Software Requirements Specification

for

Society Sphere

Version 1.0 approved

Prepared by

Maham Javed 21L-1845

Ayza Tahir 21L-1854

Ali Farooq 21l-5088

Alina Abid 21L-5313

Sadia Inayat 21L-1853

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of the Society Sphere project is to develop an integrated software solution aimed at enhancing community management within residential societies or complexes. This Software Requirements Specification (SRS) document delineates the precise requirements and functionalities essential for the successful implementation of the Society Sphere platform.

Encompassing the entire ecosystem of community management, including resident interaction, administrative tasks, and facility management, this project aims to provide a robust digital platform for streamlining various operations and fostering better communication among residents and administrators.

The scope of the Society Sphere project spans across multiple modules and features, including:

- Resident authentication and account management.
- Complaint management and service request handling.
- Billing management for community services and amenities.
- Visitor registration and access control.
- Property transactions facilitation.
- Booking and management of community amenities and facilities.

This SRS document serves as a comprehensive blueprint for the development team, outlining the specific functionalities, interfaces, security measures, and performance requirements necessary to realize the Society Sphere platform. It establishes clear expectations for the final release, ensuring the delivery of a seamless, user-friendly, and efficient community management solution.

1.2 Document Conventions

- **Font and Formatting**:
  - This SRS employs a standard font style and size throughout the document to ensure consistency and readability.
- Section headings are formatted in bold for easy identification and navigation.
- Key terms or important points may be italicized or underlined to provide emphasis and clarity.
- **Prioritization**:

- Priorities are assigned to requirements using a defined scale, indicating the relative importance of each requirement.
- Higher-level priorities are assumed to be inherited by detailed requirements unless explicitly stated otherwise.

- **Special Symbols**:

- Special symbols or notations specific to the Society Sphere project may be used to highlight key points, potential risks, or other relevant information.
- A legend or key is provided within the document to explain the meaning of these symbols, ensuring clarity for readers.

- **Release Identification**:

- The current revision or release number of the SRS is clearly indicated at the beginning of the document.
- Stakeholders can easily reference the version of the SRS under consideration, facilitating effective communication and version control.

- **Version Control**:

- Any changes or updates made to the document are tracked and documented in a dedicated version control section.
- This section includes the date of modification, a brief description of the change, and the name or initials of the individual responsible for the modification.
- Version control ensures transparency and accountability in managing document revisions and updates.

1.3 Intended Audience and Reading Suggestions

1.3.1.1 Intended Audience:

The Software Requirements Specification (SRS) for Society Sphere is intended for various stakeholders involved in the development, management, and utilization of the Society Sphere platform. These stakeholders include:

- **Developers**: Those responsible for implementing the software, including frontend and backend developers, UI/UX designers, and database administrators.
- **Project Managers**: Individuals overseeing the planning, execution, and delivery of the Society Sphere project. This includes project managers, team leads, and scrum masters.
- **Testers**: Quality assurance engineers and testing teams responsible for validating the functionality, performance, and usability of the Society Sphere platform.
- **Marketing Staff**: Individuals involved in promoting and marketing the Society Sphere platform to potential users and stakeholders. This includes marketing managers, content creators, and social media strategists.

- **Users**: Residents and administrators who will interact with the Society Sphere platform on a regular basis. This includes both end-users who will utilize the platform's features and administrators who will manage the community.
- **Documentation Writers**: Those tasked with creating user guides, manuals, help documentation, and other instructional materials for the Society Sphere platform. This ensures users have access to comprehensive resources for utilizing the platform effectively.

1.3.1.2 Reading Suggestions:

For effective utilization of the Society Sphere SRS document, stakeholders are recommended to follow this suggested reading sequence:

- **Introduction and Overview**: Start with the Introduction section to gain an understanding of the project's objectives, scope, and context. Proceed to the Overview sections to familiarize yourself with the key features and functionalities of the Society Sphere platform.
- **Functional and Non-Functional Requirements**: Review the Functional and Non-Functional Requirements sections to understand the specific functionalities, constraints, and quality attributes expected from the Society Sphere platform.
- **Use Cases**: Dive into the Use Cases section to explore detailed descriptions of various user interactions and system behaviors. This will provide insight into how users will engage with the platform and achieve their objectives.
- **System Architecture and Data Model**: Explore the System Architecture and Data Model sections to understand the underlying structure, components, and data organization of the Society Sphere platform. This will provide a deeper understanding of the technical aspects of the system.
- **External Interfaces and Dependencies**: Review the External Interfaces, Constraints, Assumptions, and Dependencies sections to understand any external integrations, limitations, or dependencies that may impact the development and usage of the Society Sphere platform.
- **Appendices**: Finally, refer to the Appendices section for additional supporting information, such as glossaries, references, and supplementary documentation that may enhance your understanding of the Society Sphere platform.

Following this suggested reading sequence will provide stakeholders with a comprehensive understanding of the Society Sphere platform's requirements and specifications, tailored to their respective roles and responsibilities in the project.

1.3.2

1.4 Product Scope

The Society Sphere platform is a sophisticated digital solution crafted to revolutionize the management and interaction within residential communities. It serves as a comprehensive ecosystem tailored to meet the diverse needs of both residents and administrators, offering an array of features and functionalities aimed at enhancing community living experiences.

1.4.1.1 **Description and Purpose:**

Society Sphere serves as a centralized digital hub where residents and administrators can seamlessly communicate, collaborate, and manage various aspects of community life. The platform's core purpose is to streamline community management processes and foster a sense of belonging and engagement among residents. Key components and functionalities of the Society Sphere platform include:

- **User Authentication and Profiles**: Secure authentication mechanisms ensure that only authorized residents and administrators can access the platform.
- **Complaint Management System**: Residents can easily submit complaints or service requests through the platform, which administrators can efficiently manage, assign, and track to ensure swift resolution and resident satisfaction.
- **Billing and Payment Management**: Administrators can generate and manage bills for community services such as maintenance, utilities, and amenities. Residents can securely view bills, understand breakdowns, and make payments through the platform, promoting transparency and accountability.
- **Visitor Management**: Residents have the ability to register visitors, providing necessary details such as names, contact information, and visit durations. Administrators can approve registrations to maintain community security and monitor visitor activity.
- **Amenity Booking**: Residents can conveniently book community amenities such as pool areas, gyms, or event spaces through the platform. Administrators can efficiently manage bookings, ensuring fair allocation and optimizing resource utilization.

1.4.1.2 Corporate Goals and Business Strategies:

The Society Sphere platform is strategically aligned with corporate objectives and business strategies aimed at enhancing community living experiences, promoting operational efficiency, and fostering a strong sense of community within residential environments. The platform aims to achieve the following key objectives:

- **Enhanced Community Engagement**: By providing residents with user-friendly tools for communication, complaint resolution, and resource management, Society Sphere aims to enhance resident engagement and satisfaction, ultimately fostering a more cohesive and vibrant community.

- **Streamlined Community Management**: Society Sphere facilitates more efficient community management processes for administrators, reducing manual workload and enabling better organization and allocation of resources, thus optimizing operational efficiency.
- **Transparent Communication and Collaboration**: The platform promotes transparent communication and collaboration between residents and administrators, fostering trust and transparency by keeping all stakeholders informed and involved in community-related activities and decisions.
- **Cost Optimization and Resource Utilization**: By digitizing processes such as billing management and amenity booking, Society Sphere helps reduce administrative overheads, optimize resource allocation, and potentially lower operational costs for residential communities, contributing to long-term financial sustainability.

In essence, Society Sphere serves as a strategic enabler for achieving corporate objectives related to enhancing community living experiences, improving operational efficiency, and fostering a vibrant and connected community within residential environment

1.5 References

2. Overall Description

2.1 Product Perspective

The Society Sphere platform described in this SRS is a standalone product developed to address the need for a community-based social network. Originating from the growing demand for online communities, this platform provides users with a centralized environment for social interaction, content sharing, and community engagement. While independent, the Society Sphere platform may integrate with external systems for authentication, data storage, and communication purposes, ensuring seamless connectivity and interoperability with existing tools and services.

2.2 Product Functions

The platform offers a wide range of functionalities tailored to meet the diverse needs of residents, administrators, and service providers within the residential community. Below is a high-level summary of the major functions:

- **Apply for Registration**: Residents can apply for registration on the platform by filling out a registration form, providing necessary personal information, and submitting the application for verification.
- **Check Bills**: Residents can view their old bills.
- **Pay Bills**: Residents can verify bill payments by confirming bank details, providing payment screenshots, and verifying payment timestamps, ensuring accuracy and accountability in financial transactions.
- **Register a Visitor**: Residents can register visitors for entry into the premises, ensuring security and streamlined visitor management processes.
- **Book Amenities**: Residents can book community amenities for specified dates and times, ensuring fair access and efficient utilization of resources.
- **Manage Complaints**: Residents can submit complaints, track their resolution status, and provide feedback, facilitating effective communication and issue resolution within the community.
- **Managing Members**: Administrators can manage member accounts, verify login credentials, and handle password changes, ensuring secure access and account management.
- **Verify Registration**: Administrators can verify the registration of new users, granting access to system features and ensuring the integrity of user data.
- **Generate Bills**: Administrators can generate bills for customers based on billing periods and criteria, streamlining the billing process and enhancing financial management.
- **View Visitors**: Administrators can view a list of all visitors in the system, enabling effective visitor tracking and management for security purposes.
- **View Bookings**: Administrators can view bookings made by residents for community amenities, facilitating resource allocation and scheduling.
- **Resolve Complaints**: Administrators can resolve customer complaints, implement corrective measures, and update complaint statuses, ensuring prompt and satisfactory issue resolution.
- **View Bills**: Administrators can view all bills in the system, enabling comprehensive financial oversight and management.
- **Sell House**: Residents can list their houses for sale through an admin-managed real estate system, streamlining the selling process and expanding housing options within the community.
- **Buy House**: Residents can purchase houses listed on the real estate platform, facilitating property transactions and promoting community growth and diversity.

These functions collectively contribute to the efficiency, transparency, and overall well-being of the residential community, fostering a harmonious and thriving living environments for all stakeholders involved.

2.3 User Classes and Characteristics

Based on the provided classes, here are the identified user classes along with their characteristics:

- **House Owners/Residents:**

o **Characteristics:**
- Own or reside in properties within society.
- Need access to functions related to managing their properties, paying bills, registering complaints, and accessing amenities.
- Vary in technical expertise and familiarity with the system.

o **Importance:** High. They are primary users who directly interact with the system for various purposes related to their residency.

- **Admins:**

o **Characteristics:**
- System administrators responsible for managing and overseeing the entire system.
- Have elevated privileges to perform tasks such as user verification, complaint resolution, bill generation, and overall system maintenance.
- Require high technical expertise and system knowledge.

o **Importance:** High. Admins play a critical role in ensuring the smooth operation and security of the system.

- **Payment Processors:**

o **Characteristics:**
- External entities responsible for processing payments for bills and other transactions.
- Require integration with the system to facilitate secure payment transactions.
- May have limited interaction with the system interface but play a crucial role in financial transactions.

o **Importance:** Medium to High. Their integration is essential for enabling secure financial transactions within the system.

- **Visitors:**

o **Characteristics:**
- Individuals who visit the society for various purposes.
- Need to register their visits and provide necessary details.
- Limited interaction with the system compared to residents and admins.

o **Importance:** Medium. While not as frequent as residents or admins, visitors still interact with the system for registration purposes.

- **Complaint Handlers:**

o **Characteristics:**
- Responsible for receiving, managing, and resolving complaints submitted by residents.
- Require access to the complaint management system to process and address resident grievances.

o **Importance:** Medium. They ensure that resident complaints are addressed promptly and effectively, contributing to resident satisfaction.

- **Amenity Managers:**

o **Characteristics:**

- Oversee the management and scheduling of amenities such as grounds and pools within the society.
- Responsible for ensuring the proper utilization and maintenance of amenities.
- Require access to the amenity booking system and related functionalities.

o **Importance:** Medium. They ensure that amenities are available and accessible to residents while maintaining their upkeep.

2.4 Operating Environment

The operating environment for the software can be summarized as follows:

Hardware Platform:

- The software should be designed to operate on a variety of hardware platforms to ensure accessibility for users. This includes:

o Personal computers (desktops and laptops)

o Mobile devices

Operating Systems and Versions:

- The software should be compatible with a range of operating systems to accommodate different user preferences and device types. This includes:

o Windows

o macOS (versions 10.12 and above)

o Android

o iOS

- Compatibility with various operating system versions ensures broad accessibility across different devices used by residents, admins, and other stakeholders.

2.5 Design and Implementation Constraints

- **Community Engagement:** The platform should facilitate community engagement and participation, requiring features such as forums, discussion boards, or social networking capabilities. Design and implementation should prioritize functionalities that foster interaction and collaboration among residents.
- **Privacy and Confidentiality:** Given the sensitive nature of information within a society environment, strict privacy and confidentiality measures must be enforced. Access controls, data encryption, and anonymization techniques should be implemented to safeguard resident data and ensure compliance with privacy regulations.
- **Customization and Localization:** The platform may need to support customization to accommodate specific requirements of different societies or communities. Localization features, including multilingual support and region-specific settings, may be necessary to cater to diverse user demographics.

