MULTIMEDIA UNIVERSITY

Software Requirements Specification

Campus Ride-Sharing Platform With Parking System Integration

CSE6224 Software Requirements Engineering (TT2L)

Group Name:

Halin Roychana A/P I Roi	**1211108961**
Noor Azwani Binti Shamsudin	**1211109188**
Nur Aqilah Tan	**241UC2407T**
Farhana Adnin Binti Ahmad Effendi	**242UC2451X**

Date: 25 MAY 2025

CONTENTS

1 Introduction

1.1 Purpose

The goal of the Campus Ride-Sharing Platform with Parking System Integration is to give MMU employees and students a centralized, effective way to manage parking and ride-sharing. This system works by requiring users to either start or join a carpool group before they can book a parking space on campus as one of the ways on how the platform encourages carpooling.

The goals of this integrated system are to lessen traffic jams, make the best use of the few parking spots available, and promote eco-friendly transportation. Only from a single platform users can organize carpool arrangements, monitor real-time parking availability, make reservations and payments, and securely log in using their university credentials. The system promotes campus sustainability goals, improves user convenience, and streamlines administrative procedures associated with campus transportation by fusing ride-sharing coordination with intelligent parking features.

1.2 Scope

This system will handle booking, approval, and tracking of campus carpool and parking reservations. It allows authenticated university members the capability to create or join carpool groups, and only upon carpool approval, parking spot reservation by real-time availability in designated campus zones, are enabled only after carpool approval.

Key Features:

- Secure Authentication: University ID login ensures access is limited to authorized members.
- Flexible Carpooling: Enables users to create new carpool groups or join existing ones, with approval workflows managed by group leaders and system administrators.
- Real-time Parking Management: Displays live parking availability in designated campus zones, allowing approved carpools to make reservations.
- Comprehensive Administration: Provides an administrator panel for efficient management of requests and proactive system monitoring.
- Enhanced User Experience: Incorporates notifications and feedback mechanisms for quality tracking and continuous improvement.

The website is limited to the campus community and does not support third-party or off-campus services.

1.3 Product overview

The Campus Ride-Sharing Platform with Parking System Integration is a secure web application that is designed to serve the university community, including students, faculty, and staff. The platform encourages green transportation by allowing users to create or join carpooling groups, reducing single-occupancy vehicle trips to campus. Once a carpool has been approved, users can reserve parking spots depending on real-time parking capacity within designated campus areas. The system uses university digital ID authentication to provide access to the service by authenticated users only. It also supports an administrator dashboard for handling ride and parking requests, approving requests, and monitoring system activity. The users are also informed about booking statuses and can provide feedback after completing their rides. By combining carpool coordination and parking management, this website will streamline campus traffic flow, optimize parking capacity, and promote more sustainable commuting behavior.

1.3.1 Product Perspective

With the capability to reserve parking spaces, the Campus Ride-Sharing Platform provides a centralized application for university employees and students to arrange rides. Because of its modular design and service-oriented framework, the system may be integrated in real time with other subsystems including administrative tools, payment gateways, and parking availability tracking.By allowing users to plan their routes and handle hybrid travel (e.g., drive part way and then book a ride), the platform improves campus mobility. For dependable data handling and seamless operation, the system depends on administrative supervision, backend monitoring, and real-time synchronization.

The platform consists of four key roles:

1. **Users**
   - Login Credentials
   - Fills In Required Informations
   - Creating Carpool Group
   - Joining Carpool Group
   - Reserve Parking Spot
   - Make Payments
   - Leaves Feedback
2. **Administrator**
   - Update And Store Users Data
   - Review Carpool Request
   - Validates Ride Details
   - Approves/Reject Carpools
   - Receive Parking Approval From System Monitor
   - Update And Store Data Of Bookings Into Database
   - Reviews Comment/Feedback
3. **System Administration**
   - Update status to users
   - Retrieves availability
   - Displays list of spots to User
   - Calculates total cost (ride + parking)
   - verified payment
   - Notify User with Invoice
   - Monitors backend logs for error
4. **System Monitor**
   - Monitors Parking Data Feed
   - Confirm & Update Available Slots
   - Receives Notification Of Parking Reserve
   - Review Parking Reserve
   - Approves Parking Reserve
   - Update Data Of Parking Reservation

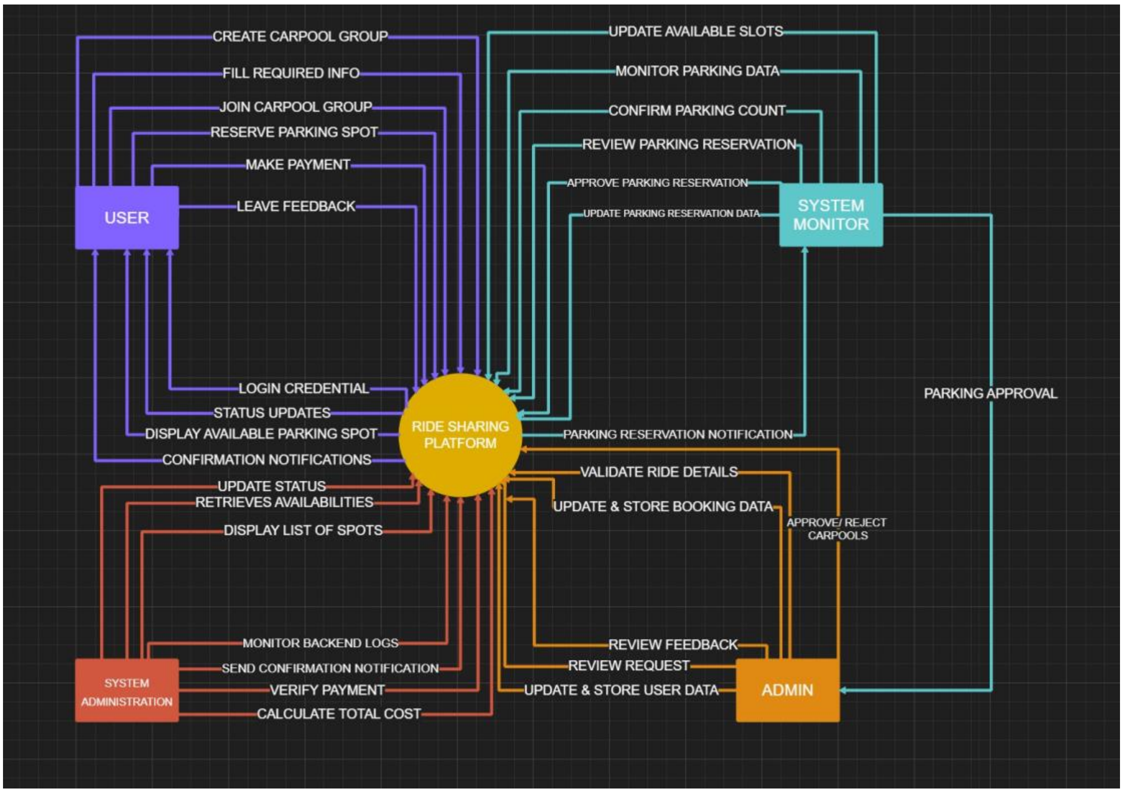


Figure 1.3.1: Context Diagram of Campus Ride Sharing Platform

1.3.1.1 System Interfaces

For the Ride-Sharing Platform to function properly, it communicates with a number of internal and external systems. These interfaces preserve system operation and guarantee appropriate module-to-module communication.

