CommUni3y

COMMUNITY MANAGEMENT SYSTEM

SOFTWARE DESIGN AND ANALYSIS PROJECT

Your All-in-One Solution for Community Harmony

PRESENTED BY

Ahmed Mustafa	22i-2301
Awais Khalid	22i-2355
Umer Ali	22i-2374

1. Introduction

1.1 Purpose

This document outlines the software requirements and design specifications for the **Community Management System (CMS)**. The CMS is intended to streamline the management of residential communities, facilitating the automation of various tasks like payments, maintenance requests, event management, parking booking, and notifications. This system also aims to enable admins to oversee and manage all resident interactions effectively.

1.2 Product Scope

The Community Management System is designed to provide an integrated platform for the management of a residential community. The system allows residents to submit service requests, make payments, RSVP for events, book parking spaces, and receive notifications. Administrators will manage all these tasks, including monitoring payments, surveillance checking, creating events, managing maintenance requests, and sending alerts.

1.3 Title

Community Management System:
A Platform for Automating and Streamlining Community Management Tasks

1.4 Objectives

• Enable residents to make payments, book parking, submit maintenance requests, and RSVP for events.
• Provide an interface for administrators to manage resident data, payments, events, maintenance requests, and notifications.
• Automate notifications for residents regarding events, payment statuses, and maintenance requests.
• Implement role-based access control with admins overseeing the system and residents using it for day-to-day operations.

1.5 Problem Statement

Managing a community manually leads to inefficiencies, miscommunication, and lack of transparency. The **Community Management System** addresses these issues by automating various tasks such as payment processing, service requests, parking bookings, and event management. This system reduces administrative overhead, minimizes errors, and enhances communication between administrators and residents.

2. Overall Description

2.1 Product Perspective

The CMS is a standalone software solution designed to simplify and automate the management of community-related tasks. It interacts with a SQL database for data storage and retrieval. This product will replace traditional manual methods of managing payments, events, maintenance requests, etc. The system's key components include resident and admin modules, payment processing, parking management, event handling, and notification delivery.

2.2 Product Functions

□ **Resident Features**:

- Make payments
- Submit maintenance requests
- RSVP for events
- Book parking spaces
- Receive automated notifications

□ **Admin Features**:

- Manage resident data
- Manage Surveillance data
- Approve maintenance requests
- Create and manage events
- Manage Visitors data
- Track and manage payments
- Send notifications

2.3 List of Use Cases

□ **Resident Use Cases**:

- Make Payment
- Book Parking
- Submit Maintenance Request
- RSVP for Event
- Receive Notification
- Manage Bills

□ **Admin Use Cases**:

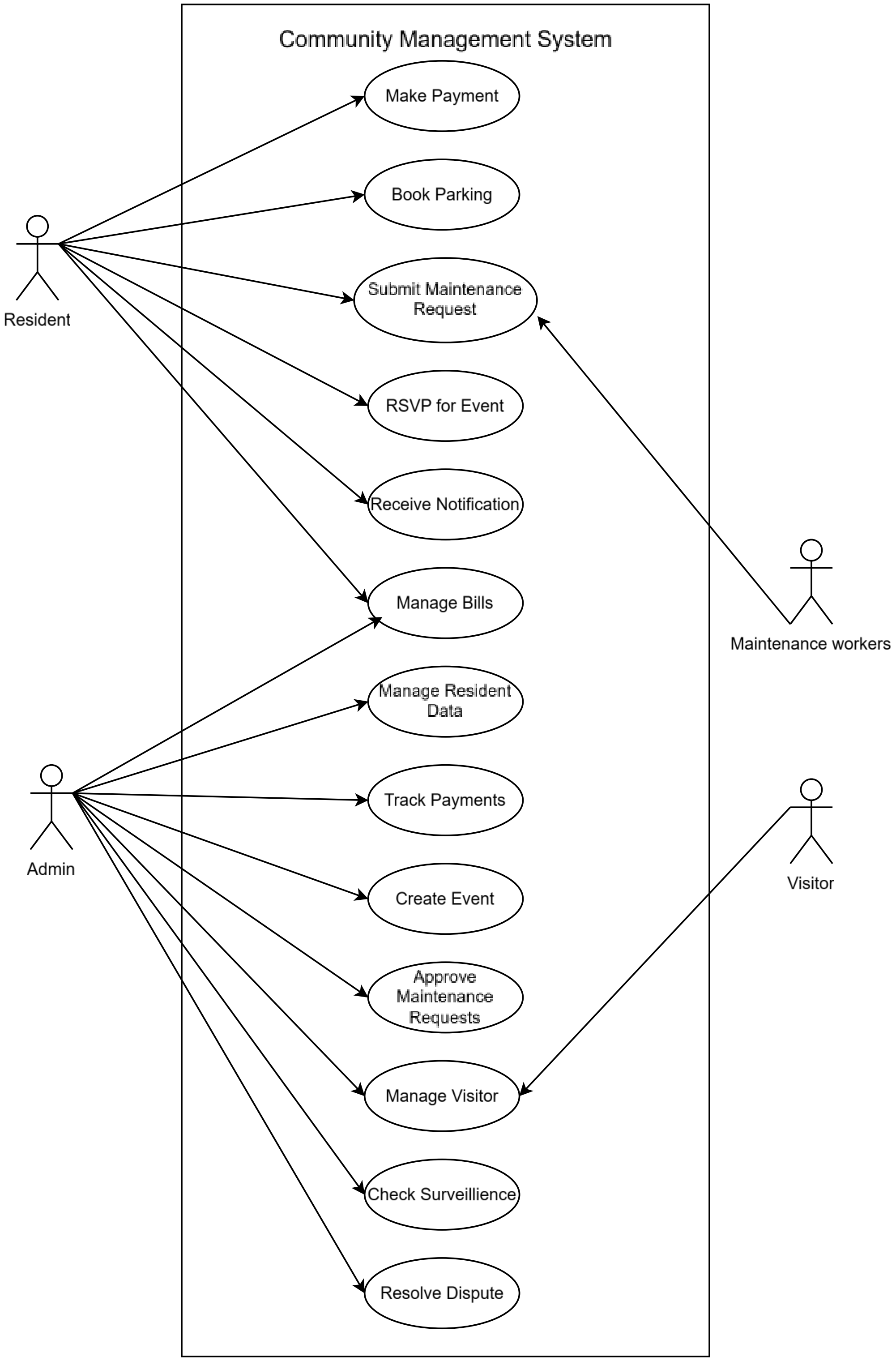
- Manage Resident Data
- Track Payments
- Update Payments
- Create Event
- Manage Events
- Adjust Bills
- Approve Maintenance Requests

- Send Notifications

2.4 Extended Use Cases

- **Make Payment**: A resident logs in, selects the payment method, and enters the required details (amount, payment method). The system validates the payment and updates the payment records in the database.
- **Book Parking**: A resident searches for available parking slots, selects a slot, and confirms the booking. The system checks availability and updates the parking slot status.
- **Submit Maintenance Request**: A resident submits a maintenance request, specifying the location, urgency, and description. The system stores the request for admin review and approval.

2.5 Use Case Diagram



3. Other Nonfunctional Requirements

3.1 Performance Requirements

- The system should handle up to 500 concurrent users with minimal performance degradation.
- The response time for any transaction (payment, parking booking, etc.) should not exceed 5 seconds.

3.2 Safety Requirements

- The system must ensure that all sensitive data such as payment details and personal information are securely transmitted and stored.
- Error messages must be clear and avoid revealing sensitive information.

3.3 Security Requirements

- User authentication will be required for residents and admins.
- All passwords will be stored using encryption techniques.
- The system must implement role-based access control to restrict functionalities based on user roles (Resident vs Admin).

3.4 Software Quality Attributes

- **Reliability**: The system should be available 99% of the time with no major disruptions.
- **Usability**: The system must be easy to navigate for both residents and admins, with clear instructions and a user-friendly interface.
- **Scalability**: The system should be able to scale with an increasing number of residents and admin tasks.

3.5 Business Rules

- Residents must be authenticated before submitting payments or requests.
- Only admins can approve maintenance requests.
- Admins can create events and send notifications, but only after proper validation of resident participation.