- **Integration with Community Services:** Integration with local service providers or community amenities, such as emergency services, maintenance teams, or recreational facilities, may be required. APIs and interfaces should be designed to enable seamless integration with these external services, enhancing the overall user experience.

- **Community Governance:** The platform may need to support features for community governance, such as voting mechanisms for decision-making or administrative functions for managing community resources. Design and implementation should accommodate these governance requirements while ensuring transparency and fairness.
- **Data Ownership and Control:** Residents should have ownership and control over their data within the platform. Clear policies and mechanisms for data ownership, consent management, and data portability should be implemented to empower residents and build trust in the platform.

Addressing these constraints will be crucial in designing and implementing a society sphere platform that effectively meets the needs of residents while ensuring security, privacy, and usability.

2.6 User Documentation

or the society sphere platform, the following user documentation components will be delivered along with the software:

- **User Manuals:** Comprehensive guides that provide step-by-step instructions on how to use the platform's features and functionalities. User manuals will cover topics such as account registration, navigation, accessing community services, managing profiles, and engaging with other users.
- **Online Help:** Interactive help resources accessible within the platform, offering contextual guidance and support to users as they navigate different sections and perform tasks. Online help may include tooltips, contextual pop-ups, and searchable knowledge bases to assist users in real-time.
- **Tutorials:** Interactive tutorials or walkthroughs designed to familiarize users with the platform's key features and workflows. Tutorials may include guided tours, interactive demos, or video guides that demonstrate how to accomplish common tasks and achieve specific goals within the platform.
- **Community Forums:** Online discussion forums or community platforms where users can engage with peers, ask questions, share experiences, and collaborate on platform-related topics. Community forums foster user interaction, knowledge sharing, and peer-to-peer support within the user community.

- **Feedback Channels:** Channels for users to provide feedback, suggestions, and report issues or bugs encountered while using the platform. Feedback channels may include online forms, surveys, or dedicated communication channels for direct interaction with platform administrators and support teams.

2.7 Assumptions and Dependencies

Assumptions:

- **Community Engagement:** It is assumed that residents and members of the society will actively engage with the platform for various purposes such as registering visitors, submitting complaints, and booking amenities. Adequate promotion and communication within the community are assumed to encourage user adoption and participation.
- **Sufficient Internet Connectivity:** The assumption is made that residents within the society have reliable internet connectivity to access the online platform. In areas with poor internet infrastructure, access to the platform may be limited, impacting the effectiveness of features like online bookings and complaints management.
- **User Training and Support:** It is assumed that residents and administrative staff will receive sufficient training and support to effectively use the platform. User-friendly interfaces and comprehensive documentation are assumed to be provided to assist users in navigating the platform's features.
- **Security and Privacy Compliance:** The assumption is made that the platform will comply with relevant security and privacy regulations to protect user data. Measures such as encryption, access controls, and regular security audits are assumed to be in place to safeguard sensitive information.

Dependencies:

- **Integration with Society Management Systems:** The project depends on integration with existing society management systems or databases to access resident information, visitor records, and amenity booking schedules. Seamless integration is crucial for maintaining data consistency and providing accurate services to users.
- **Regulatory Compliance:** Dependencies exist on compliance with local regulations and guidelines governing society management practices. Changes in regulations related to visitor registration, complaint resolution, and data privacy may require updates to the platform to ensure compliance.
- **Third-Party Services Integration:** Dependencies exist on third-party services such as payment gateways for processing dues and bills. Integration with these services requires

adherence to their APIs and protocols, and any changes or disruptions in these services may affect the platform's functionality.

- **Hardware Infrastructure:** The project relies on the availability of adequate hardware infrastructure to support the platform's operation. Dependencies exist on the reliability and scalability of hosting services, internet service providers, and networking equipment to ensure uninterrupted access to the platform.

These assumptions and dependencies are critical considerations for the successful development and deployment of the society sphere project. Addressing these factors effectively will contribute to the platform's usability, security, and overall value to the community.

3. External Interface Requirements

3.1 User Interfaces

3.2 Hardware Interfaces The society sphere project does not have direct hardware interfaces. However, it will be compatible with various hardware devices, including desktop computers, laptops, tablets, and smartphones, running on supported operating systems.

3.3 Software Interfaces

- **Database Integration:** The software will interface with a relational database management system (e.g., MySQL, PostgreSQL) to store and retrieve user data, transaction records, and system configurations.
- **Operating System Compatibility:** The software will be compatible with major operating systems such as Windows, macOS, iOS, and Android.

3.4 Communications Interfaces

Messaging Interface:

- **Communication Type:** Registers Complaints
- **Participants:** Admin, Resident
- **Purpose:** Facilitates communication between residents and administrators by allowing residents to register complaints, with administrators receiving notifications of these complaints.

- **Nature of Communications:**

- Residents will use the system interface to register complaints, providing details such as description and date.
- Upon complaint registration, the system notifies the admin of the new complaint.
- Admins can then access and view the complaints, taking appropriate actions to resolve them.
  - **Services Needed:** Notification service for administrators when a new complaint is registered.

4. System Features

Functional Requirements:

4.1.1.1 User Authentication:

- **User Login:**

- Users (residents and administrators) must be able to securely log in to their accounts using their username/email and password.
- The system should authenticate users' credentials to grant access to their respective accounts.
  - **User Registration:**
- New users should be able to sign up for accounts by providing necessary information such as name, email, address, and desired username/password.
- The system should verify the uniqueness of the chosen username and validate the provided information during registration.

4.1.1.2 Complaint Management:

- **Complaint Submission:**

- Residents should be able to submit complaints or service requests through the platform, providing details such as the nature of the issue, location, and urgency.
- The system should record and timestamp each complaint submission for tracking purposes.
  - **Administrative Management:**
- Administrators should have access to view and manage complaints, including assigning them to relevant personnel for resolution.
- Administrators should be able to update the status of complaints (e.g., pending, in progress, resolved) and add notes or comments regarding the resolution process.

4.1.1.3 Billing Management:

- **Billing Generation:**

- Administrators must be able to generate bills for community services, including maintenance, utilities, and amenities, based on predefined billing periods.

- The system should calculate charges accurately and generate itemized bills for each resident.
  - **Billing Viewing and Payment:**
- Residents should be able to securely view their bills on the platform, understanding the breakdown of charges and due dates.
- Residents should have the option to make payments securely through the platform using various payment methods.

4.1.1.4 Visitor Management:

- **Visitor Registration:**
- Residents should have the ability to register visitors by providing necessary details such as visitor's name, contact information, purpose of visit, and duration of stay.
- The system should validate the provided visitor information and record the visit details for security purposes.
  - **Administrative Approval:**
- Administrators must review and approve visitor registrations to ensure security within the community.
- Administrators should have access to view and manage visitor records, including the ability to revoke visitor access if necessary.

4.1.1.5 Property Transactions:

- **Property Listing:**
- Residents should be able to list properties for sale or express interest in purchasing properties through the platform, providing details such as property type, location, price, and description.
- The system should facilitate property transactions by matching buyers with sellers and providing necessary documentation for the transaction process.
  - **Administrative Oversight:**
- Administrators should oversee property transactions, ensuring transparency and adherence to community guidelines.
- Administrators may need to verify property listings and facilitate communication between buyers and sellers as needed.
- Booking Amenities:
  - **Resident Booking:**
    - Registered residents should be able to book community amenities such as the pool or ground through the platform.
    - Residents should specify the date, time, and duration for which they want to book the amenity.
    - The system should validate the resident's booking request and check the availability of the requested amenity for the specified date and time.
  - **Admin Approval:**
    - The system should notify administrators of pending booking requests.

  - Administrators should review the booking requests and check the availability of the requested amenity.
  - If the amenity is available for the requested date and time, the administrator should approve the booking request.
  - If the amenity is not available or there is a scheduling conflict, the administrator should reject the booking request and provide a reason.
- **Confirmation and Notification:**
  - Upon approval, the system should confirm the booking to the resident and update the availability status of the amenity.
  - The system should notify the resident of the approved booking, providing details such as the confirmed date, time, and location of the amenity.
- **Cancellation:**
  - Residents should have the option to cancel their booking if necessary.
  - The system should update the availability status of the amenity upon cancellation and notify the administrator of the cancellation.

4.1.2 Non-functional Requirements:

4.1.2.1 Security:

- **Data Encryption:** The platform must encrypt sensitive user data (e.g., login credentials, personal information) to prevent unauthorized access.
- **Access Control:** Access to user accounts and sensitive features should be restricted based on user roles and permissions.
- **Audit Trails:** The system should maintain audit trails to track user actions and system activities for security monitoring and compliance purposes.

4.1.2.2 Performance:

- **Scalability:** The platform should be able to handle a growing number of users and data volume without significant performance degradation.
- **Response Time:** The system should respond to user actions promptly, with page load times kept to a minimum.
- **Availability:** The platform should be available and accessible to users with minimal downtime or service interruptions.

4.1.2.3 Usability:

- **Intuitive Interface:** The user interface should be intuitive and easy to navigate, with clear instructions and visual cues to assist users in performing tasks.
- **Accessibility:** The platform should be accessible to users with disabilities, complying with web accessibility standards.

- **Multi-device Support:** The platform should be responsive and compatible with various devices and screen sizes, including desktops, tablets, and smartphones.