Interface	Description	Purpose
University Authentication System	The platform will integrate with the university's Single Sign-On (SSO) or student/staff login portal to authenticate users based on their University ID	Ensures that only verified MMU students and staff can access and use the platform.
Centralised Database System	Stores data related to user profile,carpool group information,parking slots,bookings and feedback	Enable persistent data storage,retrieval and management of platform-related activities
User browser	The platform provides a user-friendly and responsive web interface where users can log in, view parking availability, manage carpools, and make bookings.	Facilitates interaction between users and the system.
Web-based Admin istrator Dashboard	A secure panel for administrators to view and manage carpool requests, monitor parking lot usage, approve/reject bookings, and handle feedback	Enables efficient system management and oversight.
Notification System	Send updates to user regarding carpool status, parking reservation confirmation and feedback request	Keeps users informed about their interactions with the system in real-time.

Figure 1.3.1.1: Table of System Interface for Ride-Sharing Platform

1.3.1.2 User Interfaces

Table 1.3.1.2 shows the user interface elements for the Campus Ride-Sharing Platform and shows all the main screens for a user to interact with. Each interface is explained regarding its purpose on the platform, the required input fields, the primary button or action a user can take, and the feedback from the system after a user takes action. This helps demonstrate that the design and interfaces honor user needs and follow the system expected behavior.

Interface Name	**Description**	**Input**	**Button/Action**	**System Feedback**
Login Page	Allows user to log in using university credentials	Email/Username, Password	Log In	Error on invalid login; redirect on success.
User Profile	Allows user to complete or update personal and vehicle info.	Name, Contact, Vehicle Info, Preferences	Save / Update	"Profile Updated" message or validation errors.
Create Carpool Group	Interface to create a new carpool ride.	Ride Time, Route, Max Capacity, Vehicle Details	Submit Ride	Confirmation of request submission or error alert.
Join Carpool Group	Lets users search and request to join an active group.	Search Filters (Location, Time), Group List	Join Group	"Request Sent" or "Group Full" notifications.
Reserve Parking Spot	Displays available spots and lets user make a reservation.	Date, Time Slot, Location	Reserve Parking	"Parking Reserved" confirmation or "Slot Unavailable."
Make Payment	Interface to pay for ride/parking fees.	Card Details / FPX / Payment Method	Pay Now	"Payment Successful" or "Payment Failed."
Leave Feedback	Interface to submit feedback for past rides.	Rating (1–5), Comment	Submit Feedback	"Feedback Submitted" or error message.
Dashboard / Home Page	Main navigation panel after login;	-	Navigation Tabs (Profile,	Displays user name, carpool &

	quick links to all modules.		Book, Feedback,/ etc.)	parking status updates.
Notification Popups	Small alerts on booking, approval, and feedback responses.	-	Close / View More	Real-time updates (e.g., “Your carpool is approved.”).

Table 1.3.1.2: User Interfaces for Ride-Sharing Platform

1.3.2 Product Functions

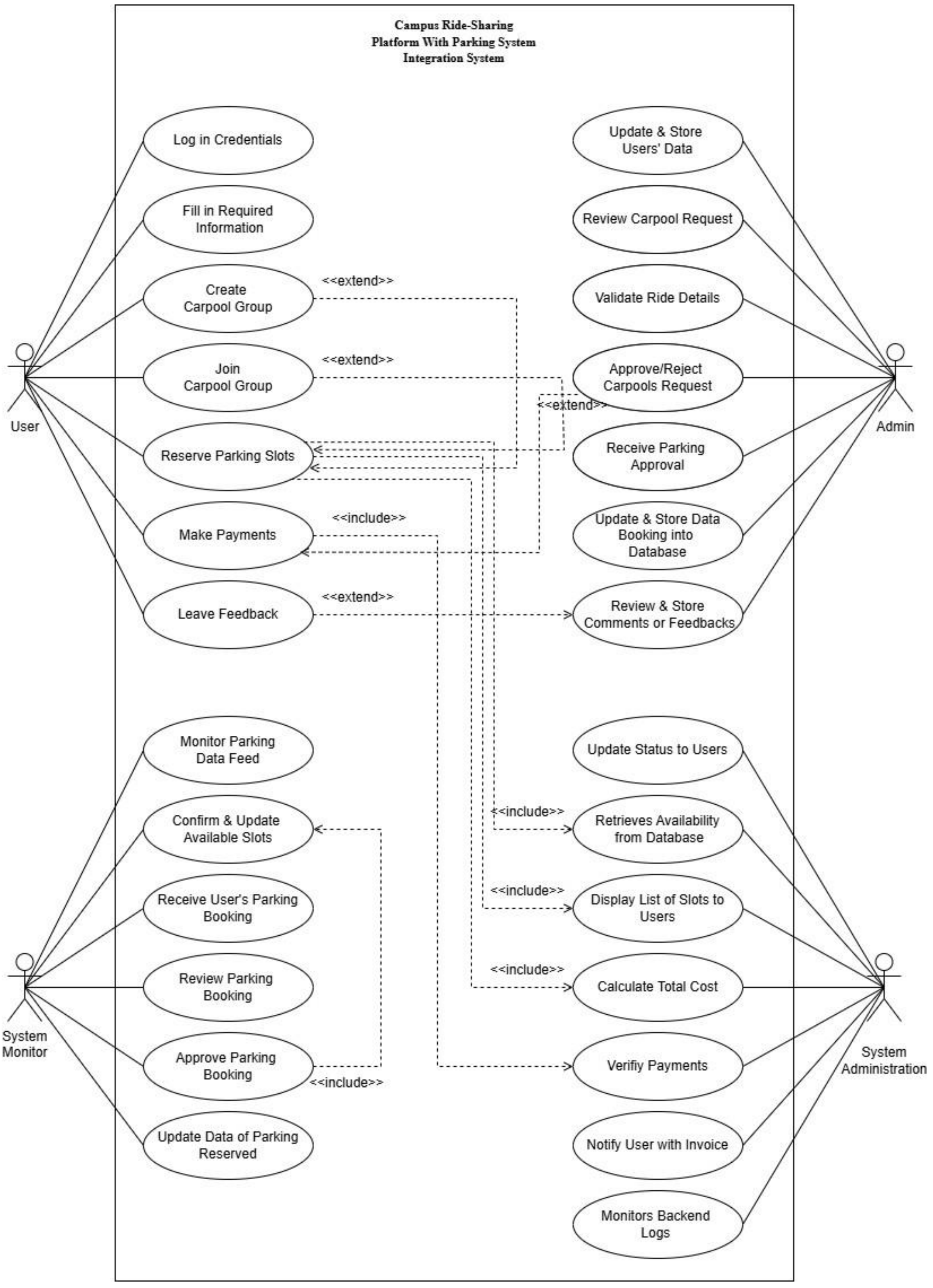


Figure 1.3.2: Use Case Diagram of Campus Ride Sharing Platform

1.3.2.1 User

1.3.2.1.1 Login Credentials

Use Case Name	Login with Credentials	**Version**	**1.0**
Description	Allows users to securely log in with registry email/username & password to use the features of the platform.		
Primary Actor	User		
Precondition	• Users are registered on the platform. • System is operational.		
Postcondition	• User is authenticated and directed to the dashboard/home page.		
Main Success Scenario	1. User clicks on the log in page. 2. User enters log in credentials. 3. User clicks on the Log In button. 4. System checks credentials. 5. User is logged in and gets directed to home page.		
Alternative Scenario	• Characters are incorrect: system indicates an error message. • User account is inactive; system indicates this to the user		