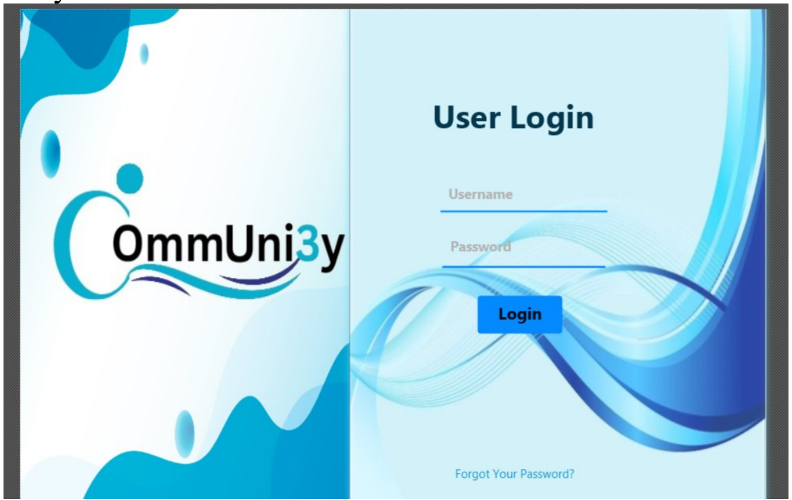
3.6 Operating Environment

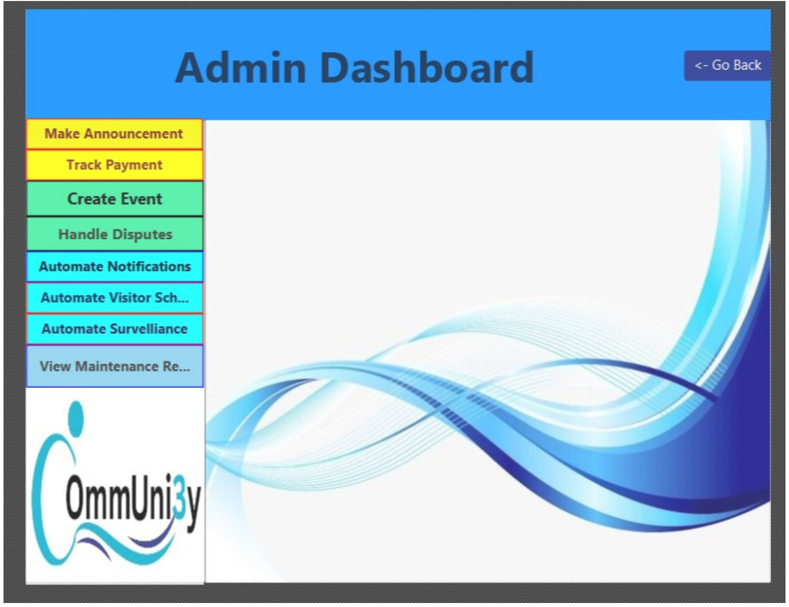
- The system will run on a standard server environment with SQL Management Server as the backend database.
- Supported browsers include Chrome, Firefox, and Edge.
- Java IDE (for Project Based only)

3.7 User Interfaces

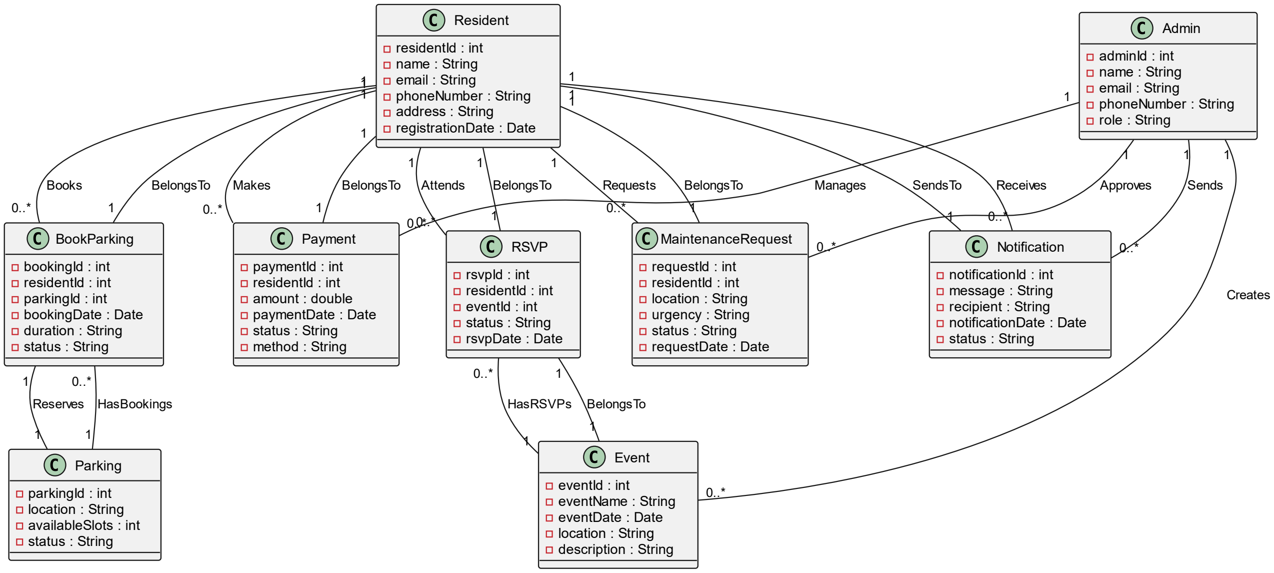
- **Admin Dashboard**: Displays options for managing payments, residents, events, and maintenance.
- **Resident Dashboard**: Allows residents to perform tasks like making payments, submitting requests, and RSVPing to events.
- **Login Center**: A section for user to login their credententials.

They are attached below:



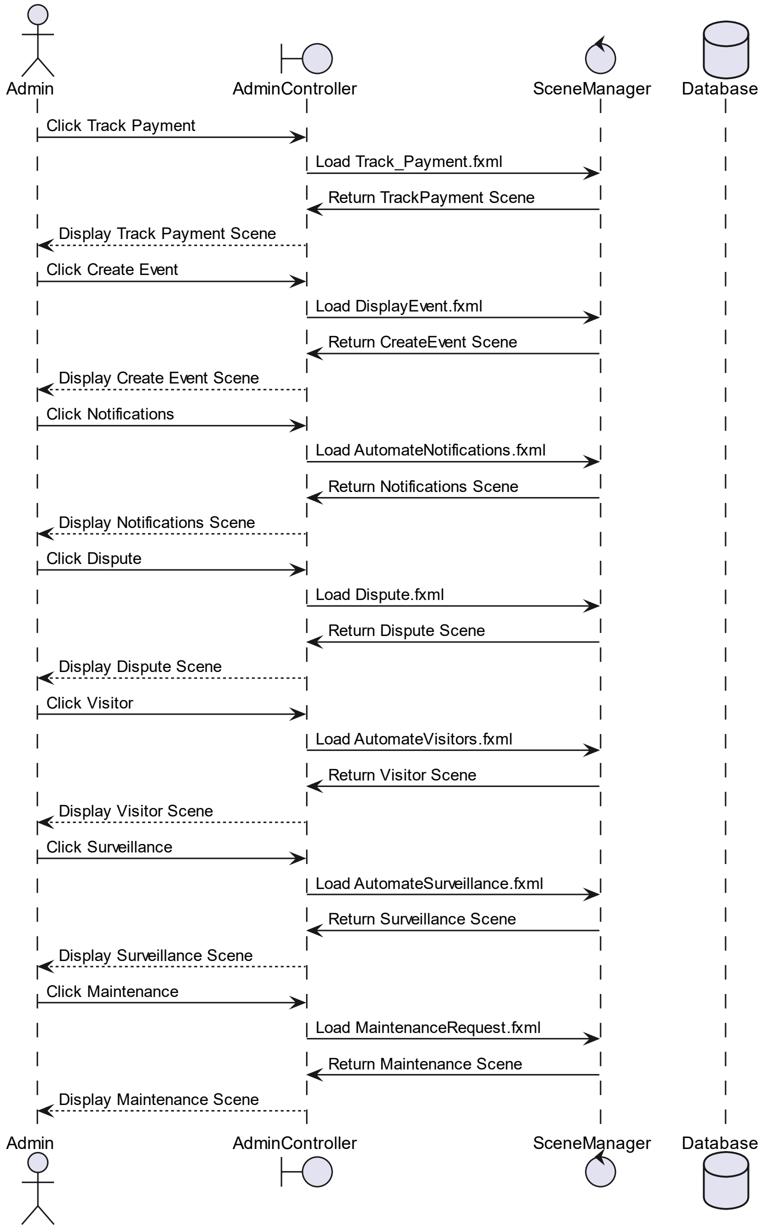


4. Domain Model

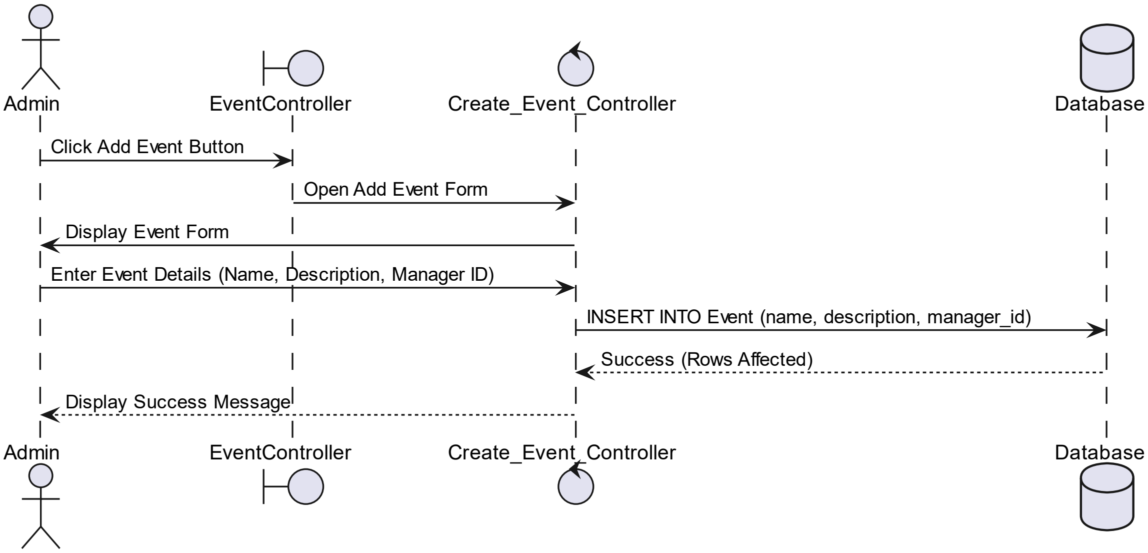


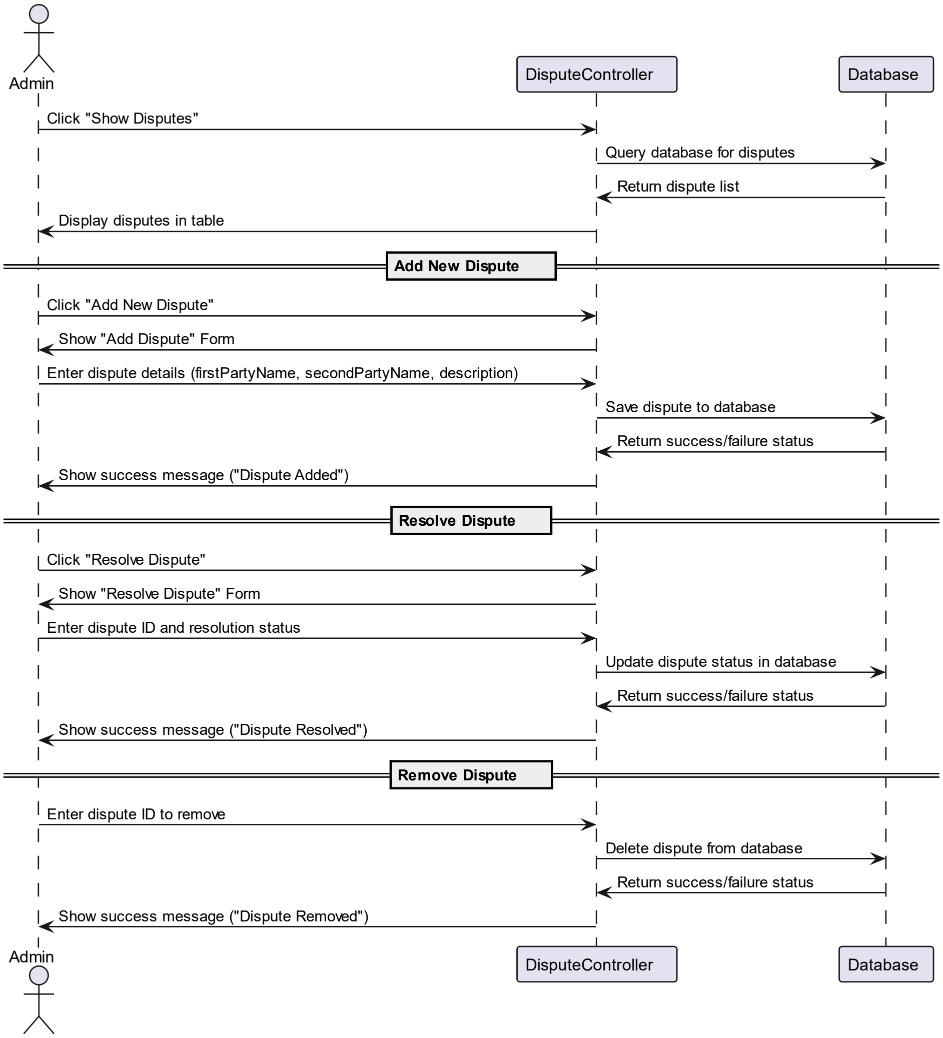
5. System Sequence Diagram

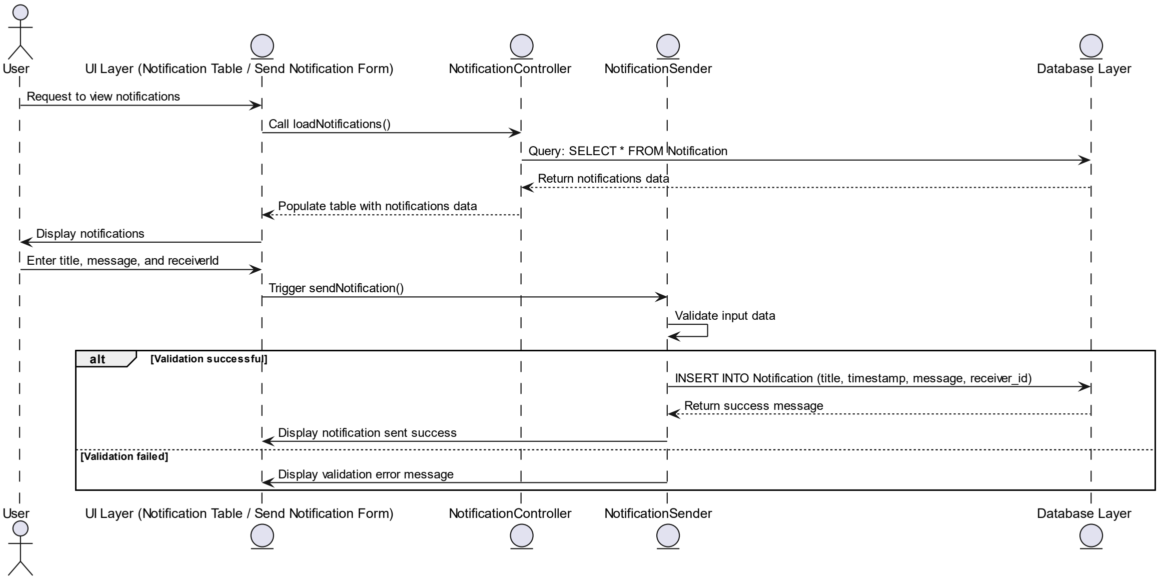
For Admin:



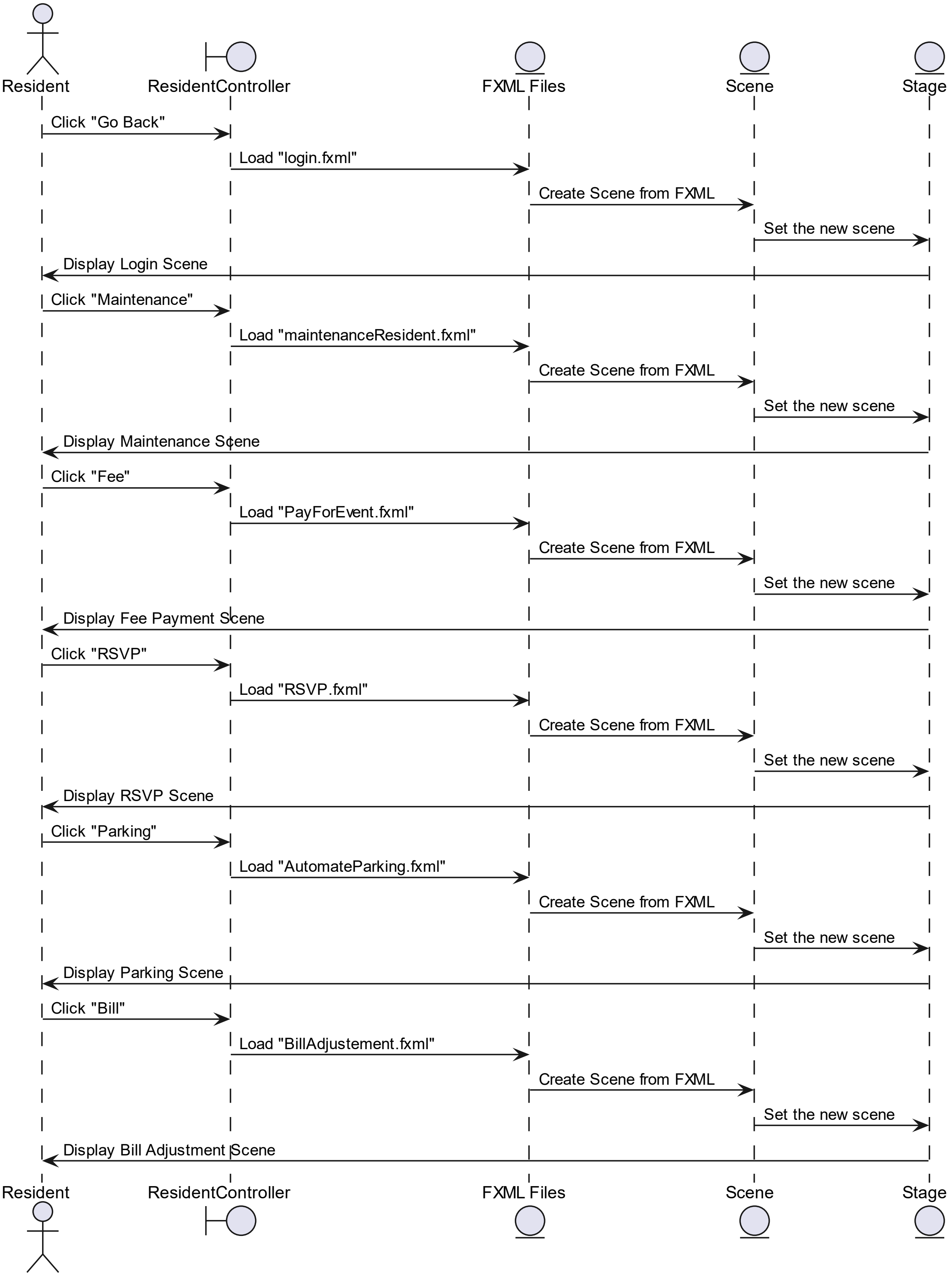
And its use cases



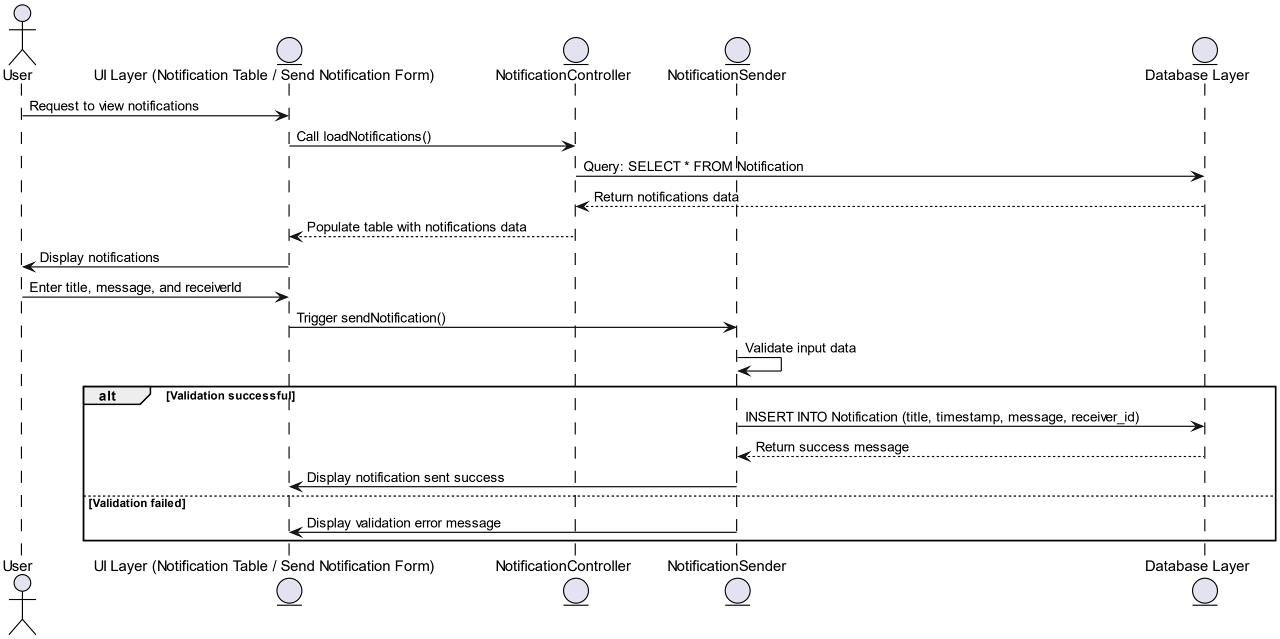

Admin
DisputeController
Database
Click "Show Disputes"
Query database for disputes
Return dispute list
Display disputes in table
Add New Dispute
Click "Add New Dispute"
Show "Add Dispute" Form
Enter dispute details (firstPartyName, secondPartyName, description)
Save dispute to database
Return success/failure status
Show success message ("Dispute Added")
Resolve Dispute
Click "Resolve Dispute"
Show "Resolve Dispute" Form
Enter dispute ID and resolution status
Update dispute status in database
Return success/failure status
Show success message ("Dispute Resolved")
Remove Dispute
Enter dispute ID to remove
Delete dispute from database
Return success/failure status
Show success message ("Dispute Removed")
Admin
DisputeController
Database

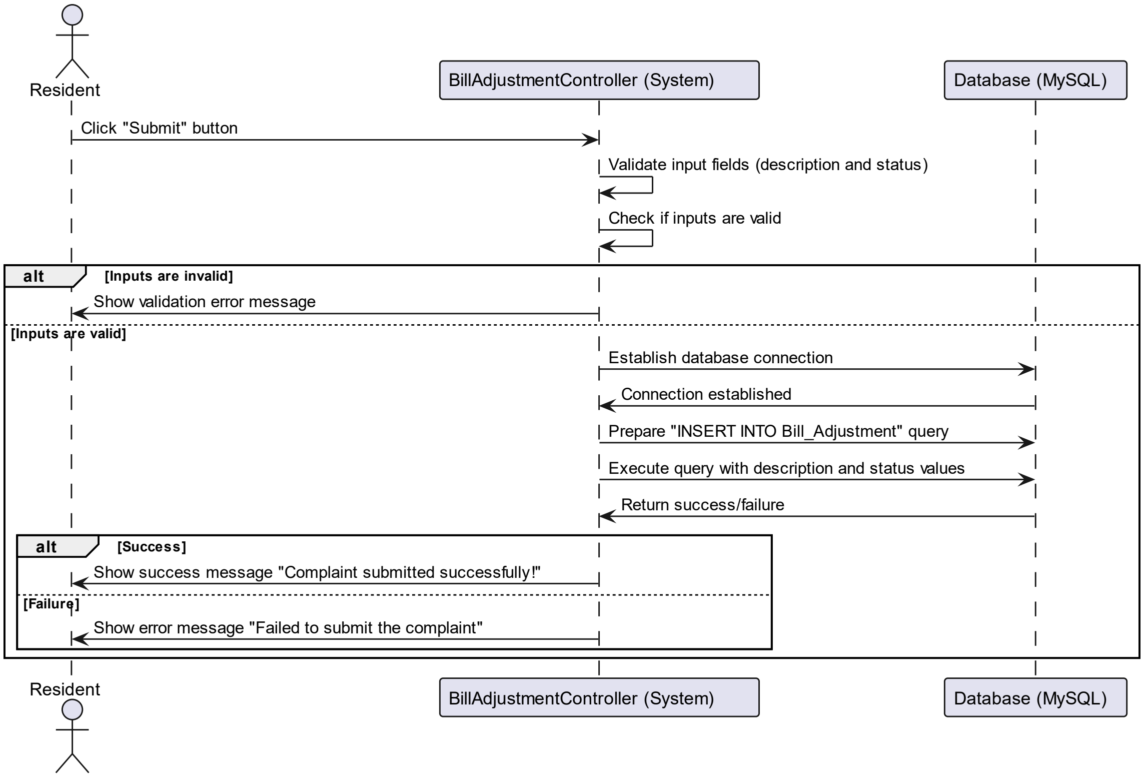

User
UI Layer (Notification Table / Send Notification Form)
NotificationController
NotificationSender
Database Layer
Request to view notifications
Call loadNotifications()
Query: SELECT * FROM Notification
Return notifications data
Populate table with notifications data
Display notifications
Enter title, message, and receiverId
Trigger sendNotification()
Validate input data
alt
[Validation successful]
INSERT INTO Notification (title, timestamp, message, receiver_id)
Return success message
Display notification sent success
[Validation failed]
Display validation error message
User
UI Layer (Notification Table / Send Notification Form)
NotificationController
NotificationSender
Database Layer

For Resident:



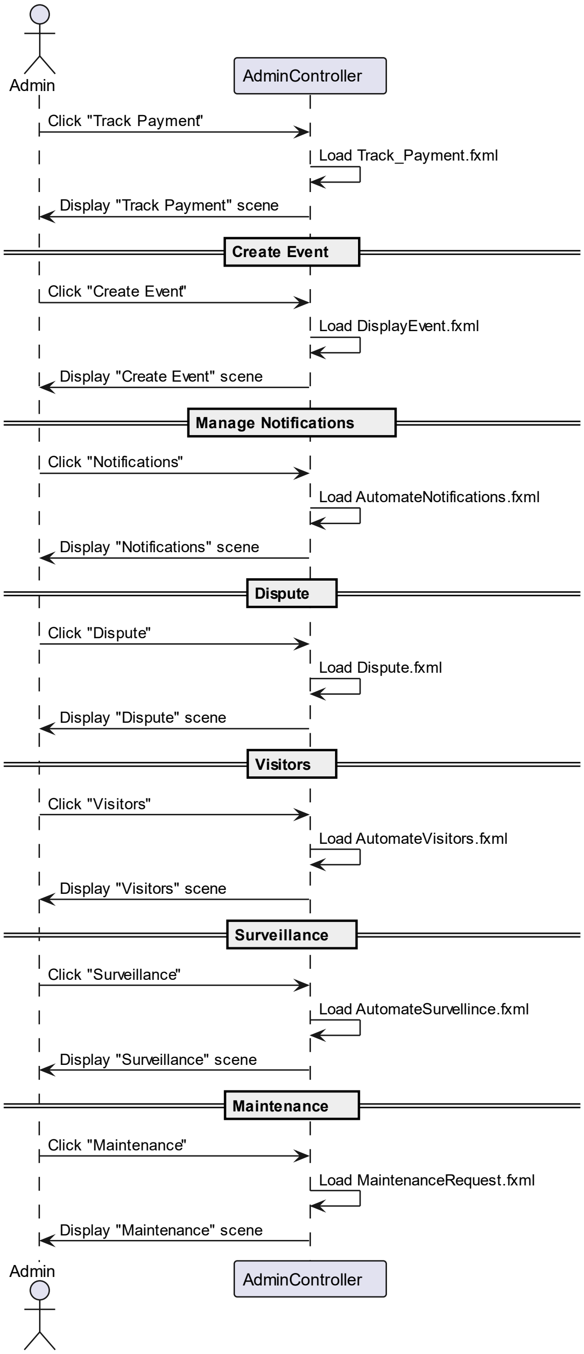
and its use cases:

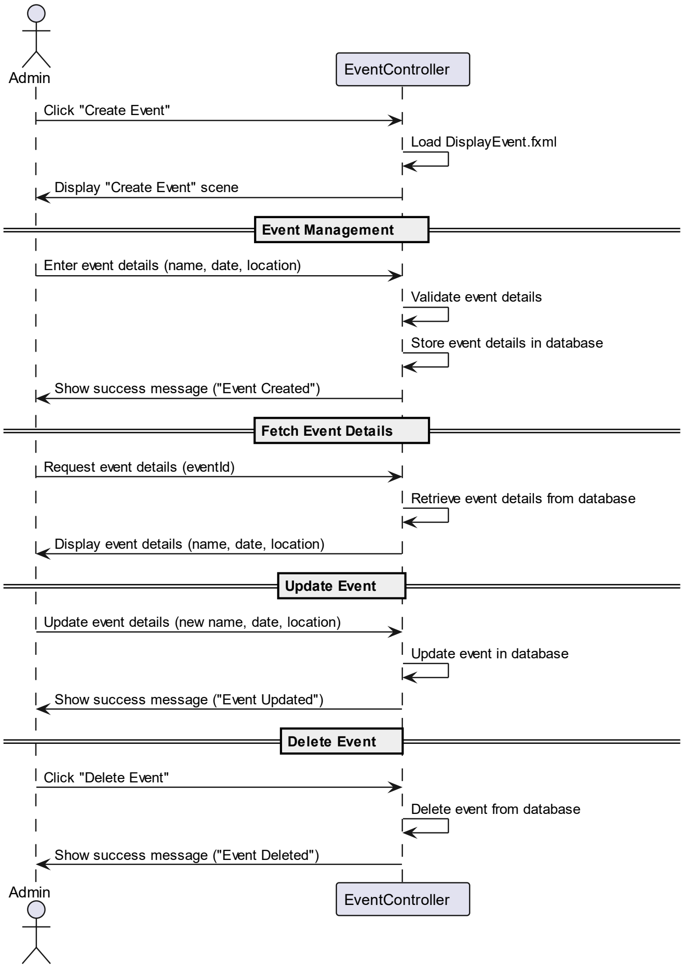

User
UI Layer (Notification Table / Send Notification Form)
NotificationController
NotificationSender
Database Layer
Request to view notifications
Call loadNotifications()
Query: SELECT * FROM Notification
Return notifications data
Populate table with notifications data
Display notifications
Enter title, message, and receiverId
Trigger sendNotification()
Validate input data
alt
[Validation successful]
INSERT INTO Notification (title, timestamp, message, receiver_id)
Return success message
Display notification sent success
[Validation failed]
Display validation error message
User
UI Layer (Notification Table / Send Notification Form)
NotificationController
NotificationSender
Database Layer

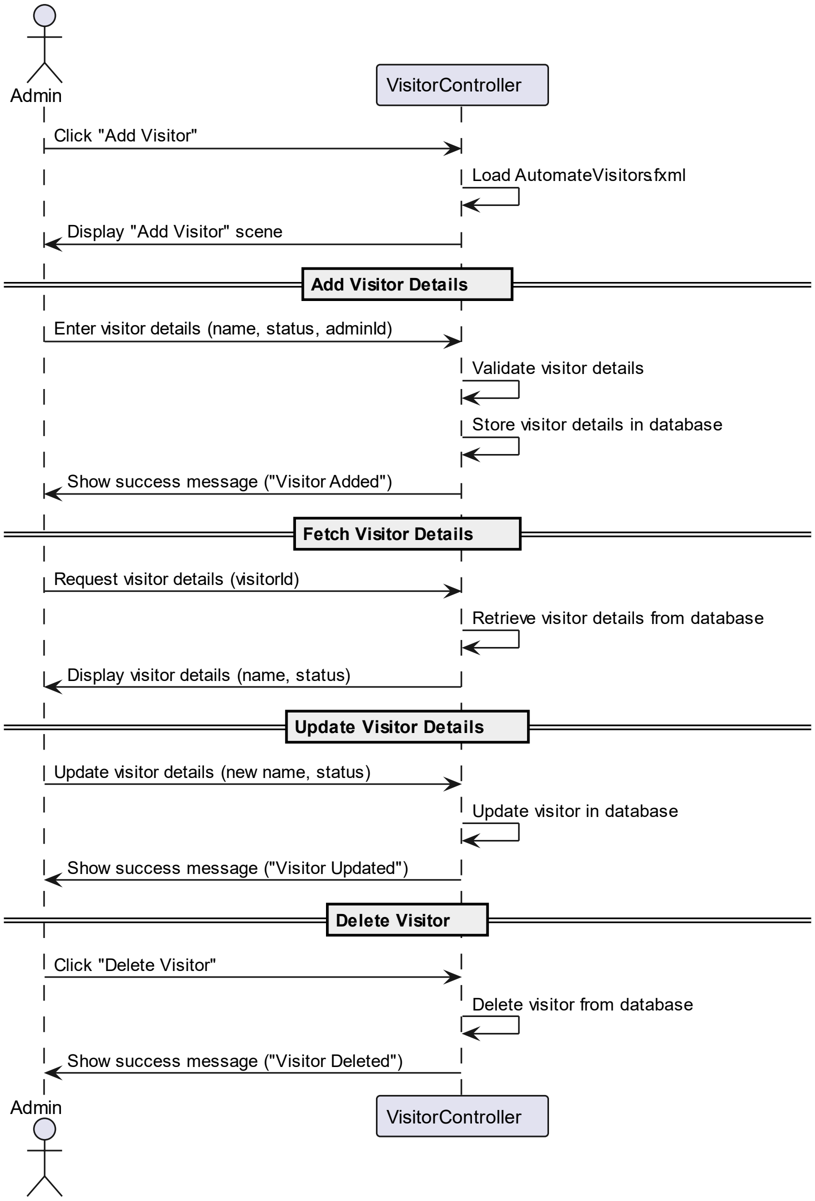

Resident
BillAdjustmentController (System)
Database (MySQL)
Click "Submit" button
Validate input fields (description and status)
Check if inputs are valid
alt
[Inputs are invalid]
Show validation error message
[Inputs are valid]
Establish database connection
Connection established
Prepare "INSERT INTO Bill_Adjustment" query
Execute query with description and status values
Return success/failure
alt
[Success]
Show success message "Complaint submitted successfully!"
[Failure]
Show error message "Failed to submit the complaint"
Resident
BillAdjustmentController (System)
Database (MySQL)

