |
| Staff | View details of staff and their duties. |
| Visitor | Log and monitor guest entries. |
| Vendor & Services | Manage service vendors like sell, rent etc. |
| Delivery | View package deliveries. |
| Directory | Browse resident contact information. |
| Alert | Send emergency or service alerts. |
| Log Out | Exit the system securely. |

Step 4: Check Helpdesk Tickets

- View the Helpdesk card to see the number of open and resolved tickets.
- Click on Helpdesk in the sidebar for more details.
- You can:
 - Open a new ticket (e.g., "Water leakage").
 - Check responses from society staff.
 - Close or follow up on issues.

Step 5: Monitor Visitors

- The Visitors card shows total guests logged.
- Click on Visitor in the sidebar to:
 - Add new visitor entries.
 - Check visitor history.
 - Approve/deny guest access.

Step 6: Track Deliveries

- See the Deliveries card to check recent parcels received.
- Click on Delivery in the sidebar to:
 - Mark packages as delivered.
 - Notify recipients.

Step 7: Switch Theme (Dark/Light Mode)

Click the moon icon on the top-right to toggle between Dark Mode and Light Mode.

Step 8: Log Out

Click “Log Out” in the sidebar after completing your tasks to securely exit the system.

Chapter 11: Conclusion & Future Work

11.1 Conclusion

The development of the Society Management System has successfully addressed several critical needs for efficient and streamlined operations within residential or community societies. The system provides a centralized platform for managing tasks such as member registration, financial transactions, maintenance requests, event management, and communication between society members and administrators. By automating many manual processes, the system has improved transparency, accountability, and responsiveness, reducing the administrative burden on society managers while enhancing the overall resident experience.

11.2 Future Work

Several areas of potential improvement and expansion for the Society Management System can be explored in future work:

- **Mobile Application Development:** Creating a dedicated mobile app for Android and iOS platforms to provide residents with more convenient access to society services, such as notifications, bill payments, and maintenance tracking on the go.
- **Advanced Data Analytics:** Implementing machine learning algorithms to analyze resident behavior, predict maintenance needs, or optimize resource usage would provide data-driven insights to both residents and society administrators.
- **Enhanced Security Features:** Adding multi-factor authentication (MFA) and more advanced encryption techniques to further secure sensitive data such as financial information and personal details of residents.
- **Customizable Modules:** Developing more customizable features that cater to the unique needs of different societies, enabling administrators to add or modify system modules without extensive coding.

- **Resident Feedback System:** Building a more sophisticated feedback mechanism where residents can rate services, report issues, and provide suggestions, leading to continuous system improvement based on user input.

While the Society Management System has proven effective in addressing key operational challenges, there is ample scope for enhancing its capabilities to make it even more user-centric and feature-rich. Future advancements will focus on providing greater flexibility, security, and intelligence, ensuring that the system remains a valuable tool for modern society management.

CHAPTER 12: ANNEXURE

12.1 Terms:

- **Resident Profile Management:** A feature to create, update, and manage resident details within the system.
- **Visitor Management:** A module for tracking visitor check-ins and approvals, enhancing society security.
- **Billing System:** Automates maintenance fee calculations, invoicing, and payment tracking for residents.
- **Admin Panel:** An interface for administrators to manage users, vendors, and system settings.
- **Helpdesk:** A system for residents to raise complaints and track resolutions.
- **Alerts:** Notifications sent to residents for events, emergencies, or important updates.
- **Vendor Management:** A feature to track and communicate with service providers like cleaners or repair personnel.
- **Reports and Analytics:** Tools to analyze society operations, such as staff attendance or financial transactions.
- **Multi-Factor Authentication (MFA):** A security method requiring two or more verification steps for system access.

12.2 Abbreviations:

- **SMS:** Society Management System
- **CRUD:** Create, Read, Update, Delete
- **UI/UX:** User Interface/User Experience
- **API:** Application Programming Interface
- **AWS:** Amazon Web Services
- **CI/CD:** Continuous Integration/Continuous Deployment
- **HTML:** Hypertext Markup Language
- **CSS:** Cascading Style Sheets
- **SQL:** Structured Query Language

12.3 References:

- <https://mysocietyclub.com/>
- [https://mscw.ac.in/NAAC/Criteria1/Samples-of-ProjectWork_Fieldwork/Computer_Science/software_Engineering/Software%20Engineering/society%20mgmnt/Software%20Mnagement%20Software\(SMS\).docx](https://mscw.ac.in/NAAC/Criteria1/Samples-of-ProjectWork_Fieldwork/Computer_Science/software_Engineering/Software%20Engineering/society%20mgmnt/Software%20Mnagement%20Software(SMS).docx)
- <https://chatgpt.com/>
- <https://github.com/DarkLord125/Society-Management-System>
- <https://www.perplexity.ai/backtoschool>
- <https://meeraacademy.com/dfd-for-society-management-system-project/>
- <https://www.freeprojectz.com/entity-relationship/apartment-maintenance-management-system-er-diagram>

12.4 About Tools and Technologies

Tools

- Visual Studio Code: Lightweight code editor for efficient development.
- GitHub: Version control and collaboration platform for managing code repositories.
- AWS (Amazon Web Services): Cloud hosting for scalable deployment of the system.
- Postman: API testing tool for validating backend functionality.
- Jira/Trello: Project management tools for tracking Agile sprint progress.
- MySQL Workbench: Database design and management interface for MySQL.

Technologies:

- Backend: Java, Spring Boot
- Frontend: HTML, CSS, JavaScript, Bootstrap
- Database: MySQL
- Hosting: AWS (S3, EC2)
- Security: OAuth 2.0, SSL/TLS
- Testing: JUnit, Selenium

About College



Ganpat University-U. V. Patel College of Engineering (GUNI-UVPCE) is situated in Ganpat Vidyanagar campus. It was established in September 1997 with the aim of providing educational opportunities to students from It is one of the constituent colleges of Ganpat University various strata of society. It was armed with the vision of educating and training young talented students of Gujarat in the field of Engineering and Technology so that they could meet the demands of Industries in Gujarat and across the globe.

The College is named after Shri Ugarchandbhai Varanasibhai Patel, a leading industrialist of Gujarat, for his generous support. It is a self-financed institute approved by All India Council for Technical Education (AICTE), New Delhi and the Commissioner ate of Technical Education, Government of Gujarat.

The College is spread over 25 acres of land and is a part of Ganpat Vidyanagar Campus. It has six ultra- modern buildings of architectural splendor, class rooms, tutorial rooms, seminar halls, offices, drawing hall, workshop, library, well equipped departmental laboratories, and several computer laboratories with internet connectivity through 1 Gbps Fiber link, satellite link education center with two-way audio anode-way video link. The superior infrastructure of the Institute is conducive for learning, research, and training.

Our dedicated efforts are directed towards leading our student community to the acme of technical excellence so that they can meet the requirements of the industry, the nation and the world at large. We aim to create a generation of students that possess technical expertise and are adept at utilizing the technical 'know-hows' in the service of mankind.

We strive towards these Aims and Objectives:

- To offer guidance, motivation, and inspiration to the students for well-rounded development of their personality.
- To impart technical and need-based education by conducting elaborated training programs.
- To shape and mould the personality of the future generation.
- To construct fertile ground for adapting to dire challenges.
- To cultivate the feeling of belongingness amongst the faction of engineers.