Table 1.3.2.1.1: Login Credentials

1.3.2.1.2 Fill in required informations

Use Case Name	Fill in required information	**Version**	**1.0**
Description	User completes or modifies their profile with personal and ride-related preferences		
Primary Actor	User		
Precondition	User has logged into the system		
Postcondition	User's information was saved into the system for personalization and operational purposes		
Main Success Scenario	1. Users navigate to their profile settings. 2. User completes their required fields(eg. name, contact information, car information, preferences) 3. User submits the form. 4. System validates and saves the information		
Alternative Scenario	• If the user has not filled out the fields: the system messages the user to fill out the fields. • If the data is in the wrong format: the system tells the user to correct the data.		

Table 1.3.2.1.2: Fill in required informations

1.3.2.1.3 Creating Carpool group

Use Case Name	Create Carpool Group	**Version**	**1.0**
Description	Allows a user to create a new carpool group and provide ride information, such as time, route, and capacity, with a maximum of 4 users per group..		
Primary Actor	User		
Precondition	• User is logged in. • User profile contains verified vehicle details.		
Postcondition	Carpool group request is submitted for administrator approval.		
Main Success Scenario	1. User navigates to the "Create Carpool Group". 2. User enters ride information,including the number of the carpool participants where the maksimum value is 4. 3. User submits the ride information form. 4. System validates data and ensures group size is within limits. 5. System submits request to admin for review		
Alternative Scenario	• If the ride information does not include the required data: the system prompts the user to complete. • If submission fails: the system shows an error message. • If the entered group size exceeds 4: system displays an error message stating 'Carpool group cannot exceed 4 users.'		

Table 1.3.2.1.3: Creating Carpool group

1.3.2.1.4 Joining Carpool group

Use Case Name	Joining Carpool Group	**Version**	**1.0**
Description	Allows a user to search the carpool groups available, and join a group based on the user's preferences and availability.		
Primary Actor	User		
Precondition	• User is logged in. • There is at least 1 active group in the system.		
Postcondition	Users are added to the carpool group selected.		
Main Success Scenario	1. Users browse/join a carpool group. 2. User selects a group and requests to join. 3. System checks to see if there is available seats. 4. System confirms the requests and adds user to the group.		
Alternative Scenario	• If there is no seats available: system provides notification. • If joining fails: error message is displayed.		

Table 1.3.2.1.4: Joining Carpool group

1.3.2.1.5 Reserve Parking Spot

Use Case Name	Reserve Parking Spot	**Version**	**1.0**
Description	Allows the user to reserve a parking spot for a scheduled ride.		
Primary Actor	User		
Precondition	User is part of an approved carpool.		
Postcondition	A parking spot has been reserved and persisted to the system.		
Main Success Scenario	1. User accesses the parking reservation module. 2. User selects the requested location and time. 3. System checks for availability. 4. User confirms reservation. 5. System updates the reservation confirmation, and notifies the user.		
Alternative Scenario	• If there are no available slots: Reserves parking module will display not available. • If failed booking: Reserves parking module will display error message.		

Table 1.3.2.1.5: Reserve Parking Spot

1.3.2.1.6 Make Payment

Use Case Name	Make Payment	**Version**	**1.0**
Description	Allows the user to pay for services (for example parking or carpooling fees) using supported payment methods.		
Primary Actor	User		
Precondition	• The user is logged in. • A payment is owed.		
Postcondition	The payment is processed and recorded.		
Main Success Scenario	1. The user navigates to the payment section. 2. The user selects the payment method. 3. The user inputs the required payment information. 4. The user confirms the payment. 5. The system processes the payment and generates a confirmation.		
Alternative Scenario	• If payment fails: the system logs the error and notifies the user. • If input is invalid: the user is prompted to re-enter.		

Table 1.3.2.1.6: Make Payment

1.3.2.1.7 Leave Feedback

Use Case Name	Leave Feedback	**Version**	**1.0**
Description	Users can leave feedback or rate their carpool experience.		
Primary Actor	User		
Precondition	The user has taken a carpool ride.		
Postcondition	Feedback is stored, and available for admins or relevant users.		
Main Success Scenario	1. User enters the section to leave feedback. 2. User selects the ride or experience to leave feedback on. 3. User fills in the feedback form or rating. 4. User submits feedback. 5. System saves feedback.		
Alternative Scenario	• If the form is not filled out: system requires it to be filled out. • If the submission fails: system will notify the user.		

Table 1.3.2.1.7: Leave Feedback

1.3.2.2 Administrator

1.3.2.2.1 Update & Store Users' Data

Use Case Name	Update & Store Users' Data	**Version**	**1.0**
Description	The process where the Administrator updates and stores user-related data such as profile information, carpool history, and ride preferences into the system database. This ensures the data remains accurate and up to date for efficient platform management.		
Primary Actor	Administrator		
Precondition	• Administrator is authenticated and logged into the system. • Access to the user data management module is granted.		
Postcondition	• Users' Data is successfully updated in the database. • Users' Data can be retrieved accurately by the system and relevant actors.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator navigates to the user data management module. 3. Administrator selects a user profile to update. 4. Administrator edits the required fields (name, email, contact, and car details). 5. Administrator submits the changes. 6. System updates it in the database. 7. A confirmation message is shown to the Administrator.		
Alternative Scenario	• If the selected user profile does not exist: The system will display an error message. • If the submitted data fails validation: The system will prompt the Administrator to correct the data and re-submit. • If the database update fails due to connection issue: The system logs the error and notifies the Administrator.		

Table 1.3.2.2.1: Update & Store Users' Data

1.3.2.2.2 Review Carpool Request

Use Case Name	Review Carpool Request	**Version**	**1.0**
Description	This use case allows the Administrator to review carpool group creation requests submitted by users. The Administrator evaluates the ride details such as driver information, time, location, and passenger capacity to ensure they meet platform requirements.		
Primary Actor	Administrator		
Precondition	• Administrator is logged into the system and has access to the carpool management module. • At least one carpool request has been submitted by a user.		
Postcondition	The carpool request is either approved and becomes active, or it is rejected with a reason recorded in the system.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator navigates to the carpool management module. 3. Administrator views a list of pending carpool requests. 4. Administrator selects a request and reviews the ride details. 5. Administrator verifies the accuracy and completeness of the information. 6. Administrator approves the request. 7. System updates the request status and notifies the requester.		
Alternative Scenario	• If the ride details are incomplete or invalid: Administrator rejects the request and provides a reason. Then, System Administration notifies the requester of the rejection and reason. • If the system fails to update the request status: An error message is shown and logged.		