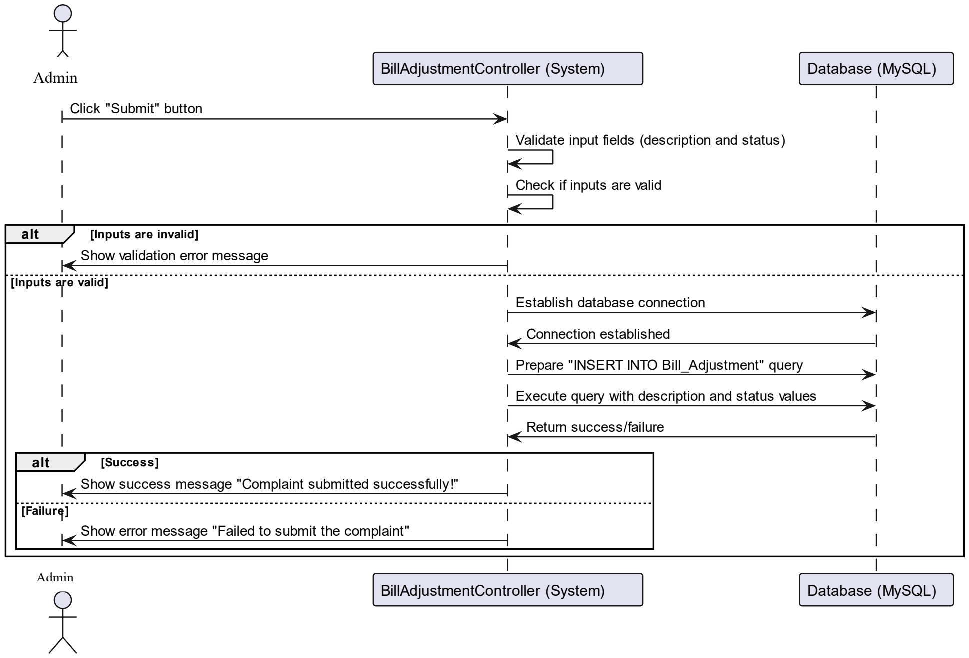
6. Sequence Diagram

For Admin:

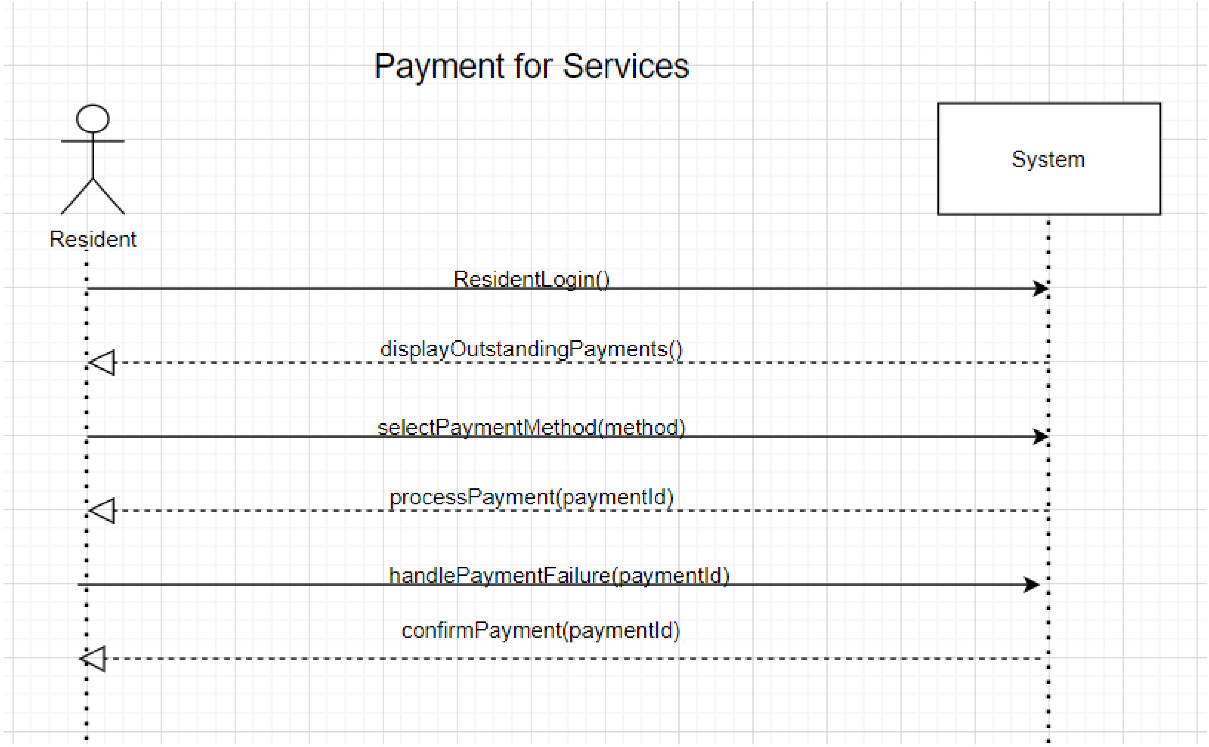


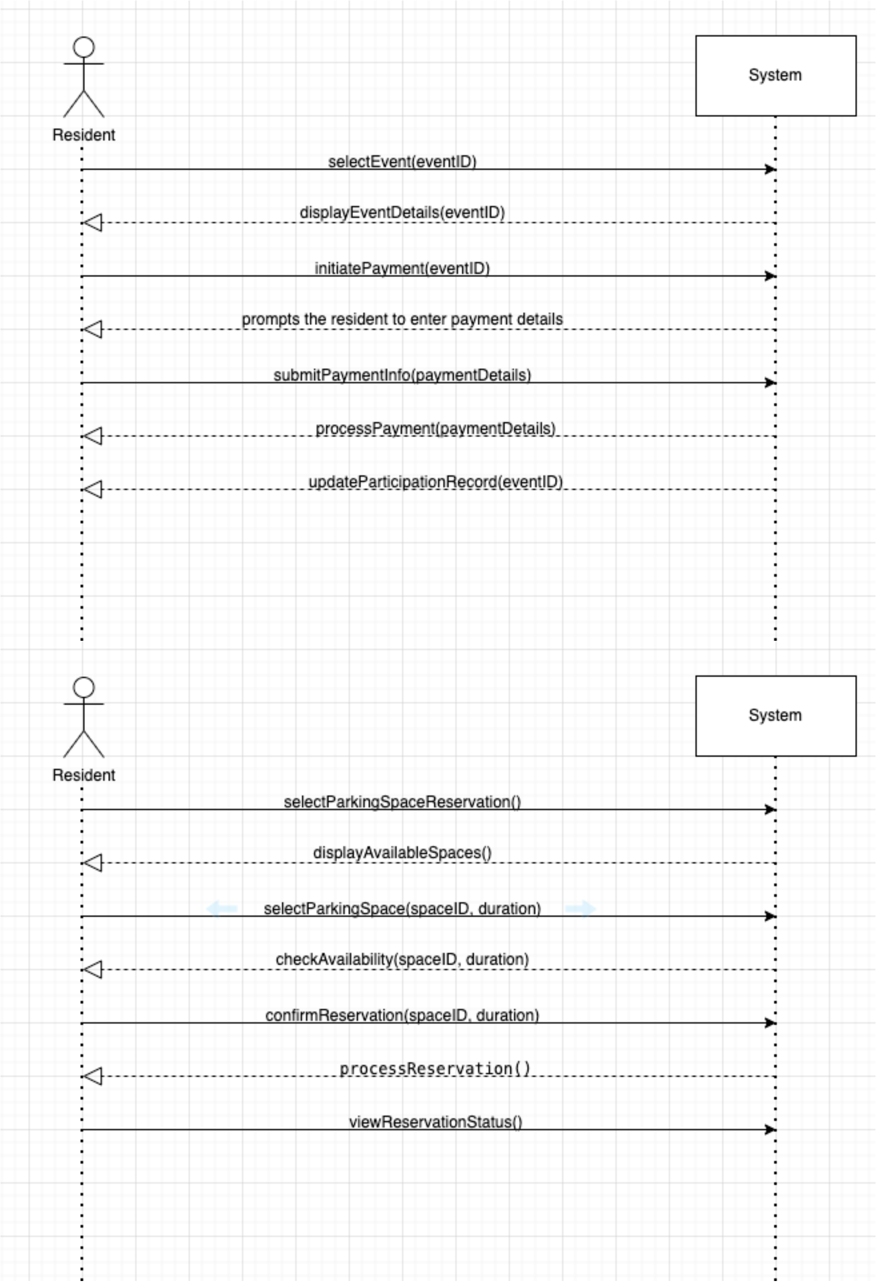

Admin
EventController
Click "Create Event"
Load DisplayEvent.fxml
Display "Create Event" scene
Event Management
Enter event details (name, date, location)
Validate event details
Store event details in database
Show success message ("Event Created")
Fetch Event Details
Request event details (eventId)
Retrieve event details from database
Display event details (name, date, location)
Update Event
Update event details (new name, date, location)
Update event in database
Show success message ("Event Updated")
Delete Event
Click "Delete Event"
Delete event from database
Show success message ("Event Deleted")
Admin
EventController