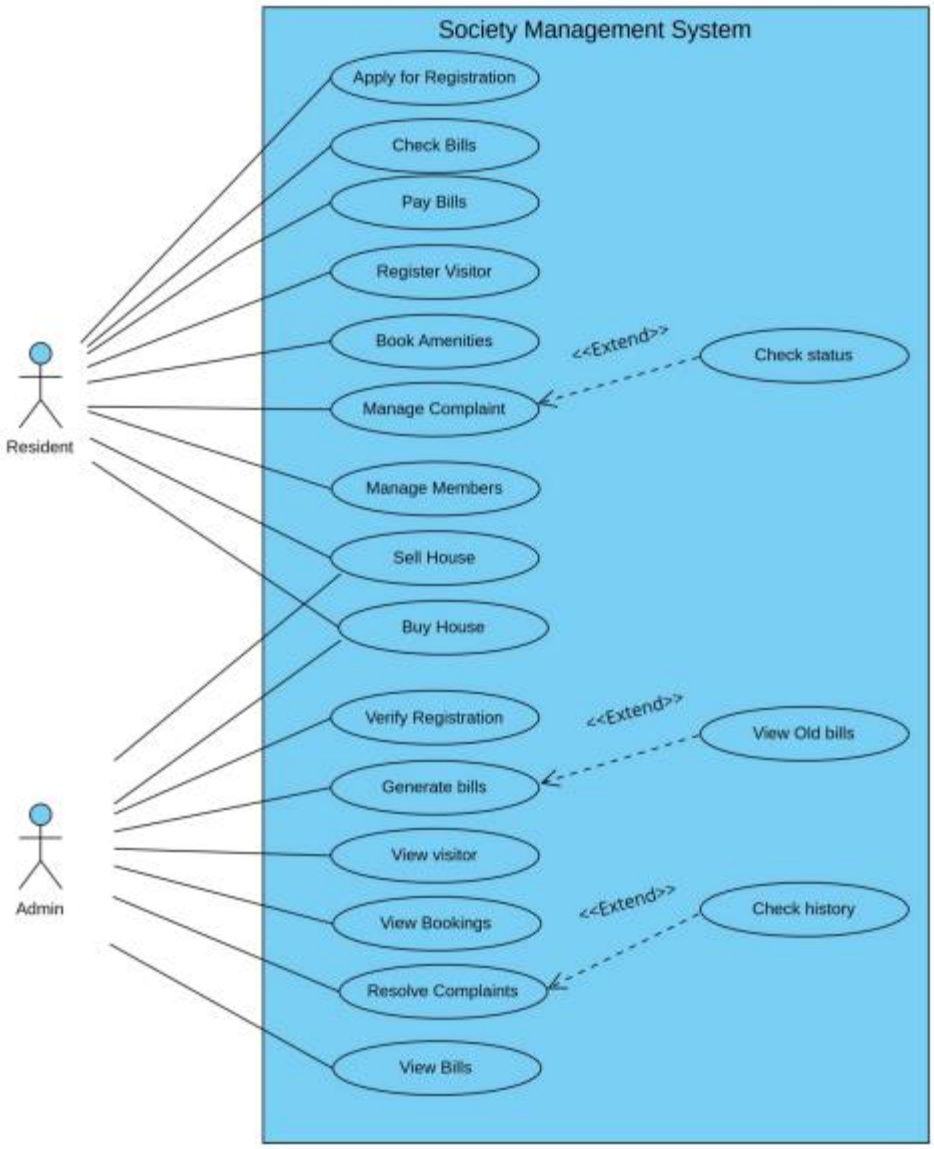
5. Other Requirements

Appendix A: Glossary

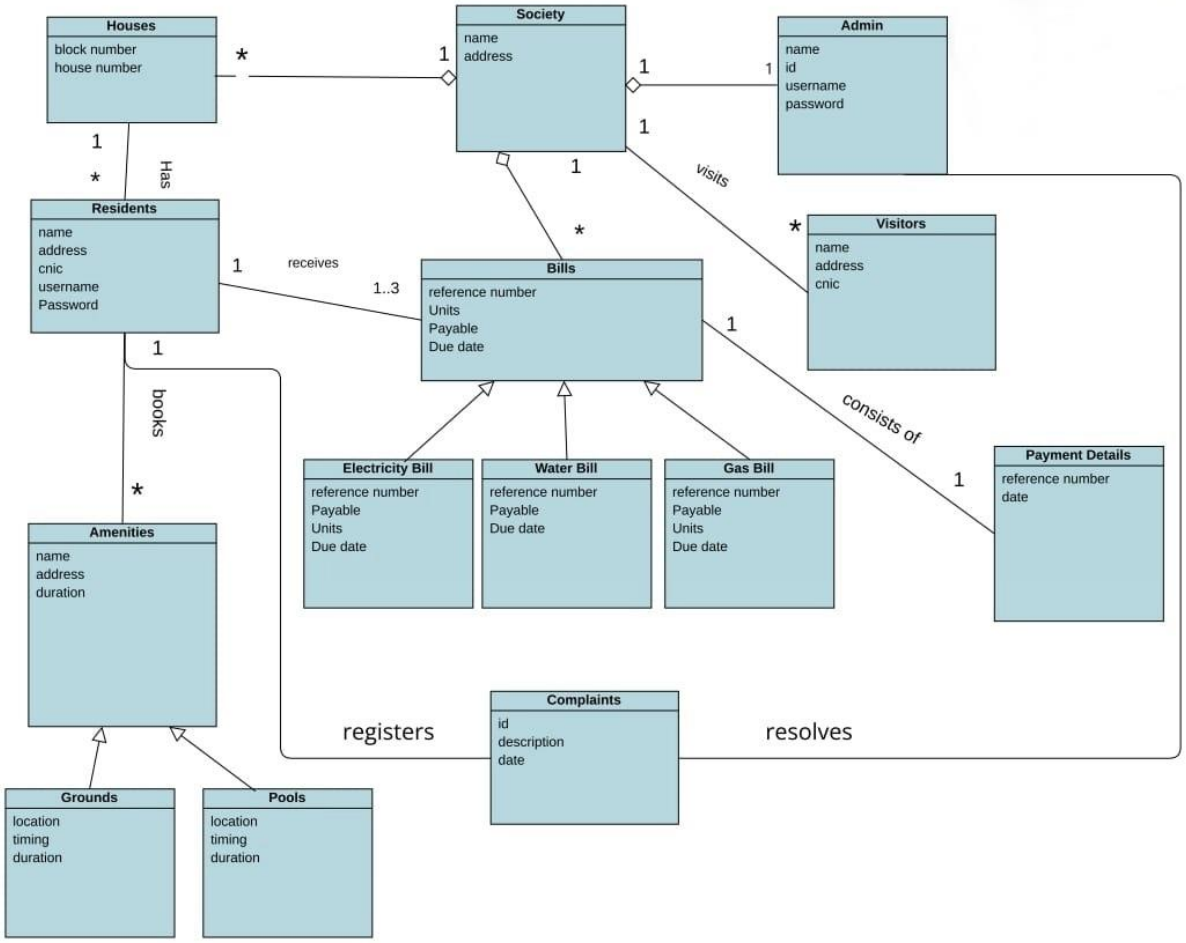
<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

USE CASE DIAGRAM



Class DIAGRAM



1.1: Apply for registration

Identifier	UC-001
Name	Apply for Registration
Summary	This use case describes the process of applying for registration on a website.
Priority	High
Actors	Resident
Pre-condition(s)	The resident is not registered on the platform. The resident has access to the registration form.
Post-condition(s)	The resident's registration application is submitted. The admin receives a notification for verification.

Typical Course of Action

S#	Actor Action	System Response
1.	The resident accesses the registration form.	
2.	The resident fills out the registration form, providing personal information such as name, email, and address.	
3.	The resident chooses a password and confirms it by entering it again.	
4.	The resident submits the registration application.	
5.		The system sends a notification to the admin with the resident's registration details for verification.

Alternate Course of Action (5. Details not correct/invalid)

S#	Actor Action	System Response
6.	If the resident provides incorrect or invalid information	
7.		System identifies the errors and provides feedback to the resident

1.2: Check bills

Identifier	UC-002
Name	Check bills
Summary	This use case allows a user to view their old bills and generate a PDF of the current month's bill.
Priority	High
Actors	Resident
Pre-condition(s)	The Resident must be logged into their account. The system must have access to the user's billing history. There should be a new bill available for the current month.
Post-condition(s)	The Resident can view their old bills. The Resident can generate a PDF of the current month's bill.

Typical Course of Action

S#	**Actor Action**	**System Response**
1.	Resident logs into their account.	
2.		The system authenticates the Resident.
3.	Resident navigates to the billing section.	
4.		The system displays the billing section.
5.	Resident selects the option to view bills.	
6.		The system retrieves and displays the Resident billing history.
	Resident selects a specific bill to view.	
		The system displays the selected bill.
	Resident selects the option to generate a PDF for the current month's bill.	The system generates a PDF of the current month's bill and provides a download link.

1.3 Pay Bills

Identifier	UC-003
Name	Pay Bills
Summary	This use case outlines the steps involved in a resident verifying a bill payment, which includes confirming bank details, providing a screenshot of the payment, and verifying the time of the payment.
Priority	High
Actors	Resident
Pre-condition(s)	The resident must have initiated a bill payment. The resident must have relevant payment details, including bank information and a screenshot of the payment

Post-condition(s)	The payment verification is completed successfully. The resident is assured that the payment has been made correctly.

Typical Course of Action

S#	Actor Action	System Response
1.	The resident logs into their account on the payment platform.	
2.	The resident selects the specific bill payment they want to verify.	
3.		The system displays the selected bill payment details
4.		The system presents the bank details used for the payment.
5.	The resident reviews the bank details to confirm that the payment was made to the correct bank or account.	
6.		The system provides an option to upload the screenshot
	The resident uploads the screenshot of the payment as supporting evidence.	
	The resident verifies the time details of the payment, such as the date and time of the transaction.	
	The resident submits the verification request.	
		The system records the verification request and notifies the resident of successful submission.

Alternate Course of Action (5. Details not correct/invalid)

S#	Actor Action	System Response
	If the resident notices any incorrect or invalid information during the verification process, they can choose to dispute the payment.	
		The system acknowledges the dispute request.
		The system initiates an investigation into the disputed payment.
		The system may request additional information or documentation from the resident to support the dispute.
		The resident receives updates on the status of the dispute resolution process.

		Once the dispute is resolved, the system notifies the resident of the outcome and any necessary actions to be taken.

1.4: Register a visitor:

Identifier	UC-004
Name	Register a visitor
Summary	This use case outlines the process for residents of a secure society to register visitors who wish to enter the premises. This use case takes into consideration the constraints of restricted access for security reasons.
Priority	Medium
Actors	Resident
Pre-condition(s)	The resident must be a member of the secure society. The resident must have a valid reason for the visitor's entry. The date and time of the visit must be predetermined. The visitor's identity and purpose of the visit must be known.
Post-condition(s)	The visitor is registered and granted access at the specified date and time. Security personnel are informed of the registered visitor.

Typical Course of Action

S#	Actor Action	System Response
1.	Initiates the visitor registration process.	
2.		System prompts the actor to provide visitor details including visitor's name, purpose of the visit, and date of the visit and time of the visit.
3.	Actor enters the required information	
4.		System verifies the information.
5.	If the information is valid.	
6.		System notifies security personnel grant access to the visitor at the specified date and time.
		System records the visitor's registration.

Alternate Course of Action (5. Details not correct/invalid)

S#	Actor Action	System Response
6.	If the actor (Resident) enters incorrect or invalid information	
		System displays an error message.
	Actor corrects the information.	
		System re-verifies the corrected information
	If the information is now valid, the system proceeds with the typical course of action.	
	If the information remains invalid after correction, the registration is denied, and the visitor is not granted access.	

1.5: Book amenities:

Identifier	UC-005	
Name	Book Amenities	
Summary	This use case describes the process of booking amenities and it ensures that only one amenity can be booked by one person at a time, and allows for bookings on different days or times.	
Priority	High	
Actors	Resident System	
Pre-condition(s)	The visitor must have access to the booking system. The visitor must be logged in.	
Post-condition(s)	The amenity is successfully booked for the specified date and time. The visitor's booking is recorded in the system.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	The resident selects the type of amenity they want to book	
2.	The resident specifies the date and time they want to book the amenity for.	
3.	The resident confirms the booking.	
4.		The system checks for availability of the selected amenity at the specified date and time.
5.		If the amenity is available, the system books it for the visitor.
6.		The system confirms the booking to the resident.
		The system records the booking in the database.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	The resident provides incorrect or invalid details during the booking process.	.
		The system detects the incorrect or invalid details.
		The system informs the visitor of the error and asks them to correct the details.
	The visitor corrects the details.	

1.6: Manage Complaint

Identifier	UC-006
Name	Manage Complaint
Summary	This use case describes how a resident can submit a complaint, have it resolved, and verify its resolution status.
Priority	Medium
Actors	Resident
Pre-condition(s)	The resident must be registered in the system. The resident must be logged into the system.
Post-condition(s)	The complaint is recorded in the database. The complaint data is stored in the database and not deleted.

Typical Course of Action

S#	Actor Action	System Response
1.	Resident logs into the system.	
2.	Resident navigates to the "Complaints" section	
3.		The system displays a list of existing complaints and also can write description and a "Submit Complaint" button.
4.	Resident clicks on the "Submit Complaint" button.	
5.		The system records the complaint in the database with a status of "Unresolved.
		The system displays a confirmation message to the resident.
	Resident can choose to check the status of the complaint or log out.	

		If the resident chooses to check the status of the complaint, they can see it marked as "Unresolved" in the complaints list.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**

1.7: Manage Members:

Identifier	UC-007
Name	Manage Members
Summary	This use case describes the process of managing member accounts within a system. It includes actions such as verifying login credentials, changing passwords, and handling scenarios where login details are not correct or invalid
Priority	High
Actors	Resident
Pre-condition(s)	The resident is registered as a member in the system. The resident has access to their login credentials (username and password).
Post-condition(s)	The member's account information is updated if necessary. The member's password is changed if requested.

Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	The Resident initiates the member management process.	
2.		The system presents options for member management, including changing the password and verifying login credentials.
3.	The Resident selects the "Verify Login" option.	
4.		The system prompts the Resident to enter their current username and password.
5.	The Resident enters their current username and password.	The system verifies the provided login credentials.
	If the login credentials are correct, the Resident proceeds to the next step. If the credentials are incorrect, the Resident is prompted to re-enter them	
	The Resident selects the "Change Password" option.	
	The system prompts the Resident to enter a new password.	

		The system validates the new password according to password policy rules, and if the password meets the requirements, it is update
	The Resident confirms the new password.	
		The system updates the member's password.
	The Resident confirms the member management process is complete.	
		The system displays a confirmation message and returns to the main menu.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	The Resident enters incorrect login credentials.	.
		The system informs the Resident that the login credentials are incorrect and provides an option to retry or return to the main menu.
	The Resident can choose to retry or return to the main menu.	
		If the Resident chooses to retry, they are prompted to re-enter their login credentials. If they choose to return to the main menu, the process ends.

1.8: Verify registration:

Identifier	UC-008
Name	Verify registration
Summary	This use case outlines the steps for an admin to verify the registration of a new user in the system.
Priority	High
Actors	Admin
Pre-condition(s)	Admin has logged into the system. A new user has completed the registration process but requires verification.
Post-condition(s)	The user's registration status is updated to "verified." The user gains access to their account and system features.

Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	Admin selects the "Pending Registrations" or similar option from the admin dashboard.	
2.		The system displays a list of users with pending registrations.
3.	Admin selects the user to be verified.	
4.		The system provides details of the selected user's registration information.
5.	Admin reviews the registration information for accuracy and completeness.	
		The system provides options to mark the user as "Verified" or "Not Verified."
	Admin selects "Verified" if the registration information is correct.	
		The user's registration status is updated to "verified."
		The user is notified of their verified status and provided access to their account.
	Admin selects "Not Verified" if the registration information is incorrect or incomplete.	
		The user's registration status remains as "pending."
		The admin may choose to send a notification or request additional information from the user to complete the registration.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	Admin selects "Not Verified" if the registration information is incorrect or invalid.	.
		The user's registration status remains as "pending."
		The admin may choose to send a notification or request additional information from the user to complete the registration.