Table 1.3.2.2.2: Review Carpool Request

1.3.2.2.3 Validate Ride Details

Use Case Name	Validate Ride Details	**Version**	**1.0**
Description	Covers the Administrator’s task of validating the ride details submitted by users when creating or updating a carpool. The Administrator ensures that all information such as driver credentials, pickup/drop-off points, schedule, and capacity comply with platform policies.		
Primary Actor	Administrator		
Precondition	Ride details have been submitted by a user and are pending validation.		
Postcondition	Ride details are either validated and ready for approval, or flagged for correction.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator navigates to the list of pending ride submissions. 3. Administrator selects a ride to validate. 4. Administrator verifies that all required details are complete and comply with platform rules (valid driver license and appropriate schedule). 5. Administrator marks the ride as "validated". 6. System updates the status and notifies relevant parties.		
Alternative Scenario	• If any ride detail is missing or invalid: Administrator marks the ride as “invalid” and provides a note for correction. Then, System Administration will notify the user to revise and resubmit the ride details.		

Table 1.3.2.2.3: Validate Ride Details

1.3.2.2.4 Approve/Reject Carpools Request

Use Case Name	Approve/Reject Carpools Request	**Version**	**1.0**
Description	This use case shows how the Administrator approves or rejects carpool requests submitted by users after reviewing and validating the ride details. Approval activates the carpool for users to join, while rejection prevents it from being listed.		
Primary Actor	Administrator		
Precondition	• Administrator is logged into the system. • Ride details have already been validated		
Postcondition	Carpool requests are either approved and published, or rejected with a reason stored in the system.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator navigates to the validated carpool requests section. 3. Administrator selects a carpool request. 4. Administrator reviews the final information. 5. Administrator selects “Approve” or “Reject”. 6. If approved, the system updates the carpool status to active and notifies the requester. 7. If rejected, the system stores the rejection reason and notifies the requester.		
Alternative Scenario	• If additional information is required: Administrator puts the request on hold and contacts the requester for clarification. • If the system fails to update the status: System logs the error and displays a message. Then, Administrator retries or reports to System Administration.		

Table 1.3.2.2.4: Approve/Reject Carpool Request

1.3.2.2.5 Receive Parking Approval

Use Case Name	Receive Parking Approval	**Version**	**1.0**
Description	This use case describes how the Administrator receives parking booking approval status from the System Monitor. Once approval is received, the Administrator records and updates the related booking data in the system for tracking and reporting.		
Primary Actor	Administrator		
Precondition	• The System Monitor has reviewed and approved a parking booking request. • The Administrator is logged in and has access to the booking management module.		
Postcondition	Parking booking status is updated in the system, and the information is stored in the database.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator navigates to the parking approvals section. 3. Administrator views the list of approved parking bookings sent by the System Monitor. 4. Administrator verifies the booking details. 5. Administrator confirms receipt and updates the booking status in the database. 6. System logs the update and sends confirmation to the user.		
Alternative Scenario	• If no approval is received yet: Administrator waits or sends a request to System Monitor for update. • If the system fails to update the parking reserve status: Error is logged and Administrator is notified. Then, Administrator retries or escalates to System Administration.		

Table 1.3.2.2.5: Receive Parking Approval

1.3.2.2.6 Update & Store Booking Data into Database

Use Case Name	Update & Store Booking Data into Database	**Version**	**1.0**
Description	This use case outlines how the Administrator inputs and stores finalized booking data into the database after both carpool and parking reservations have been approved. This ensures complete records are maintained for reporting, tracking, and confirmation purposes.		
Primary Actor	Administrator		
Precondition	Carpool request has been approved by Administrator, and parking reservation has been approved by System Monitor.		
Postcondition	Booking data is stored in the database.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator navigates to the booking management module. 3. Administrator verifies that both carpool and parking approvals are completed. 4. Administrator enters booking details (user info, ride info, parking slot, timing, cost, etc). 5. Administrator submits the booking data. 6. System stores the data in the database and System Administration sends a confirmation to the user.		
Alternative Scenario	• If approvals are not yet completed: Administrator cannot proceed and the system displays a notification. • If data submission fails: System displays an error message and logs the issue. Then, Administrator retries or contacts System Administration for assistance.		

Table 1.3.2.2.6: Update & Store Data Booking into Database

1.3.2.2.7 Review & Store Comments or Feedbacks

Use Case Name	Review & Store Comments or Feedbacks	**Version**	**1.0**
Description	This use case describes how the Administrator reviews comments or feedback submitted by users regarding their ride-sharing or parking experience and stores relevant feedback into the database for future reference, analysis, or improvement.		
Primary Actor	Administrator		
Precondition	Users have submitted comments or feedback through the system.		
Postcondition	Feedback is reviewed and either stored in the database or flagged for further action.		
Main Success Scenario	1. Administrator logs into the system. 2. Administrator accesses the feedback management module. 3. Administrator views new/unprocessed comments or feedback. 4. Administrator reviews each submission for relevance and quality. 5. Administrator stores valid and constructive feedback in the database. 6. Administrator flags any feedback that requires follow-up or moderation. 7. System updates the status of each feedback to “Stored” or “Flagged”.		
Alternative Scenario	• If the feedback contains inappropriate or abusive content: Administrator flags the content and escalates it to the System Administration or moderation team.		

Table 1.3.2.2.7: Review & Store Comments or Feedbacks

1.3.2.3 System Administration

1.3.2.3.1 Update Status to Users

Use Case Name	Update Status to Users	**Version**	**1.0**
Description	Updates booking or system-related status to users after verifying such as payment or carpool approval		
Primary Actor	System administration		
Precondition	Booking or system status exists and requires an update		
Postcondition	User notified of the updated status		
Main Success Scenario	1. System administration access user management panel 2. System receives update of database from administrator and system monitor. 3. System administration review “pending notification sending” 4. Selects the relevant booking or user 5. Updates status 6. System send notification to user		
Alternative Scenario	• System encounters database error • Status is not updated		

Table 1.3.2.3.1: Update Status to Users

1.3.2.3.2 Retrieves Availability from Database

Use Case Name	Retrieves Availability from database	**Version**	**1.0**
Description	Retrieves real-time parking slot data from the database to ensure accurate status display		
Primary Actor	System administration		
Precondition	• Log into the system • Database is functional		
Postcondition	Parking availability is retrieve and ready to be displayed to users		
Main Success Scenario	1. Administrator access parking availability module 2. System requires database 3. Current availability is retrieved 4. Data is display to the end users		
Alternative Scenario	• Database timeout or failure occur • Systems returns an error and log incident		

Table 1.3.2.3.2: Retrieves Availability from Database

1.3.2.3.3 Display Lists of Slots to Users

Use Case Name	Display List of Slots to User	**Version**	**1.0**
Description	Ensures the platform display an up-to-date data list of available parking slot		
Primary Actor	System administration		
Precondition	User request parking slot information		
Postcondition	Users see updated list of available slots for reservation		
Main Success Scenario	1. Administrator verifies slot data retrieval 2. System displays list of available parking slot to users 3. Users interact with the list to book		
Alternative Scenario	• Display fails • User see outdated list		

Table 1.3.2.3.3: Display List of Slot to User

1.3.2.3.4 Calculate Total Cost

Use Case Name	Calculate Total Cost	**Version**	**1.0**
Description	Reviews and verifies correct cost calculation for reservations based on duration, slot type, and carpooling status,including both ride and parking fees		
Primary Actor	System administration		
Precondition	• User submits booking details • Pricing scheme is available in the system		
Postcondition	Total ride cost calculated and presented to user for payment		
Main Success Scenario	1. Administrator ensures rate rules are configured in the system. 2. System calculates ride fee based on ride details 3. System calculates parking fee based on parking details 4. System calculates total cost by summing ride and parking fees 5. Final cost is shown to the user.		
Alternative Scenario	• Pricing rules are outdated or not found. • System defaults to standard rate or shows an error. • Administrator corrects rate and recalculates.		