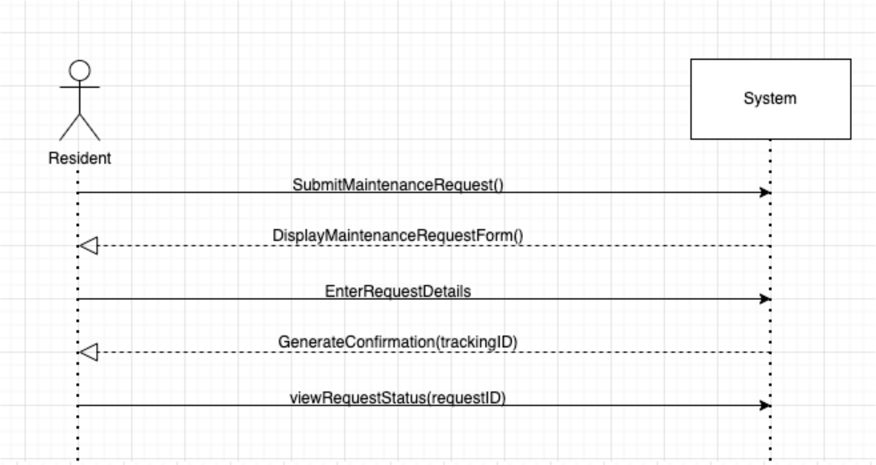

Admin
VisitorController
Click "Add Visitor"
Load AutomateVisitors.fxml
Display "Add Visitor" scene
Add Visitor Details
Enter visitor details (name, status, adminId)
Validate visitor details
Store visitor details in database
Show success message ("Visitor Added")
Fetch Visitor Details
Request visitor details (visitorId)
Retrieve visitor details from database
Display visitor details (name, status)
Update Visitor Details
Update visitor details (new name, status)
Update visitor in database
Show success message ("Visitor Updated")
Delete Visitor
Click "Delete Visitor"
Delete visitor from database
Show success message ("Visitor Deleted")
Admin
VisitorController

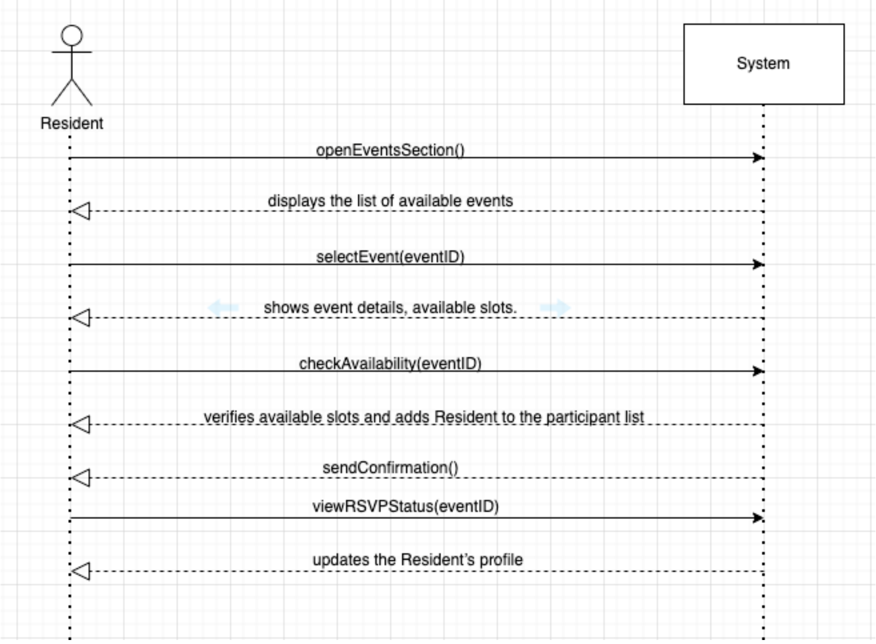

Admin
BillAdjustmentController (System)
Database (MySQL)
Click "Submit" button
Validate input fields (description and status)
Check if inputs are valid
alt
[Inputs are invalid]
Show validation error message
[Inputs are valid]
Establish database connection
Connection established
Prepare "INSERT INTO Bill_Adjustment" query
Execute query with description and status values
Return success/failure
alt
[Success]
Show success message "Complaint submitted successfully!"
[Failure]
Show error message "Failed to submit the complaint"
Admin
BillAdjustmentController (System)
Database (MySQL)

For Resident:

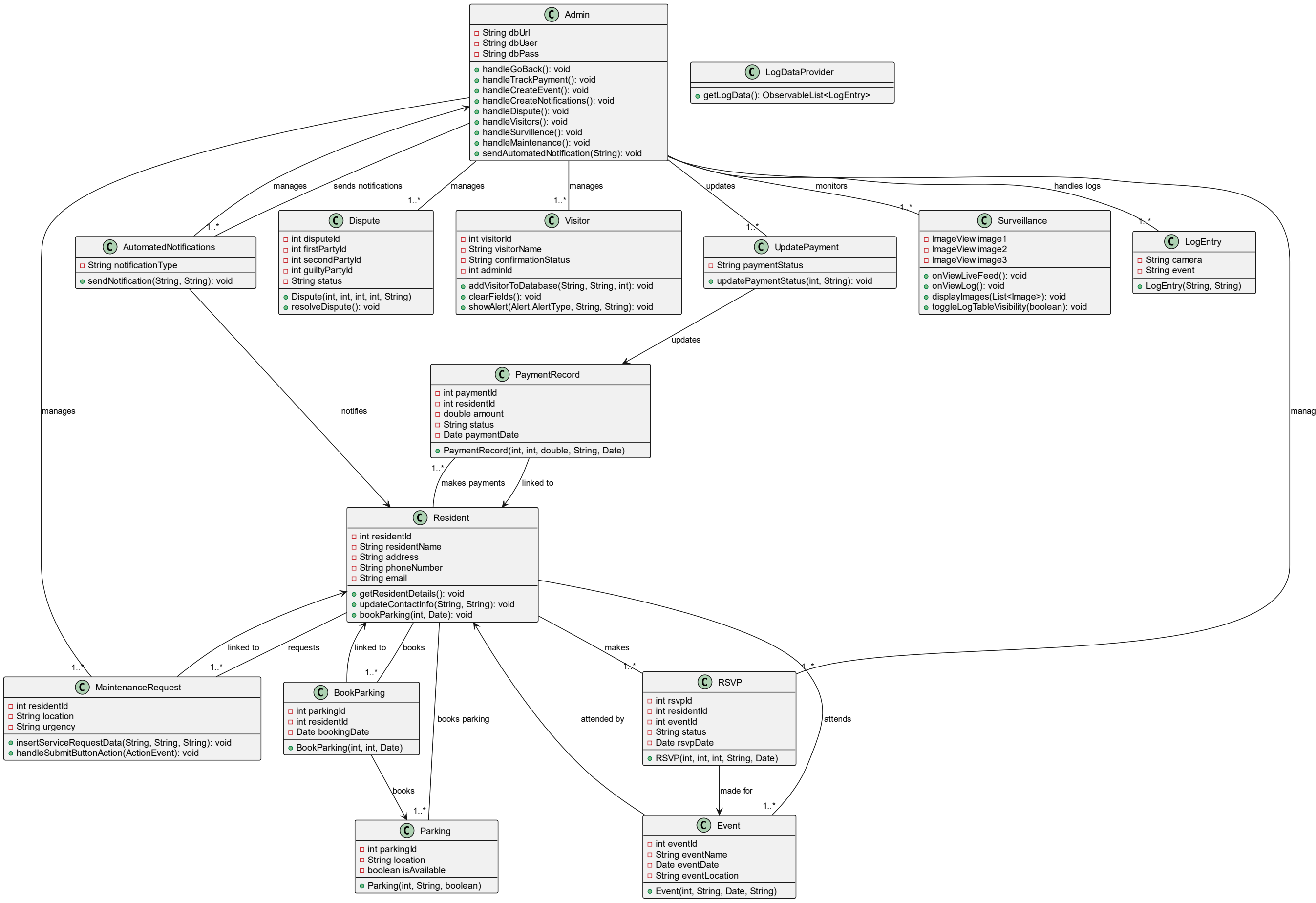

Payment for Services
Resident
System
ResidentLogin()
displayOutstandingPayments()
selectPaymentMethod(method)
processPayment(paymentId)
handlePaymentFailure(paymentId)
confirmPayment(paymentId)


Resident
System
selectEvent(eventID)
displayEventDetails(eventID)
initiatePayment(eventID)
prompts the resident to enter payment details
submitPaymentInfo(paymentDetails)
processPayment(paymentDetails)
updateParticipationRecord(eventID)
Resident
System
selectParkingSpaceReservation()
displayAvailableSpaces()
selectParkingSpace(spaceID, duration)
checkAvailability(spaceID, duration)
confirmReservation(spaceID, duration)
processReservation()
viewReservationStatus()