1.9 Generate bills:

Identifier	UC-009
Name	Generate bills
Summary	This use case outlines the process by which an admin can generate bills for customers.
Priority	High
Actors	Admin
Pre-condition(s)	The admin must be logged into the billing system.

Post-condition(s)	Bills for the selected customers are generated and saved in the system. Customers receive their bills via email or other designated communication channels.

Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	The admin logs into the billing system	
2.		The system verifies the admin's credentials and grants access to the Admin Control Center.
3.	The admin navigates to the "Generate Bills" section	
4.		The system displays options for selecting the billing period and criteria for generating bills.
5.	The admin selects the billing period and criteria and initiates the bill generation process.	
		The system generates bills based on the selected criteria and displays a summary of the generated bills.
	The admin reviews the summary of generated bills for accuracy.	
		The system presents the admin with options to make corrections or adjustments if necessary.
	The admin confirms the accuracy of the generated bills.	
		The system finalizes the bills and saves them in the system.
	The admin selects the option to notify bills to customers.	
		The system notify to customers via email .
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	If the admin initiates the bill generation process with incorrect or invalid criteria.	
		The system displays an error message indicating that the criteria are not valid or the details provided are incorrect.
	The admin reviews the error message and either corrects the criteria or provides valid details.	
		The system allows the admin to make corrections and reinitiate the bill generation process.

1.10 View visitors:

Identifier	UC-010
Name	View visitors

Summary	This use case allows the admin to view a list of all visitors in the system.	
Priority	High	
Actors	Admin	
Pre-condition(s)	The admin is authenticated and logged into the system.	
Post-condition(s)	The admin is presented with a list of all visitors.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	The admin selects the "Access Visitor Records" option from the main menu.	
2.		The system retrieves a list of all visitors stored in the database.
3.	The system displays a paginated list of visitors, including their names, contact information, and visit details.	
4.		The list of visitors is displayed to the admin, allowing them to scroll through the pages to view all the visitors.
5.	The admin can click on a visitor's name to view more details or perform actions	
		If the admin clicks on a visitor's name, the system displays a detailed view of the visitor's information
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	The admin enters incorrect login credentials or encounters an authentication issue.	
		The system displays an error message, prompting the admin to enter the correct credentials or contact support for assistance.
	The admin selects the "Access Visitor Records" option, but there is an issue with the database connection or data retrieval.	
		The system displays an error message, informing the admin that there is a technical issue and advises them to try again later or contact technical support.
	The admin attempts to view a visitor's details that do not exist in the system.	
		The system displays a message indicating that the requested visitor's details are not found.

1.11 View bookings:

Identifier	UC-011	
Name	View bookings	
Summary	This use case describes how an admin can view bookings in the system.	
Priority	High	
Actors	Admin	
Pre-condition(s)	The admin is logged into the system. There are existing bookings in the system.	
Post-condition(s)	The admin successfully views the bookings.	
Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	the admin selects the "View Bookings" option from the admin dashboard	
2.		The system displays a list of existing bookings, including details such as booking ID, customer name, date, time, and location.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	The admin notices incorrect or invalid booking details while viewing bookings.	
		The system provides the admin with different options
		Cancel the booking if it's deemed necessary due to inaccuracies or invalid information.
		Update the booking status as "Pending" while awaiting clarification or correction from the resident.

1.12: Resolve complaints:

Identifier	UC-012
Name	Resolve complaints
Summary	This use case describes how an admin resolves a customer complaint.
Priority	High
Actors	Admin
Pre-condition(s)	Admin is logged into the complaint resolution system.
Post-condition(s)	The customer complaint is marked as resolved. If necessary, appropriate actions are taken to address the complaint.
Typical Course of Action	

S#	Actor Action	System Response
1.	The admin logs into the complaint resolution system.	
2.		The admin is successfully logged in and presented with a list of unresolved complaints.
	The admin selects an unresolved complaint from the list.	
		The details of the selected complaint are displayed on the screen.
	The admin reviews the complaint details and gathers any additional information if required.	
	The admin takes appropriate actions to resolve the complaint	
	The admin Implementing corrective measures to prevent similar complaints in the future.	
	Once the complaint is resolved, the admin updates the status of the complaint in the system to "Resolved."	
		The complaint status is changed to "Resolved," and the system records the date and time of resolution.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	The admin reviews the complaint details but realizes that the information provided is incorrect or invalid.	
		The admin may choose to contact the customer to verify the details or request more accurate information.

1.13: View bills

Identifier	UC-013
Name	View bills
Summary	This use case describes how an admin can view all bills in the system.
Priority	High
Actors	Admin
Pre-condition(s)	Admin is logged into the system. Bills exist in the system.
Post-condition(s)	Admin successfully views all bills. The system remains in the same state.

Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	Admin navigates to the "Billing Overview" section of the admin interface.	
		The system displays a list of all bills, including details such as bill ID, date, amount, and status.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	Admin enters incorrect login credentials.	
		The system displays an error message indicating that the login credentials are incorrect.
	Admin faces a network connectivity issue while trying to access billing data.	
		The system displays an error message indicating a network problem and advises the admin to check their internet connection.

1.14: Sell house

Identifier	UC-014
Name	Sell a house
Summary	This use case describes the process of a resident selling their house through an admin-managed real estate system.
Priority	High
Actors	Admin Resident
Pre-condition(s)	Resident has an active account in the real estate system. Admin has access to the real estate system.
Post-condition(s)	The resident successfully lists their house for sale. The admin reviews and approves the house listing.

Typical Course of Action		
S#	**Actor Action**	**System Response**
1.	Resident logs into their account.	
	Resident navigates to the "Sell a House" section.	
	Resident enters the details of the house they want to sell, including address, price, and description.	
	Resident uploads photos of the house.	
	Resident submits the listing for review.	

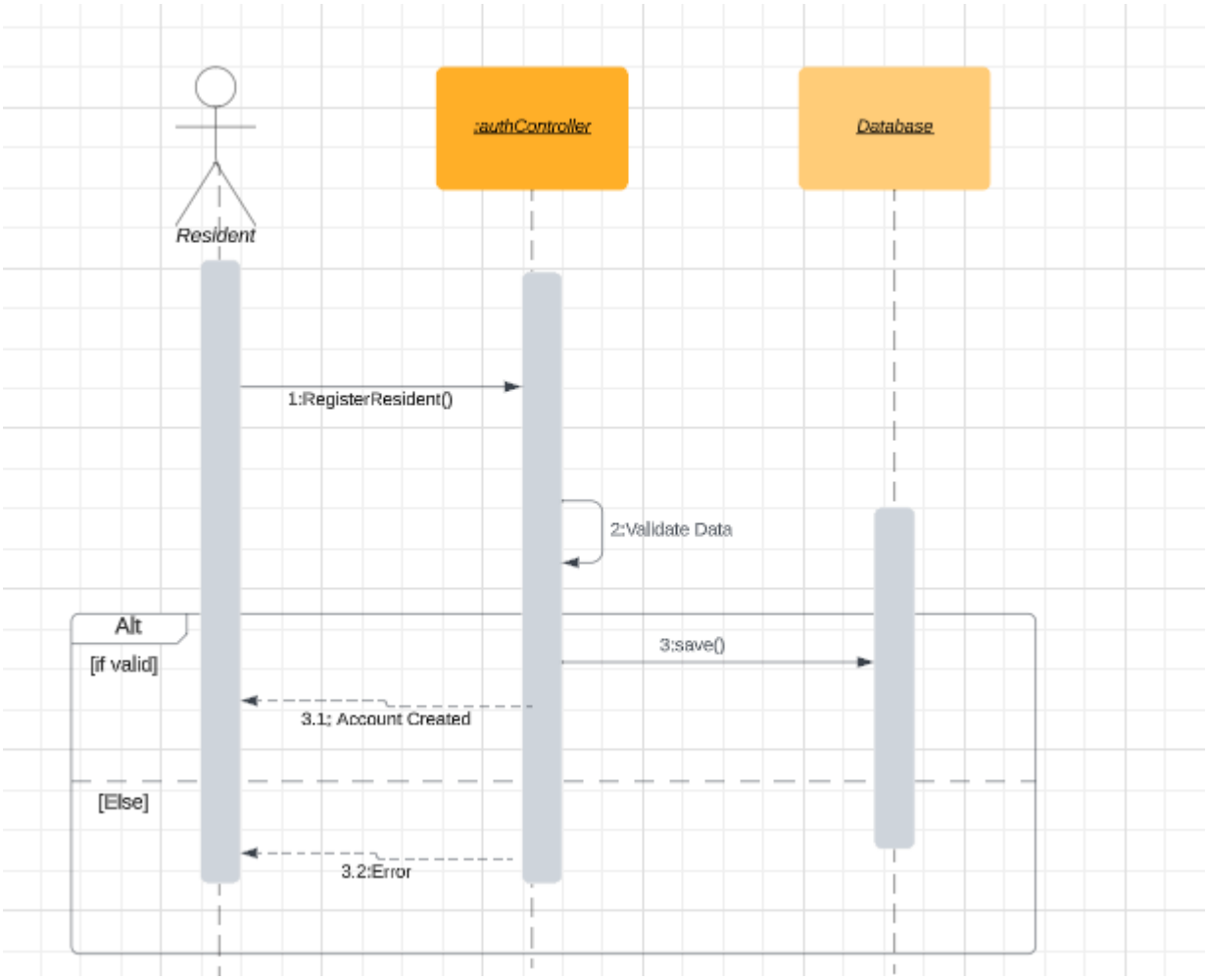
		The system confirms the submission and informs the resident that the listing is pending admin approval.
	Admin logs into their admin account and navigates to the "Pending Listings" section.	
	Admin reviews the details and photos of the house listing.	
	Admin either approves or rejects the listing.	
		If approved, the system publishes the house listing on the platform.
		If rejected, the system sends a notification to the resident with the reason for rejection.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	Resident enters incomplete or incorrect details of the house.	
	Resident uploads inappropriate or irrelevant photos and Resident submits the listing.	
		The system provides error messages indicating the issues with the submitted information.
		Resident corrects the information and resubmits the listing.
	Admin reviews the listing with incorrect or invalid details.	
		Admin rejects the listing and sends a notification to the resident explaining the reasons for rejection.

15: Buy house

Identifier	UC-015
Name	Buy house
Summary	This use case describes the process of a resident purchasing a house through an admin-managed real estate system.
Priority	High
Actors	Admin Resident
Pre-condition(s)	Resident has an active account in the real estate system. Admin has access to the real estate system. Houses are listed for sale on the real estate platform.
Post-condition(s)	The resident successfully purchases a house. The admin processes the purchase request and updates the house's status.
Typical Course of Action	
S#	**Actor Action**	**System Response**