Table 1.3.2.3.4: Calculate Total Cost

1.3.2.3.5 Verify Payments

Use Case Name	Verify Payments	**Version**	**1.0**
Description	Verifies user payment information by cross-checking with the payment gateway to confirm successful transactions		
Primary Actor	System administration		
Precondition	• A payment has been made by the user • System Administrator is authenticated		
Postcondition	• Payment is verified and logged • Status is marked as “Paid”		
Main Success Scenario	1. System administrator accesses payment verification 2. System cross-checks transaction with gateway 3. Valid payment is confirmed 4. Booking marked as paid		
Alternative Scenario	• Payment mismatch or failure. • System administrator contacts the user for resubmission. • System logs failed verification.		

Table 1.3.2.3.5: Verify Payments

1.3.2.3.6 Notify User with Invoice

Use Case Name	Notify User with Invoice	**Version**	**1.0**
Description	Sends a confirmation message to users after successful actions like booking, payment, or approval		
Primary Actor	System administration		
Precondition	A successful book is completed		
Postcondition	User receives confirmation notification via platform or email/SMS		
Main Success Scenario	1. Receive update of booking approval from administrator 2. System Administration validate the data 3. System triggers a confirmation message 4. User receives and views the message		
Alternative Scenario	• Notification service fails. • System retries or stores messages in logs.		

Table 1.3.2.3.6: Notify User with Invoice

1.3.2.3.7 Monitor Backend Logs

Use Case Name	Monitor Backend Logs	**Version**	**1.0**
Description	Monitor system logs for performance, error tracking, and audit purposes		
Primary Actor	System administration		
Precondition	• System administrator is logged in with required permissions • System generates logs regularly		
Postcondition	• System administrator reviews logs for errors, performance metrics, or suspicious activity • Actions taken are recorded		
Main Success Scenario	1. System administrator opens a log monitoring module 2. Views logs by filter (date, action type) 3. Identifies any issue or verifies normal operation 4. Take corrective action if necessary		
Alternative Scenario	• Log module fails to load • Temporary logs accessed manually		

Table 1.3.2.3.7: Monitor Backend Logs

1.3.2.4 System Monitor

1.3.2.4.1 Monitor Parking Data Feed

Use Case Name	Monitor Parking Data Feed	**Version**	**1.0**
Description	This use case describes how the system monitor is constantly monitoring the real-time parking data feed to refresh the system with up-to-date availability of parking slots.		
Primary Actor	System Monitor		
Precondition	The parking system needs to be online and interfaced with parking slot sensors or manual input sources.		
Postcondition	The system is updated with the latest available parking slot details and shows accurate availability for access by users.		
Main Success Scenario	1. The system monitor initiates the parking data feed scan. 2. The system receives real-time data from sensors or input sources. 3. The system processes the data and calculates available slots. 4. The updated parking availability is stored in the database. 5. The updated data is reflected in the user interface for reservation.		
Alternative Scenario	If the data feed is temporarily unavailable: • The system logs the issue and retries after a set interval. • If retries fail, an alert is sent to the system administrator for manual intervention.		

Table 1.3.2.4.1 Monitor Parking Data Feed

1.3.2.4.2 Confirm & Update Available Slots

Use Case Name	Confirm & Update Available Slots	**Version**	**1.0**
Description	This use case describes how the system monitor confirms the parking slot count and updates the system to reflect the latest availability based on real-time data or user actions.		
Primary Actor	System Monitor		
Precondition	Real-time parking data feed is active and the database is accessible for updates.		
Postcondition	The system reflects the most accurate number of available parking slots for reservation.		
Main Success Scenario	1. System monitor receives new data on parking occupancy. 2. Confirms accuracy of data (checks against current records). 3. Calculates the number of available parking slots. 4. Updates the database with the new count. 5. Availability is refreshed on the user interface.		
Alternative Scenario	Data inconsistency is detected: • The system flags the entry for review. • Logs the issue and notifies the system administrator. • Falls back to last known good data until resolved.		

Table 1.3.2.4.2 Confirm & Update Available Slots

1.3.2.4.3 Receive User's Parking Booking

Use Case Name	Receive User's Parking Booking	**Version**	**1.0**
Description	This use case outlines how the system monitor receives a parking reservation request from a user after carpool approval and payment verification.		
Primary Actor	System Monitor		
Precondition	• User is authenticated. • Carpool group is approved. • Payment (if required) has been verified.		
Postcondition	The parking reservation request is received and queued for review or automatic processing.		
Main Success Scenario	1. User submits a parking reservation request through the system. 2. System monitor receives the request. 3. Logs the booking details (user ID, time, zone). 4. Triggers the next use case: Review Parking Booking. 5. Notifies the administrator or responsible process for further approval.		
Alternative Scenario	User submits a request with missing or invalid data: • System rejects the request. • User is notified to resubmit with correct information.		

Table 1.3.2.4.3 Receive User's Parking Booking

1.3.2.4.4 Review Parking Booking

Use Case Name	Review Parking Booking	**Version**	**1.0**
Description	This use case describes the process in which the System Monitor examines the details of a user's parking reservation request before it is approved or rejected.		
Primary Actor	System Monitor		
Precondition	• A valid parking reservation request has been received. • Parking data feed and available slots are up to date.		
Postcondition	The booking request is either approved or marked for rejection based on availability and policy rules.		
Main Success Scenario	1. System Monitor retrieves the reservation request. 2. Compares the request details against the real-time parking availability. 3. Validates reservation timing, location, and user eligibility. 4. Marks the request as ready for approval. 5. Notifies the next process (Approve Parking Booking).		
Alternative Scenario	Slot is already taken or availability data is outdated: • System flags the request as failed. • Users are notified to choose another slot or try again later.		

Table 1.3.2.4.4 Review Parking Booking

1.3.2.4.5 Approve Parking Booking

Use Case Name	Approve Parking Booking	**Version**	**1.0**
Description	This use case outlines how the System Monitor confirms and finalizes a user's parking reservation by approving it and triggering related updates to the system data.		
Primary Actor	System Monitor		
Precondition	• The parking booking request has been reviewed and validated. • A parking slot is available.		
Postcondition	• The parking slot is reserved for the user. • The system updates availability and notifies relevant parties.		
Main Success Scenario	1. System Monitor selects a reviewed parking booking. 2. Confirms all details are valid and slots are available. 3. Approves the request. 4. Updates the system to mark the slot as reserved. 5. Sends confirmation to the user and related components (Administrator, System Administration).		
Alternative Scenario	Slot becomes unavailable during approval process: • System aborts approval. • Users are notified to make a new reservation.		