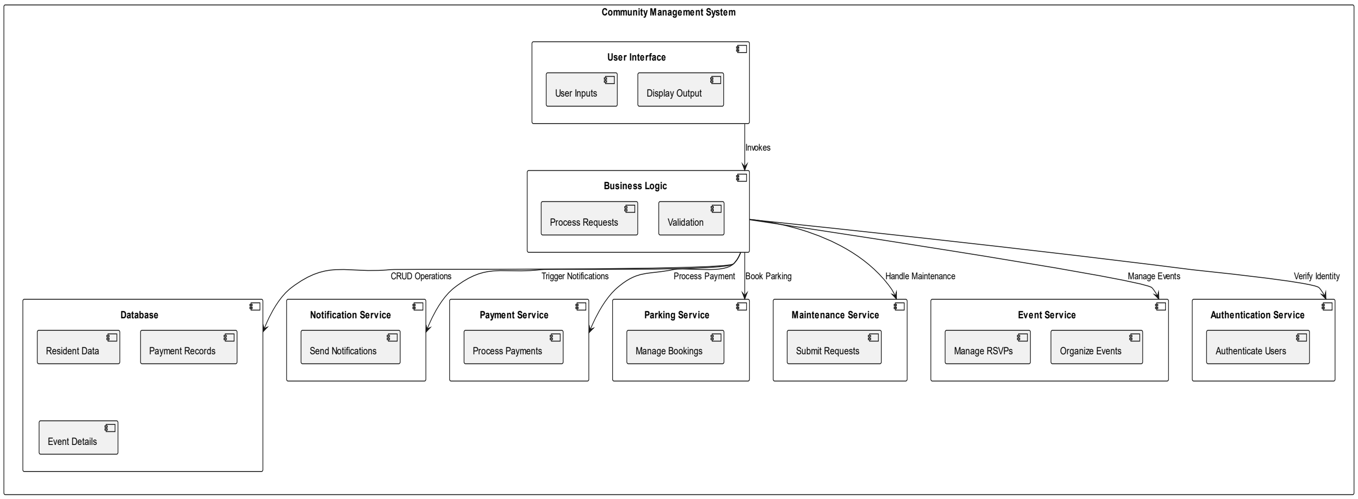

Resident
System
SubmitMaintenanceRequest()
DisplayMaintenanceRequestForm()
EnterRequestDetails
GenerateConfirmation(trackingID)
viewRequestStatus(requestID)


Resident
System
openEventsSection()
displays the list of available events
selectEvent(eventID)
shows event details, available slots.
checkAvailability(eventID)
verifies available slots and adds Resident to the participant list
sendConfirmation()
viewRSVPStatus(eventID)
updates the Resident's profile

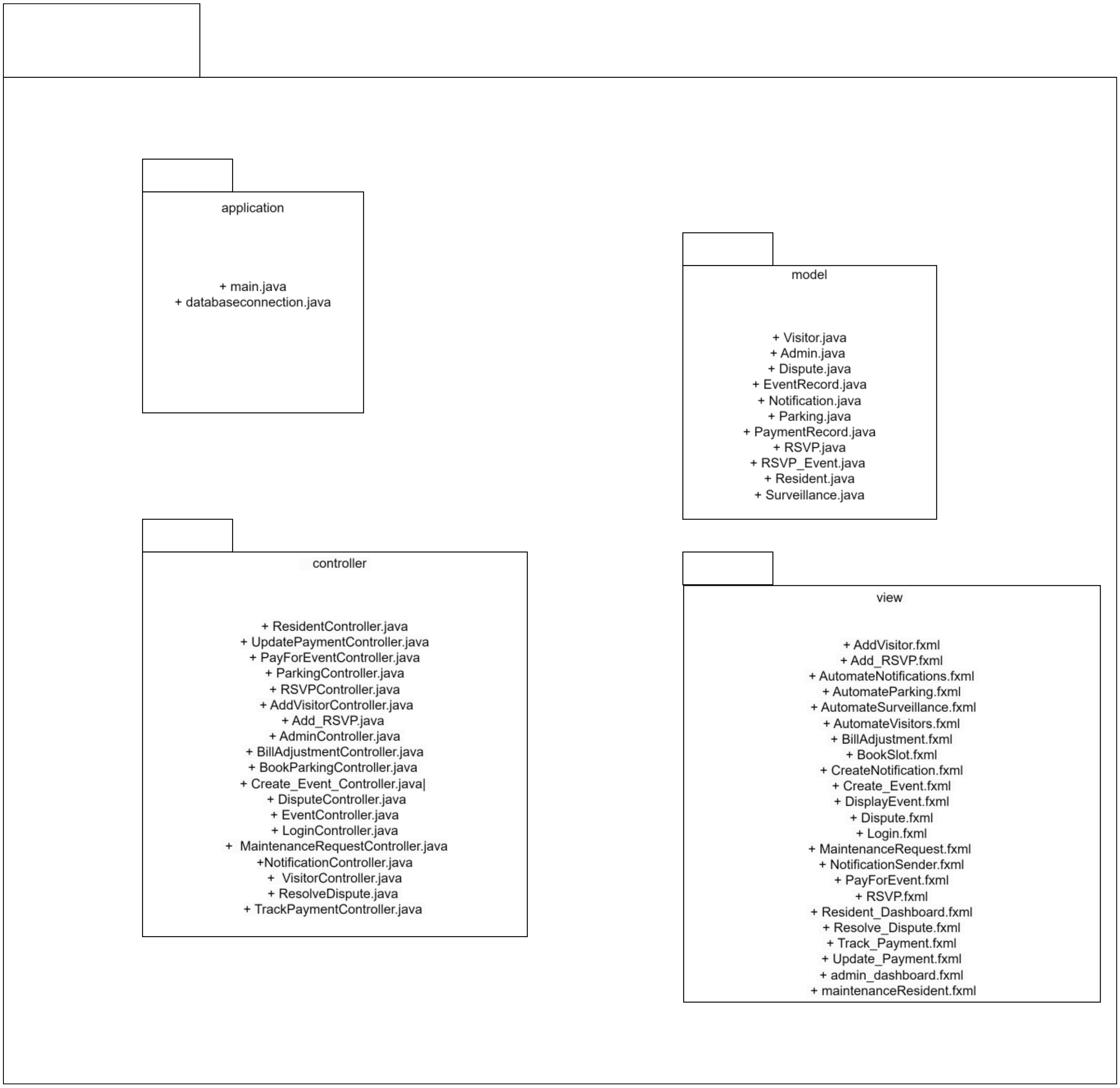
7. Class Diagram



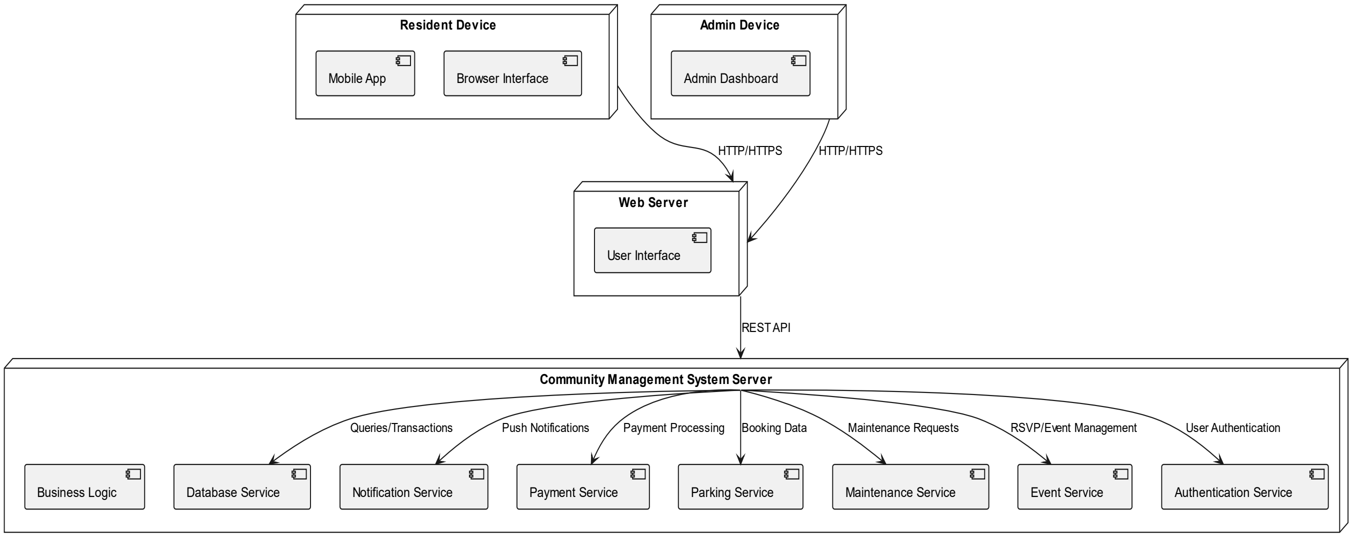
8. Component Diagram



9. Package Diagram



10. Deployment Diagram


Resident Device
Mobile App
Browser Interface
Admin Device
Admin Dashboard
HTTP/HTTPS
HTTP/HTTPS
Web Server
User Interface
REST API
Community Management System Server
Queries/Transactions
Push Notifications
Payment Processing
Booking Data
Maintenance Requests
RSVP/Event Management
User Authentication
Business Logic
Database Service
Notification Service
Payment Service
Parking Service
Maintenance Service
Event Service
Authentication Service