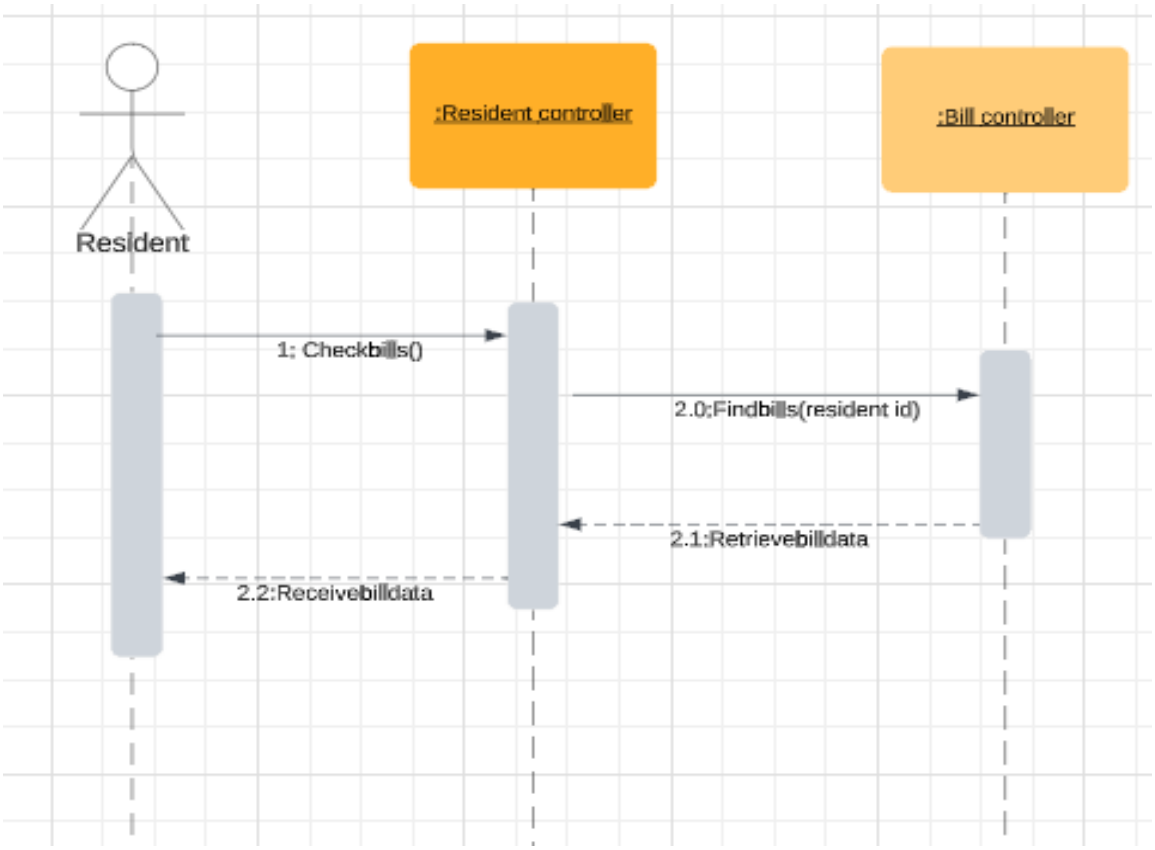
1.	Resident logs into their account.	
	Resident searches for houses based on their preferences (e.g., location, price, size).	
	Resident selects a house listing they are interested in.	
	Resident reviews the details and photos of the selected house.	
	Resident clicks on the "Buy Now" or "Contact Seller" button.	
		If "Buy Now" is selected, the system initiates the purchase process.
		If "Contact Seller" is selected, the system sends a message to the seller indicating the resident's interest.
	Admin receives a notification of the purchase request.	
	Admin reviews the request and verifies.	
		If the resident is verified, the system marks the house as "Pending Sale" and notifies the resident and seller.
	If the purchase is approved, the resident proceeds with the payment and provides necessary details.	
		The system processes the payment, updates the house's status to "Sold," and sends confirmation to both the resident and the seller.
Alternate Course of Action (5. Details not correct/invalid)		
S#	**Actor Action**	**System Response**
6.	Resident tries to initiate the purchase with incorrect or incomplete details.	
		The system provides error messages indicating the issues with the submitted information..
		Resident corrects the information and resubmits the purchase request.
	Admin receives the corrected purchase request.	
		Admin reviews the corrected request, and if the details are now correct, proceeds with the purchase process as outlined in the typical course of action.

1.1: Apply for registration

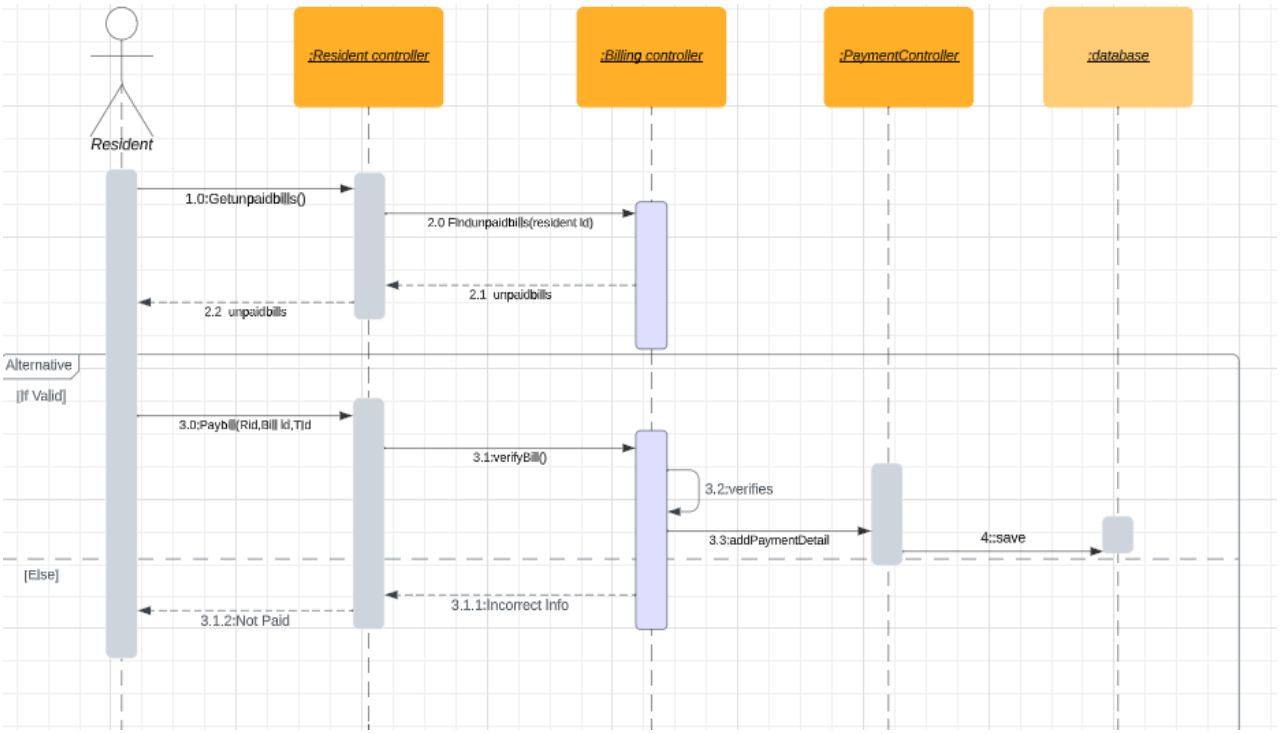


11

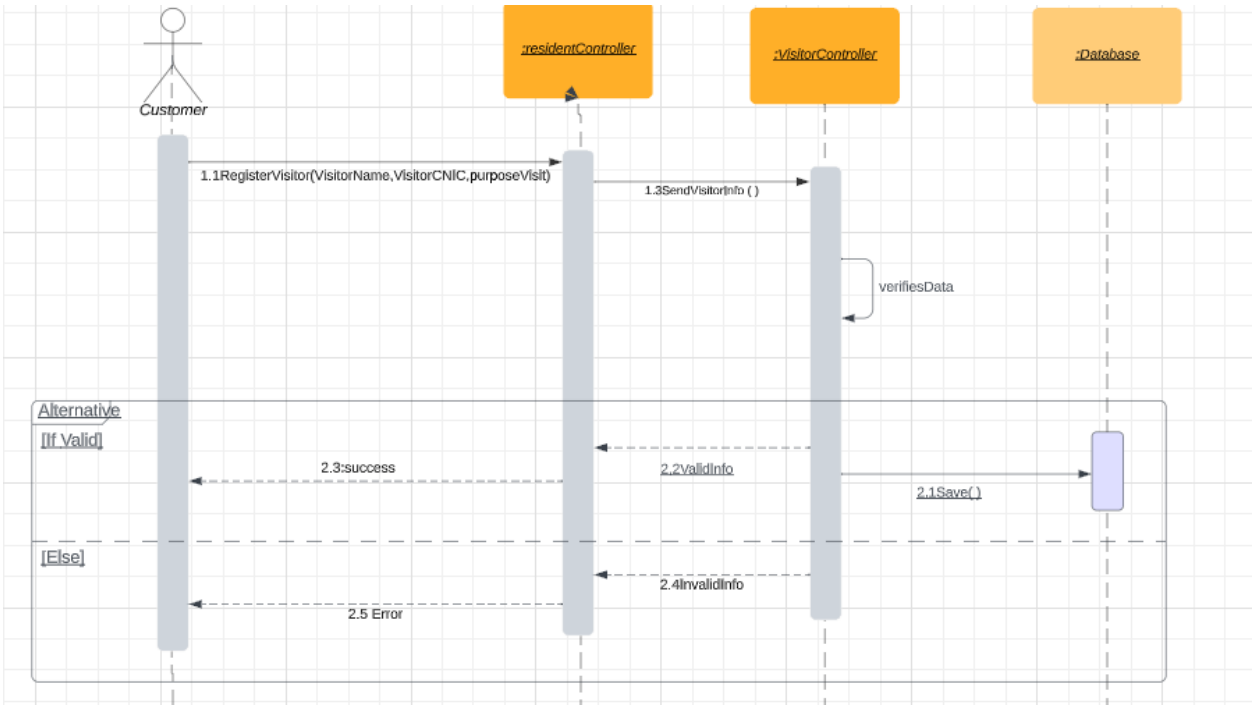
1.2:check bills


:Resident controller
:Bill controller
Resident
1: Checkbills()
2.0:Findbills(resident id)
2.1:Retrievebilldata
2.2:Receivebilldata

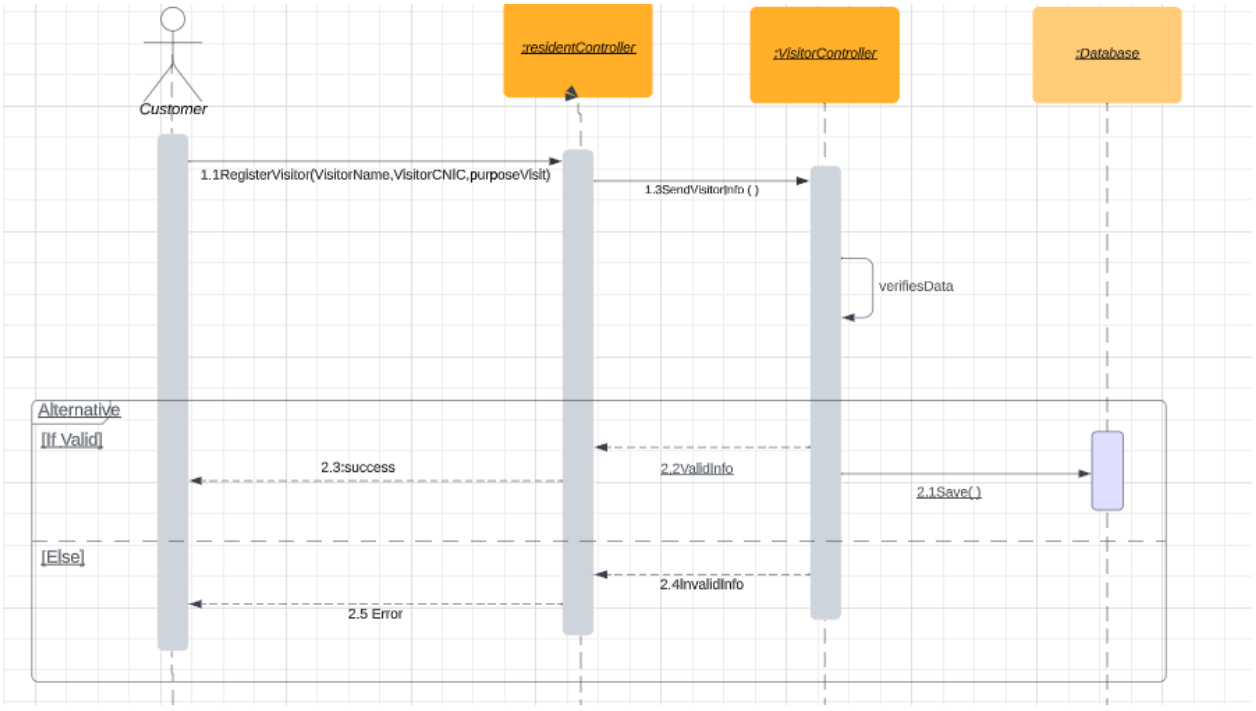
1.3 Pay Bills



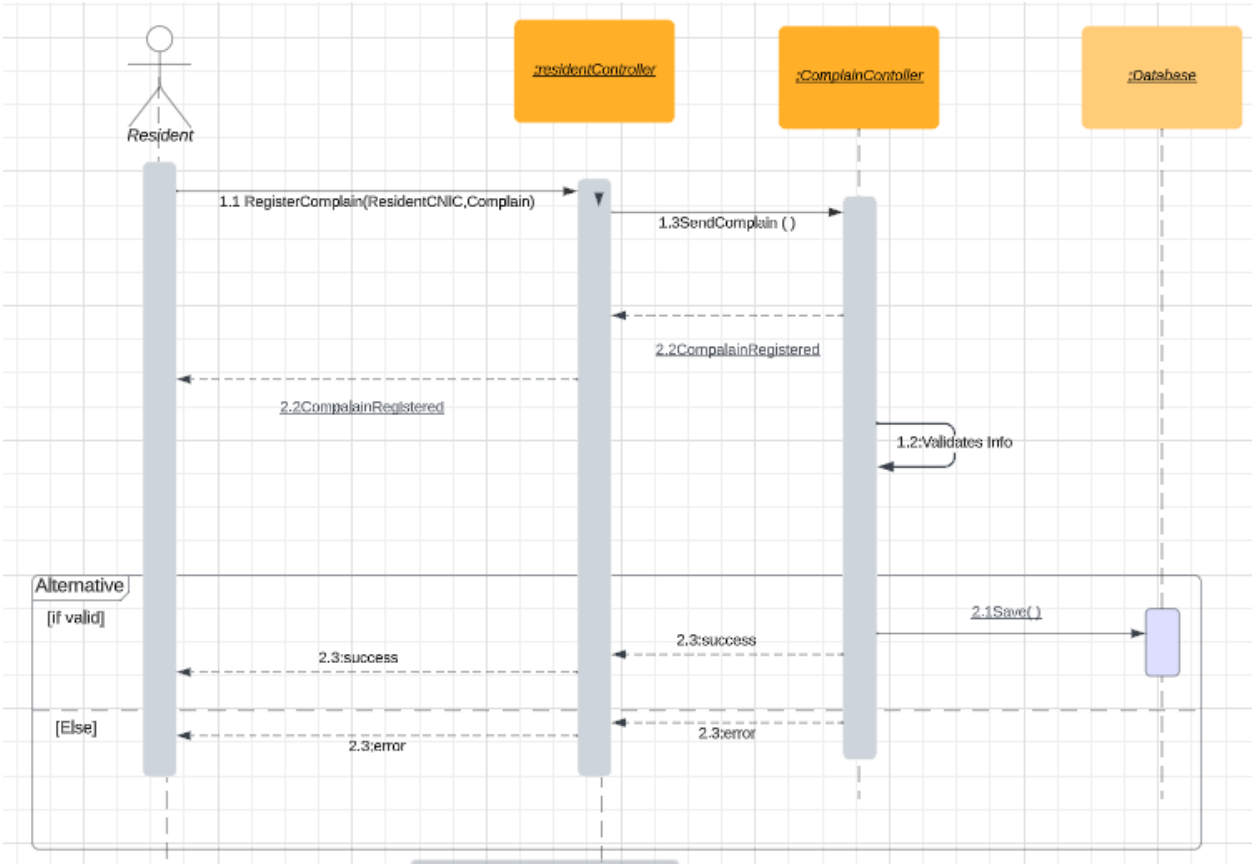
1.4: Register a visitor:



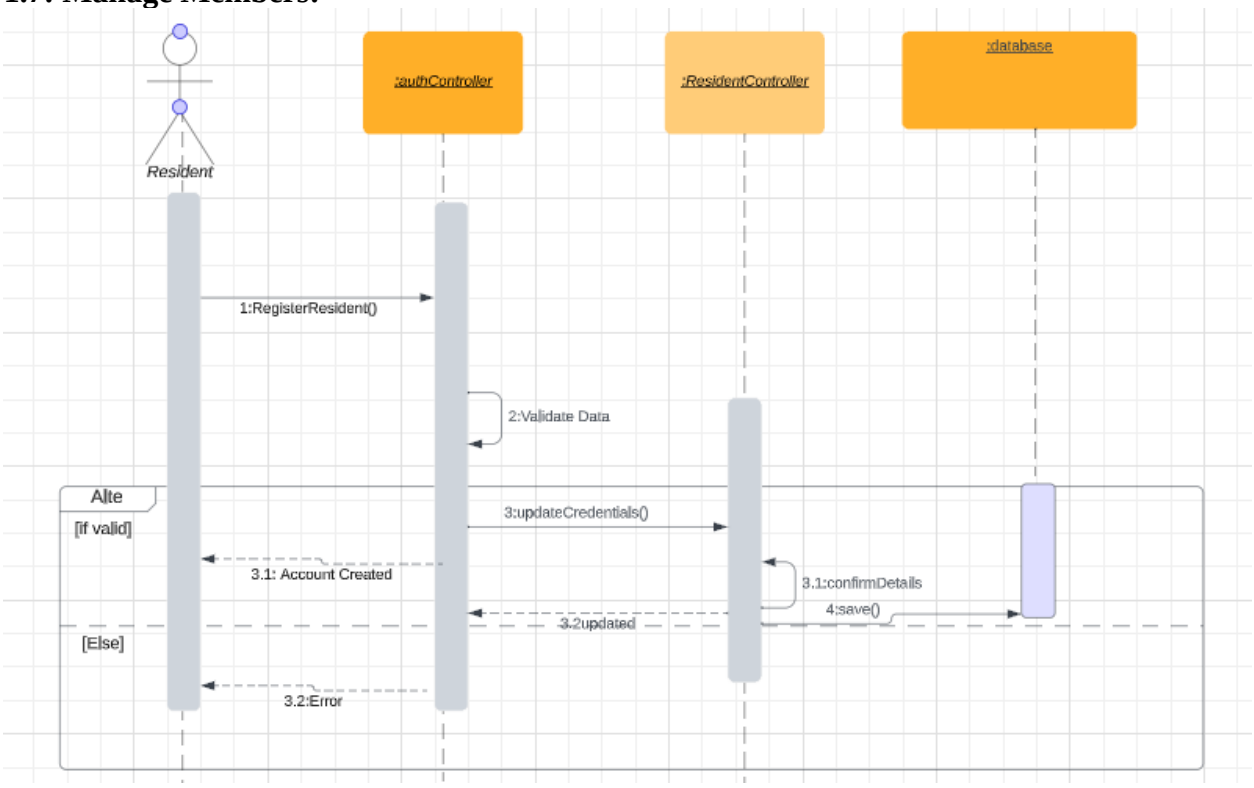
1.5: Book amenities:



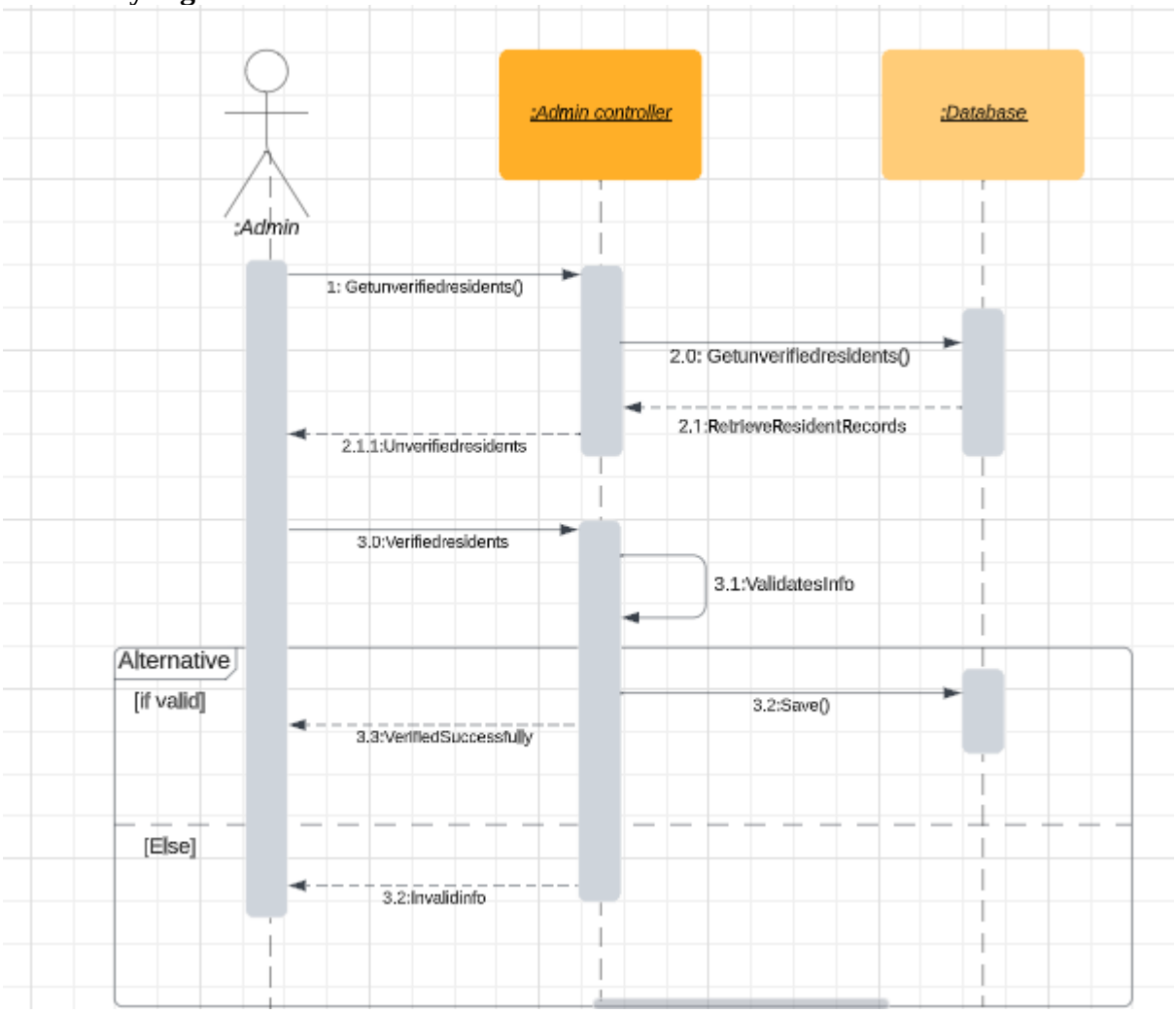
1.6: Manage Complaint



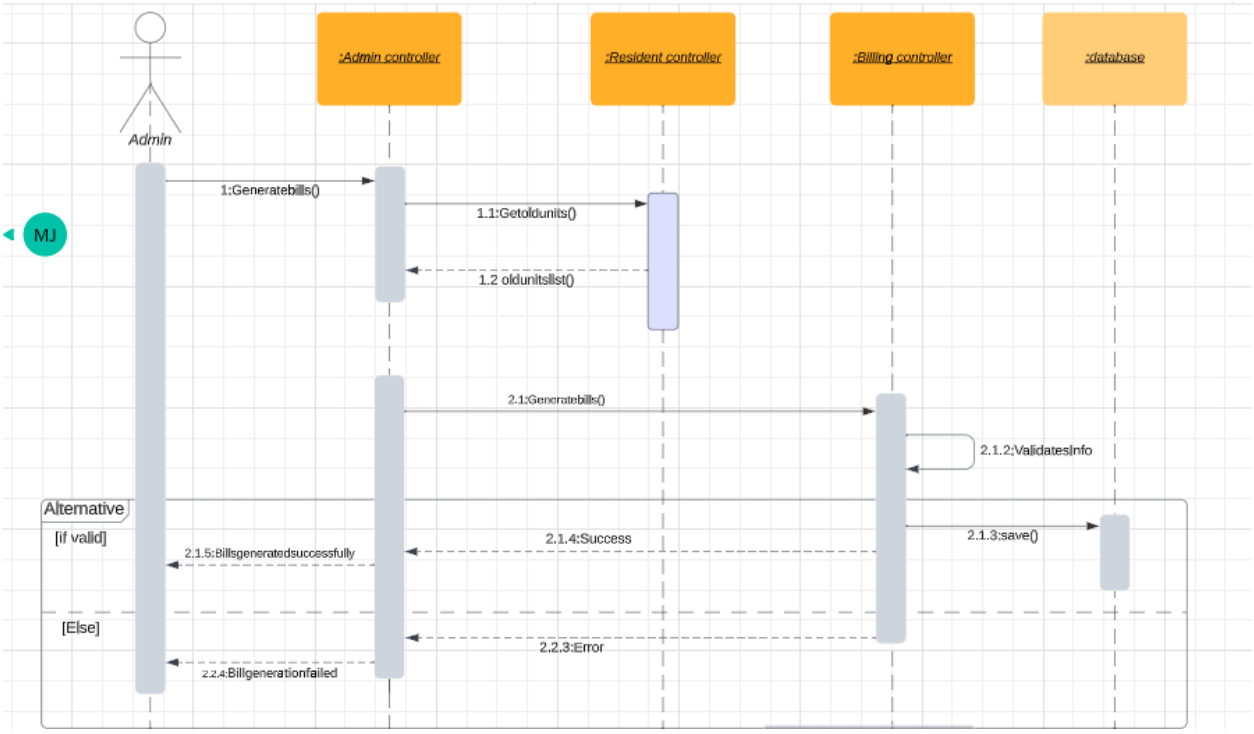
1.7: Manage Members:



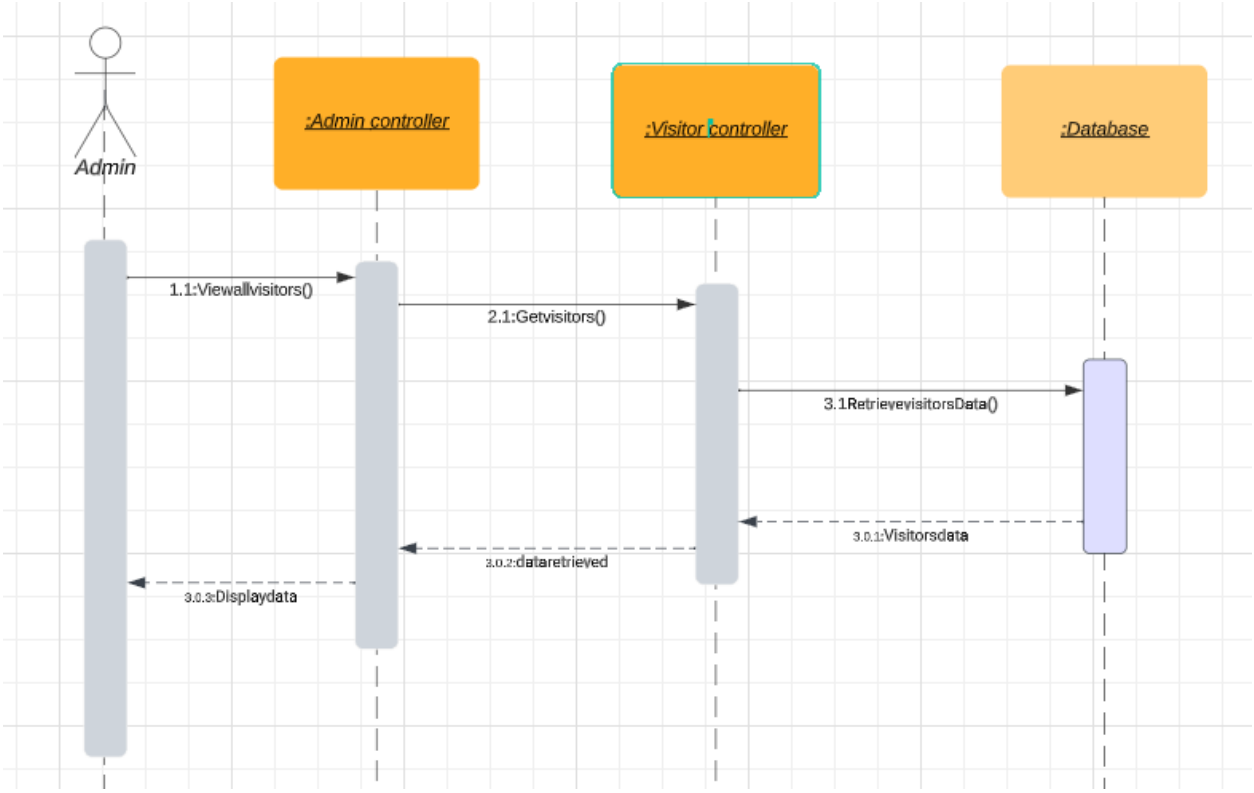
1.8: Verify registration



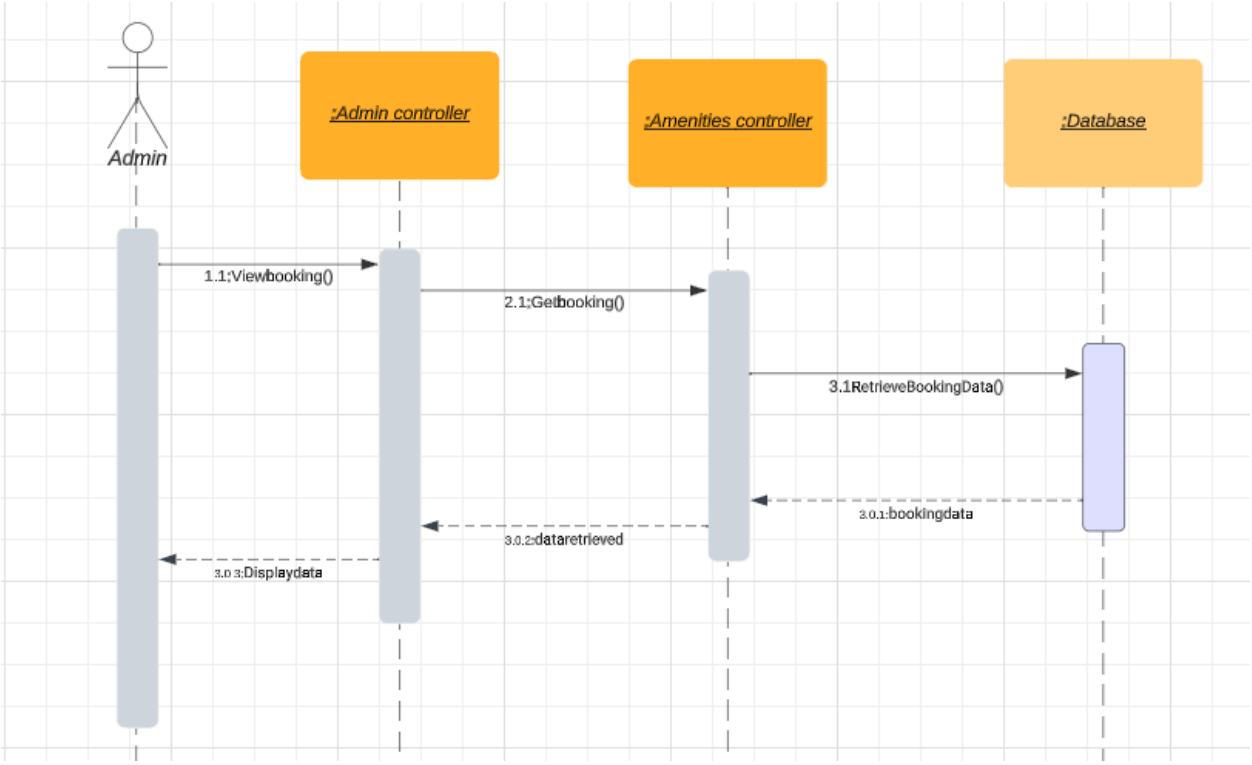
1.9 Generate bills:



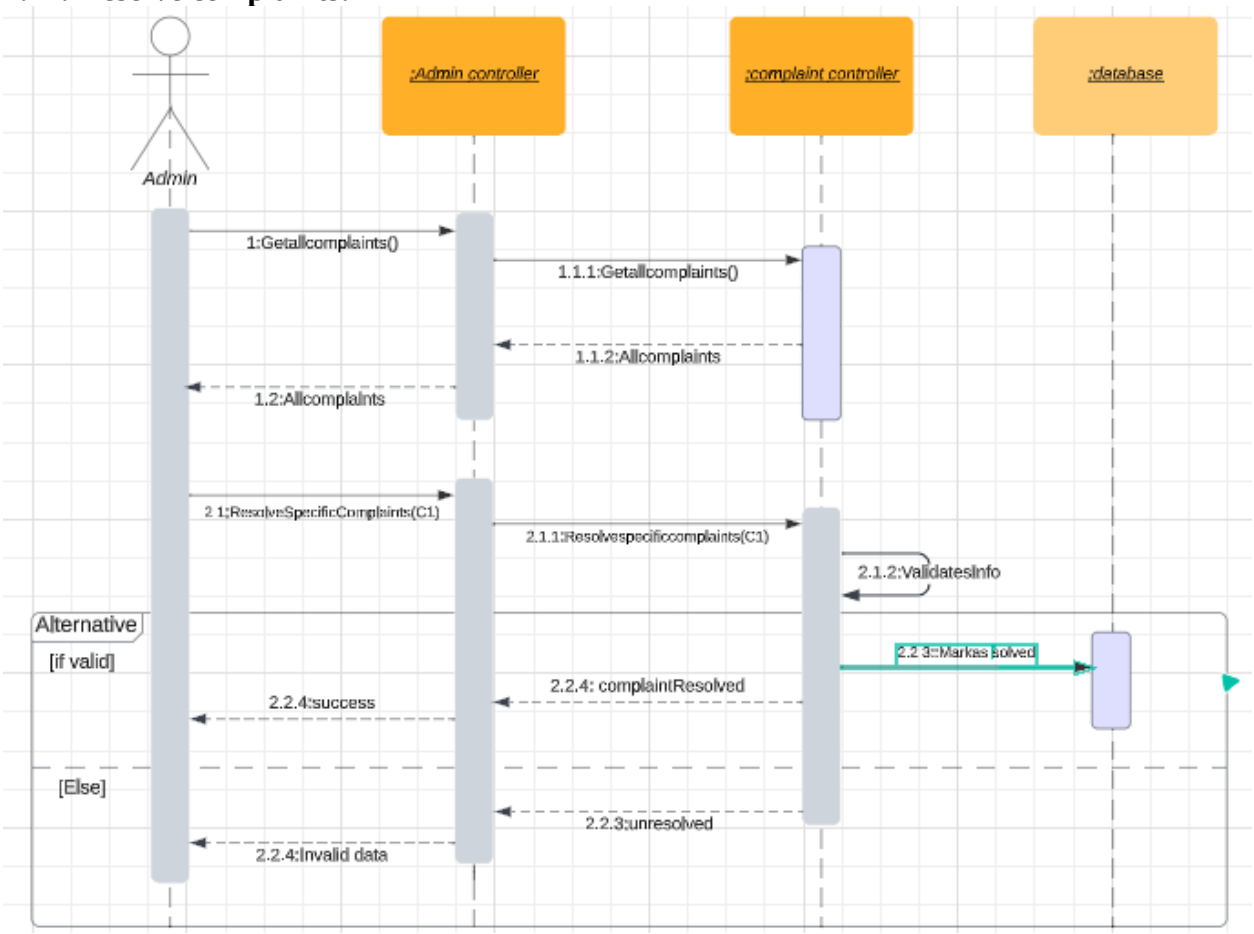
1.10 View visitors:



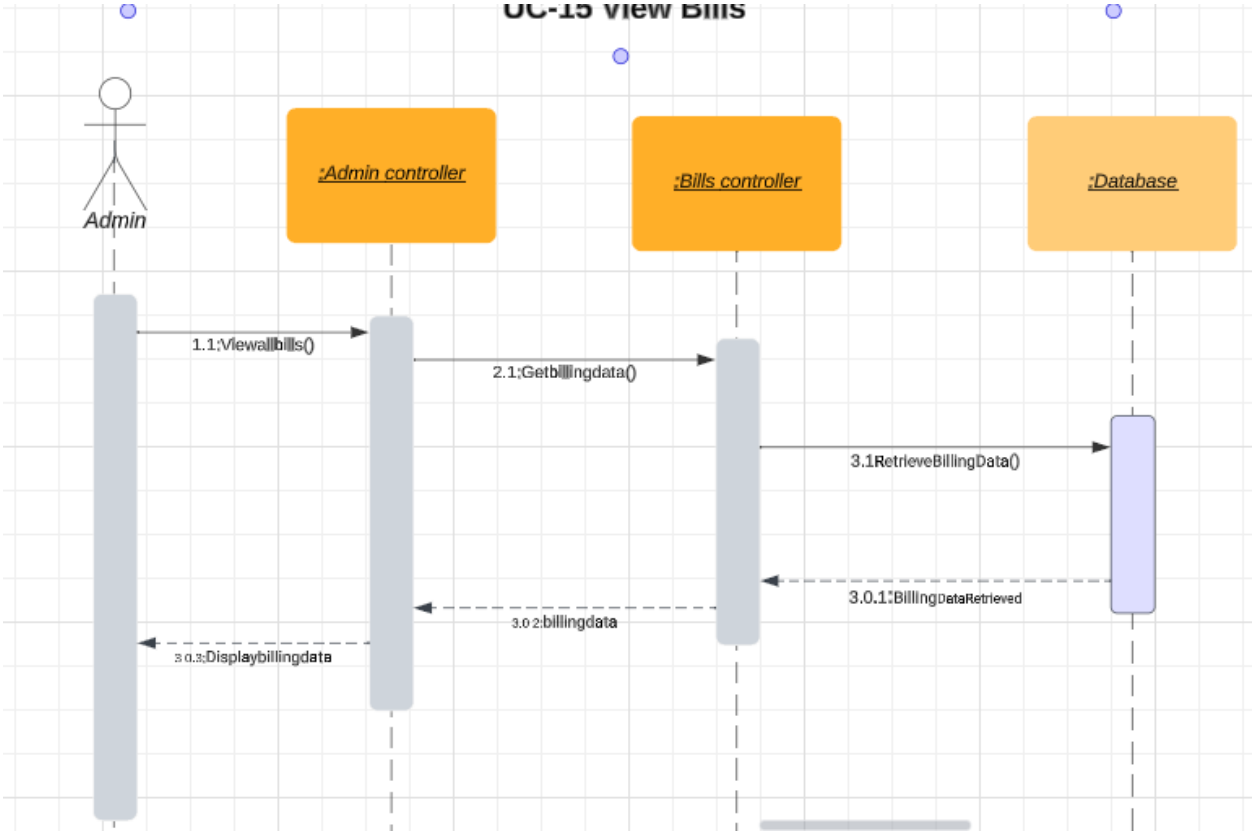
1.11 View bookings:


Admin
:Admin controller
:Amenities controller
:Database
1.1:Viewbooking()
2.1:Getbooking()
3.1RetrieveBookingData()
3.0.1:bookingdata
3.0.2:dataretrieved
3.0 3:Displaydata