Table 1.3.2.4.5 Approve Parking Booking

1.3.2.4.6 Update Data of Parking Reserved

Use Case Name	Update Data of Parking Reserved	**Version**	**1.0**
Description	This use case describes how the System Monitor updates the internal system database to reflect a successful parking reservation, ensuring that availability data remains accurate.		
Primary Actor	System Monitor		
Precondition	• A parking booking has been approved by the System Monitor.		
Postcondition	• The reserved parking slot is marked as unavailable in the system. • The booking details are recorded in the system.		
Main Success Scenario	1. System Monitor receives a confirmed approval for a parking reservation. 2. Accesses the current parking data from the system. 3. Updates the specific parking slot's status to "Reserved." 4. Logs the reservation details (user, time, location) into the database. 5. System syncs updated data across all relevant modules (Administrator, System Administration).		
Alternative Scenario	System error while updating the database: • The system logs the error and notifies the System Administrator. • The user is informed that their reservation could not be completed and must retry.		

Table 1.3.2.4.6 Update Data of Parking Reserved

1.3.3 User Characteristic

The Campus Ride-Sharing and Parking Reservation System target user groups may differ based on the deployment context. However, some of the common user characteristics that may influence usability problems are as follows:

1. User (Students, Faculty, Staff):
   - I. Users shall log in using their university credentials.
   - II. Users shall fill in required personal and ride-related information.
   - III. Users shall create a new carpool group when initiating a ride.
   - IV. Users shall join an existing carpool group upon invitation or acceptance.
   - V. Users shall reserve a parking spot once their carpool is approved.
   - VI. Users shall make parking payments through the system.
   - VII. Users shall leave feedback or ratings based on their experience.

2. Administrator:
   - I. Administrator shall update and store users' personal and carpool data.
   - II. Administrator shall review all new carpool creation and join requests.
   - III. Administrator shall validate carpool ride details such as participants and schedules.
   - IV. Administrator shall approve or reject carpool group requests.
   - V. Administrator shall receive parking approval notifications from the system monitor.
   - VI. Administrator shall update and store final booking data in the system database.
   - VII. Administrator shall review user feedback and comments for quality assurance.

3. System Administration:
   - I. System Administration shall update booking or status notifications to users.
   - II. System Administration shall retrieve parking availability from the system database.
   - III. System Administration shall display a list of available parking slots to users.
   - IV. System Administration shall calculate the total cost including ride and parking.
   - V. System Administration shall verify payment transactions.
   - VI. System Administration shall send booking confirmation notifications via system and email.
   - VII. System Administrator shall monitor backend logs and handle technical errors.

4. System Monitor:
   - I. System Monitor shall monitor real-time parking data feeds.
   - II. System Monitor shall update available parking slot information.

III. System Monitor shall confirm the current parking count.

IV. System Monitor shall receive parking reservation requests from users.

V. System Monitor shall auto-approve valid parking reservations.

VI. System Monitor shall update reserved parking data in the database.

1.3.4 Limitations

- Dependency on Real-Time Data:

  The system relies heavily on real-time parking availability data. Any delay or inconsistency in this data may affect booking accuracy.

- Limited Feedback Moderation:

  Comments flagged as inappropriate are only reviewed manually by the System Monitor. There is no automated filtering or AI moderation for user-submitted content.

- Platform scope:

  The system is only accessible to users with valid university IDs. it does not support external user or guest login.

- Dependant on Internet Connectivity:

  Users must have internet access to use the platform. Offline functionality does not support the system.

- Manual Approval:

  Carpool group approval may require manual invention by the admins, possibly introducing delay

1.4 Definitions

Table 1.4 shows the core definitions that are used throughout the Campus Ride-Sharing Platform with Parking System Integration. These definitions provide clarity on the actors and components of the system as well as terms used for user interaction, systems administration, parking management and back-end. To facilitate a cohesive understanding of the terms used herein will help ensure that the reader applies the same interpretation to the overall requirements and functionality described in this document.

Term	**Definition**
User	A student, staff, or faculty member of MMU who utilizes the platform to form or join a carpool or reserve a parking spot.
Administrator	The user role that is responsible for adding user information, checking and approving carpool requests, and modifying reservations in the system.
System Monitor	Assesses real-time availability of parking spots, collects and maintains parking data, and approves parking reservations.
System Administrator	Supervises back-end action to update reservation status, approve and deny payments, and send confirmations.
Carpool Group	A group created by users to carpool users together to/from campus; group needs to be confirmed before use.
Parking Reservation	A confirmed reservation for committed parking on campus, only available to confirmed carpool members. Feedback Comments or rating from users based on their carpool and reservation experience.
Feedback	Comments or rating from users based on their carpool and reservation experience.
Authentication System	The university login system to verify users with MMU credentials.
Dashboard	The administrative web interface that allows the Administrator user to manage requests, monitor activities within the system, and review feedback.
Real-time Parking Feed	A live feed that represents the location and availability of parking spots within the campus's many areas.
Payment Gateway	A third-party system integrated into the platform to process user payments for services.
Notification System	A component that sends system alerts, updates, and confirmations to users via email or the platform.

***Table 1.4**: Key Terms and Definitions for the Campus Ride-Sharing Platform*

2 References

This Document is prepared in reference to the following documents:

1. KDK College of Engineering. *SE Unit 2: Software Engineering – Requirements Engineering*. https://kdkce.edu.in/writereaddata/fckimagefile/SE%20Unit%202.pdf

2. D. Sachan, “Software Engineering | Requirement Engineering,” *Scaler Topics*, Sep. 20, 2023. https://www.scaler.com/topics/requirements-engineering-in-software-engineering/

3. *IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830-1998).* Institute of Electrical and Electronics Engineers, 1998. https://anyflip.com/pncyn/vebt/basic

4. GeeksforGeeks, "Software Engineering – Requirements Engineering Process." https://www.geeksforgeeks.org/software-engineering-requirements-engineering-process/

3 Requirements

3.1 Functions

3.1.1 User

3.1.1.1 Login Credentials

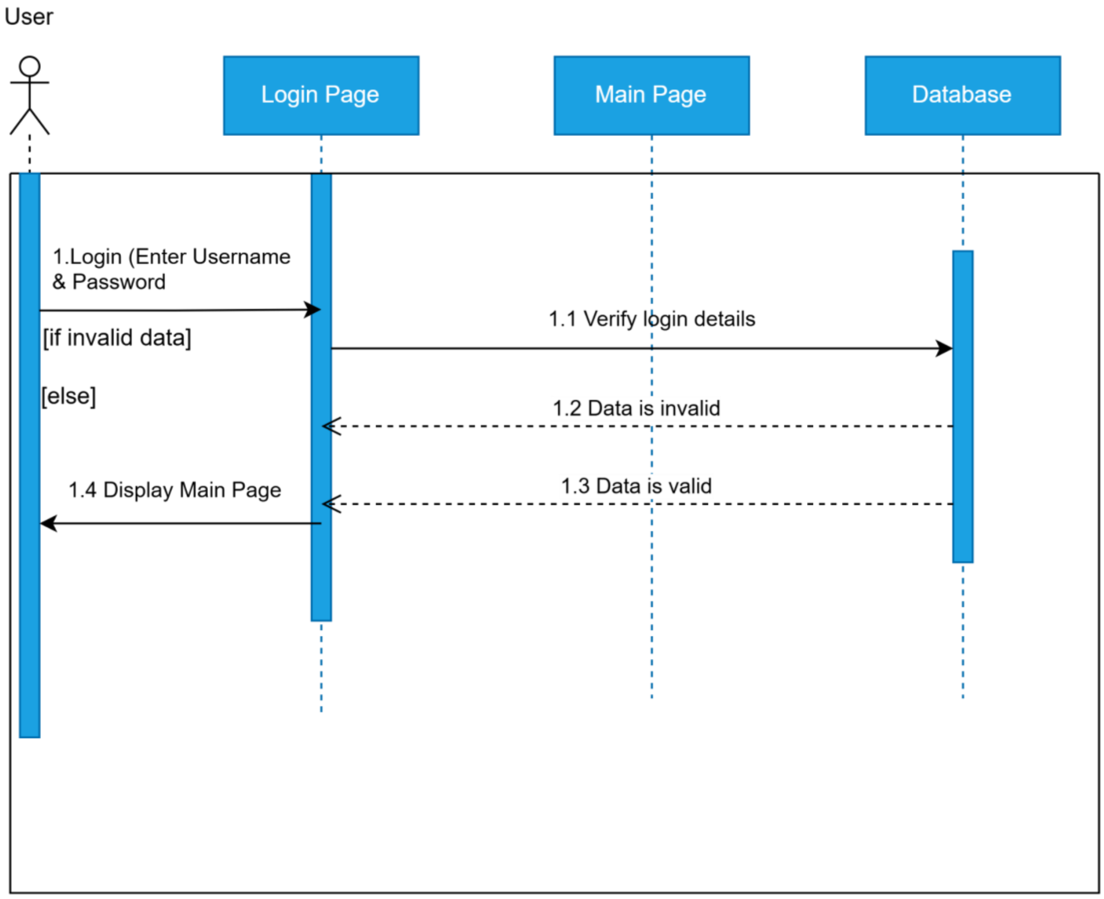


Figure 3.1.1.1: Sequence Diagram (Login Credentials)

3.1.1.2 Fill in Required Information

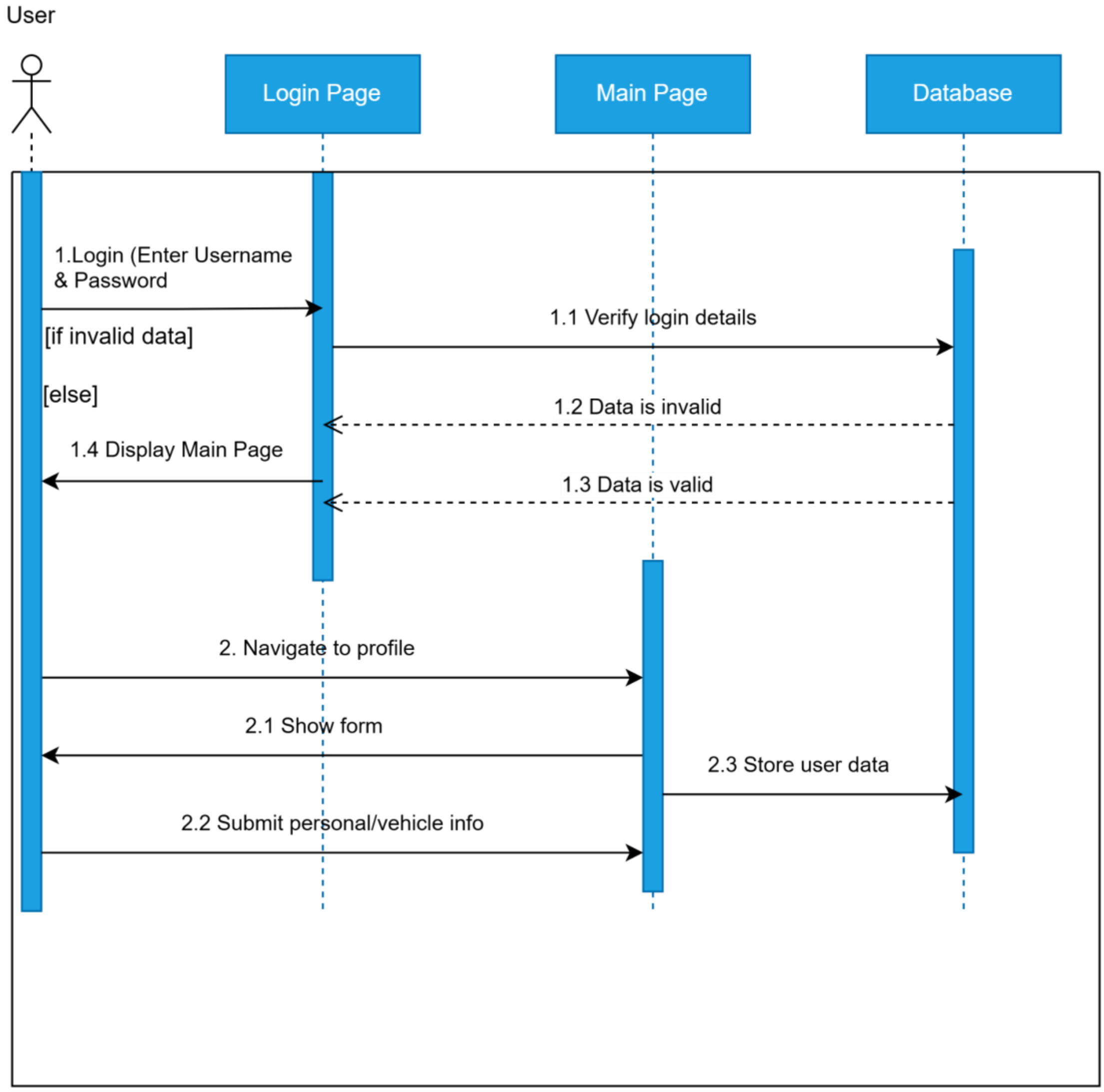


Figure 3.1.1.2: Sequence Diagram (Fill in Required Information)