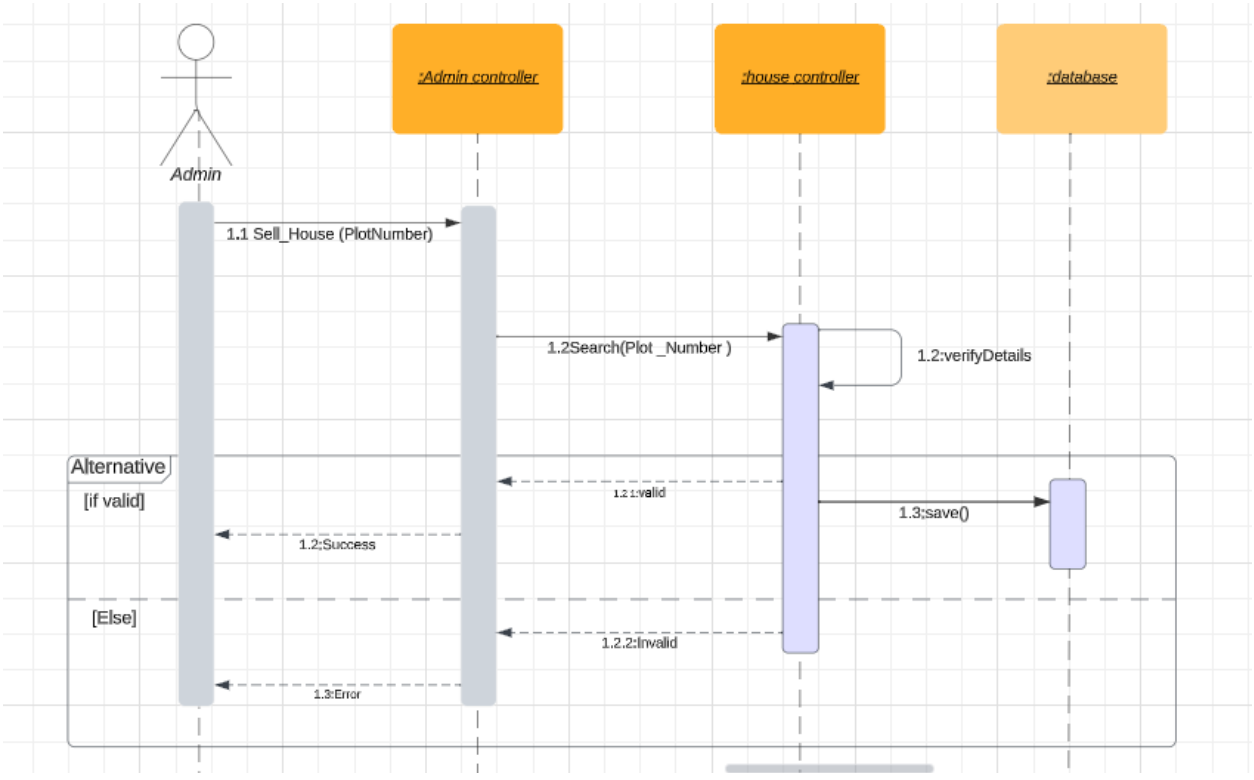
1.12: Resolve complaints:



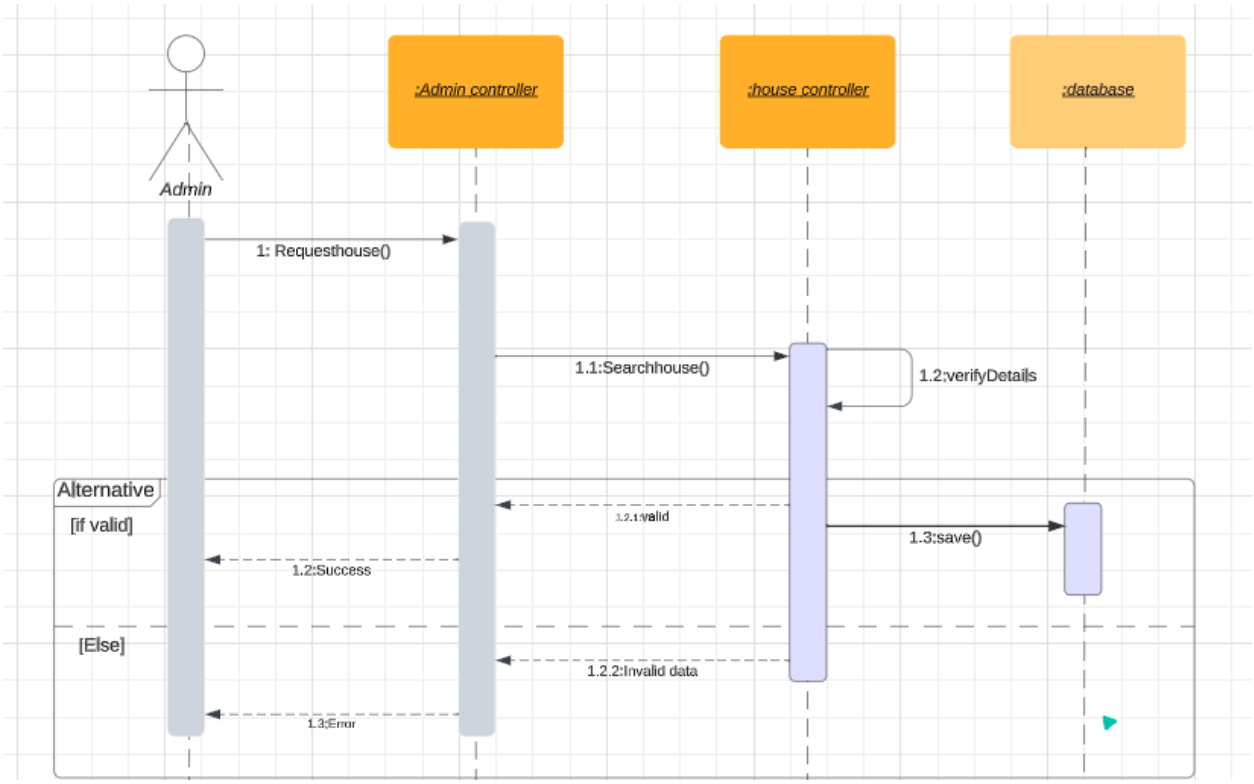
1.13: View bills



1.14: Sell house

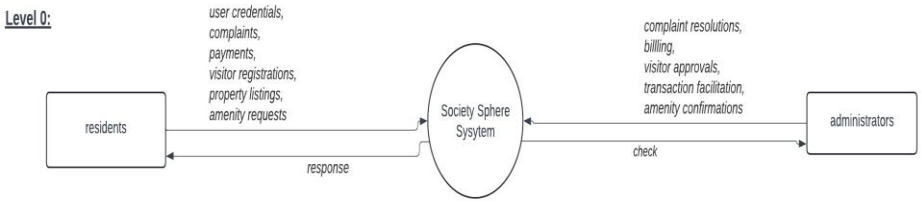


1.15: Buy house

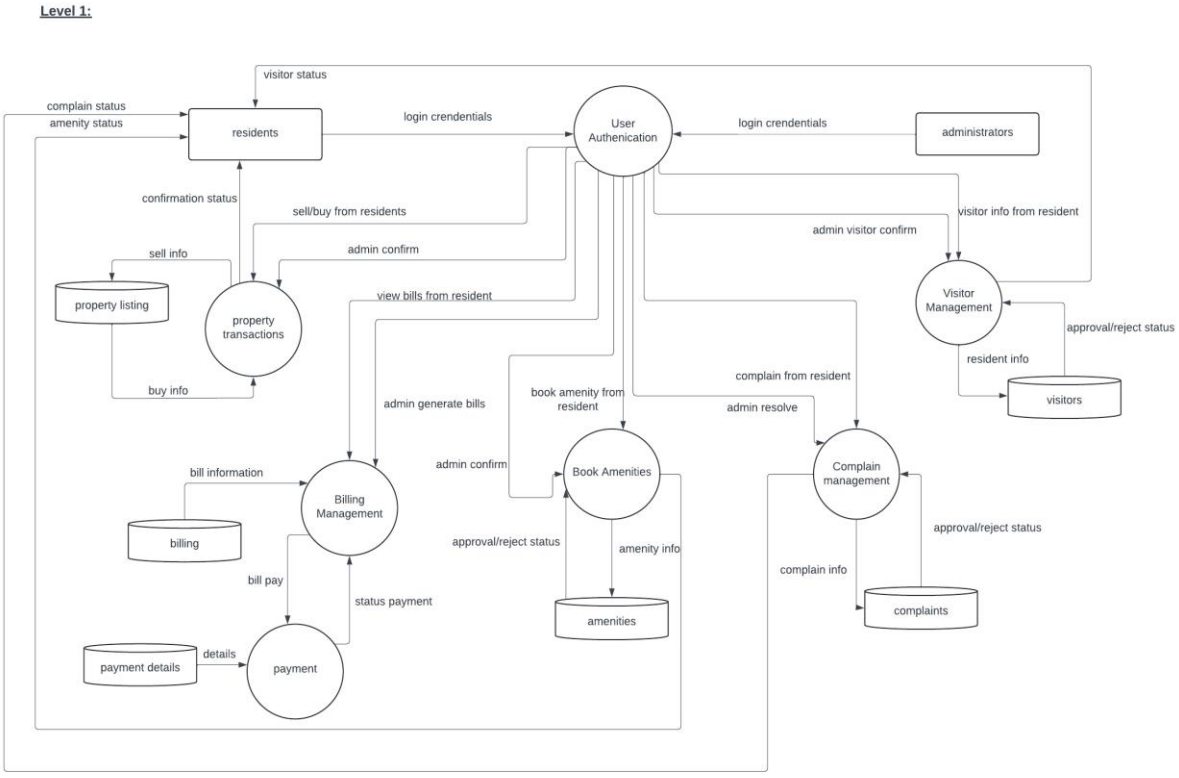


DATA FLOW DIAGRAM

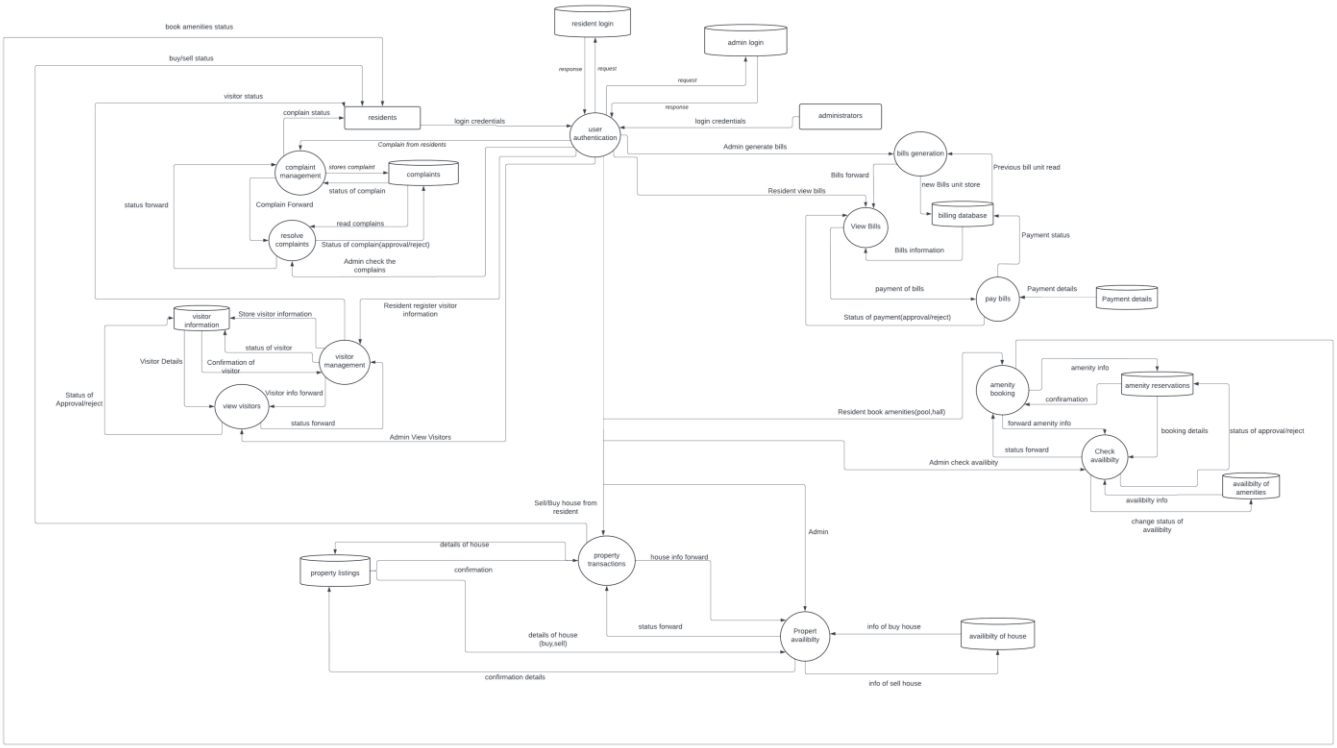
LEVEL 0:



LEVEL 1:



LEVEL 2:



Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>