3.1.1.3 Create Carpool Group

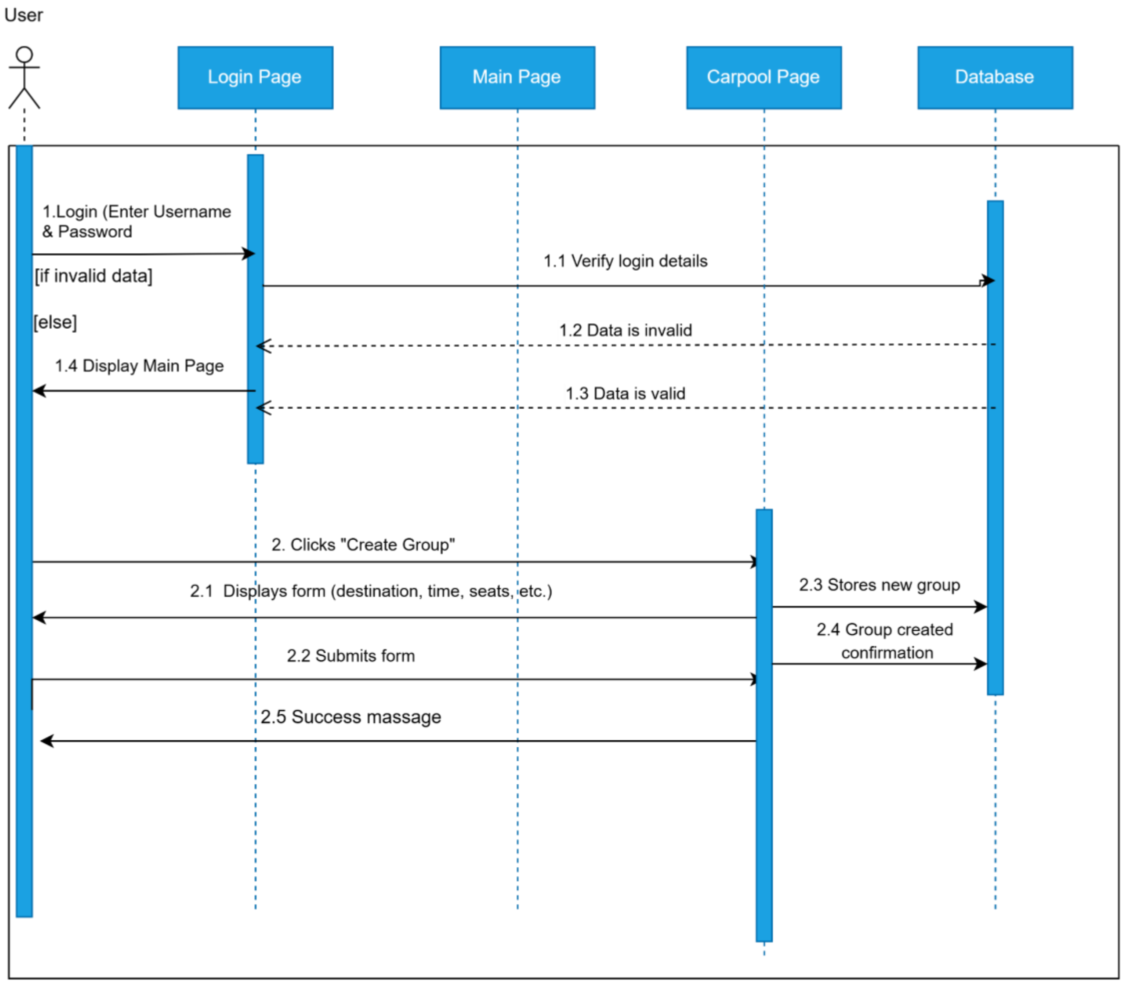


Figure 3.1.1.3: Sequence Diagram (Create Carpool Group)

3.1.1.4 Join Carpool Group

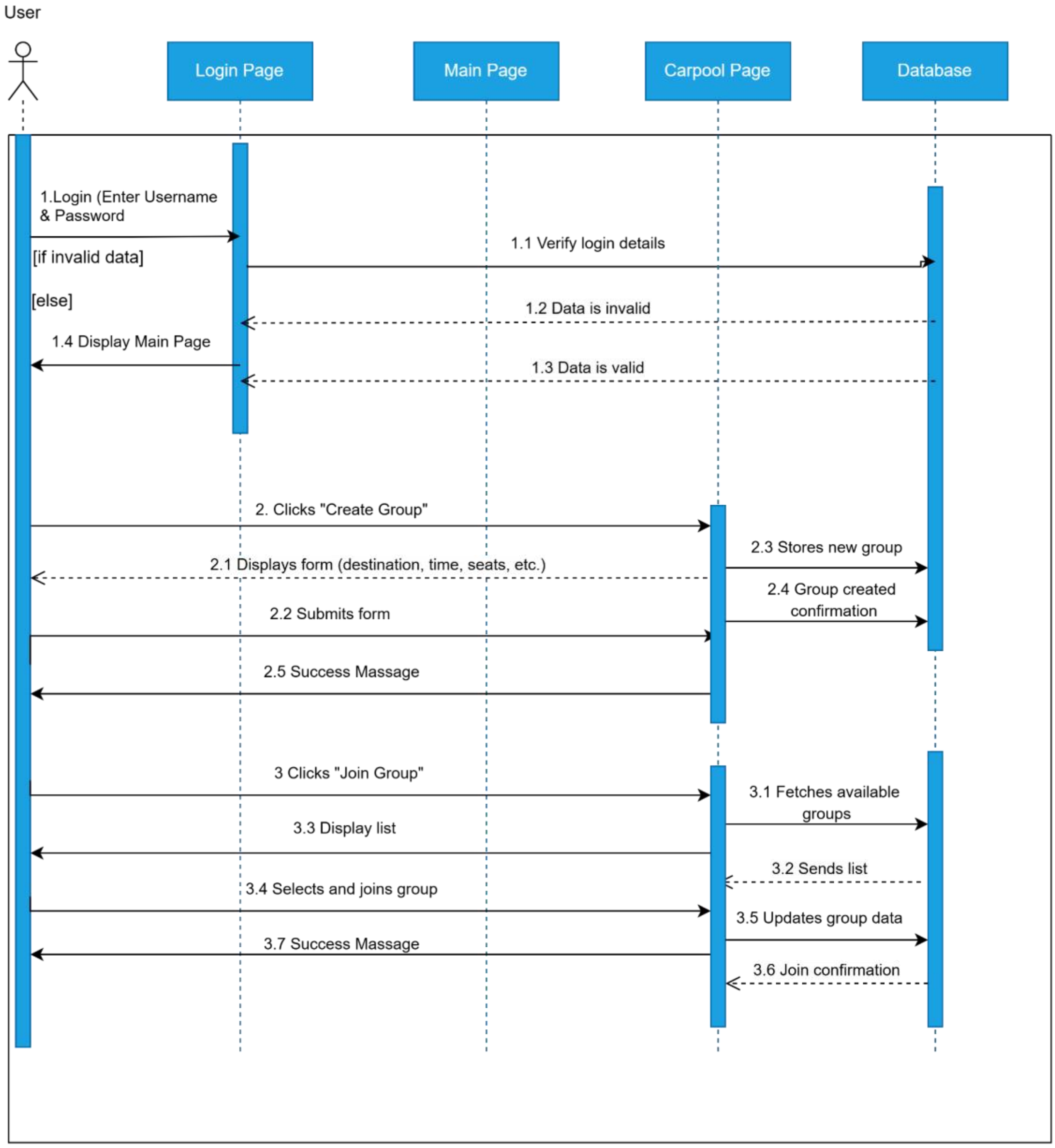


Figure 3.1.1.4: Sequence Diagram (Join Carpool Group)

3.1.1.5 Reserve Parking Spot

3.1.1.5 Reserve Parking SpotAdd commentMore actions

Users must be part of an approved carpool group before they are able to reserve a parking spot on campus. This requirement aims to encourage sustainable commuting and reduce the number of single-occupancy vehicles on campus. Carpooling is an integral part of the platform, as it optimizes the usage of parking spots and contributes to reducing campus traffic congestion.

Flexibility: While carpooling is generally required for parking reservations, the system allows exceptions for special cases. Users may reserve parking without carpooling for **emergency parking**, **short-term parking**, or **VIP users**. These exceptions will be determined based on system rules or administrator approval.

Justification for Flexibility:

The platform aims to be flexible and accommodate various needs, including users who may need parking for urgent or unforeseen circumstances. While carpooling is the default requirement, allowing some flexibility ensures that the system can still support diverse user needs.

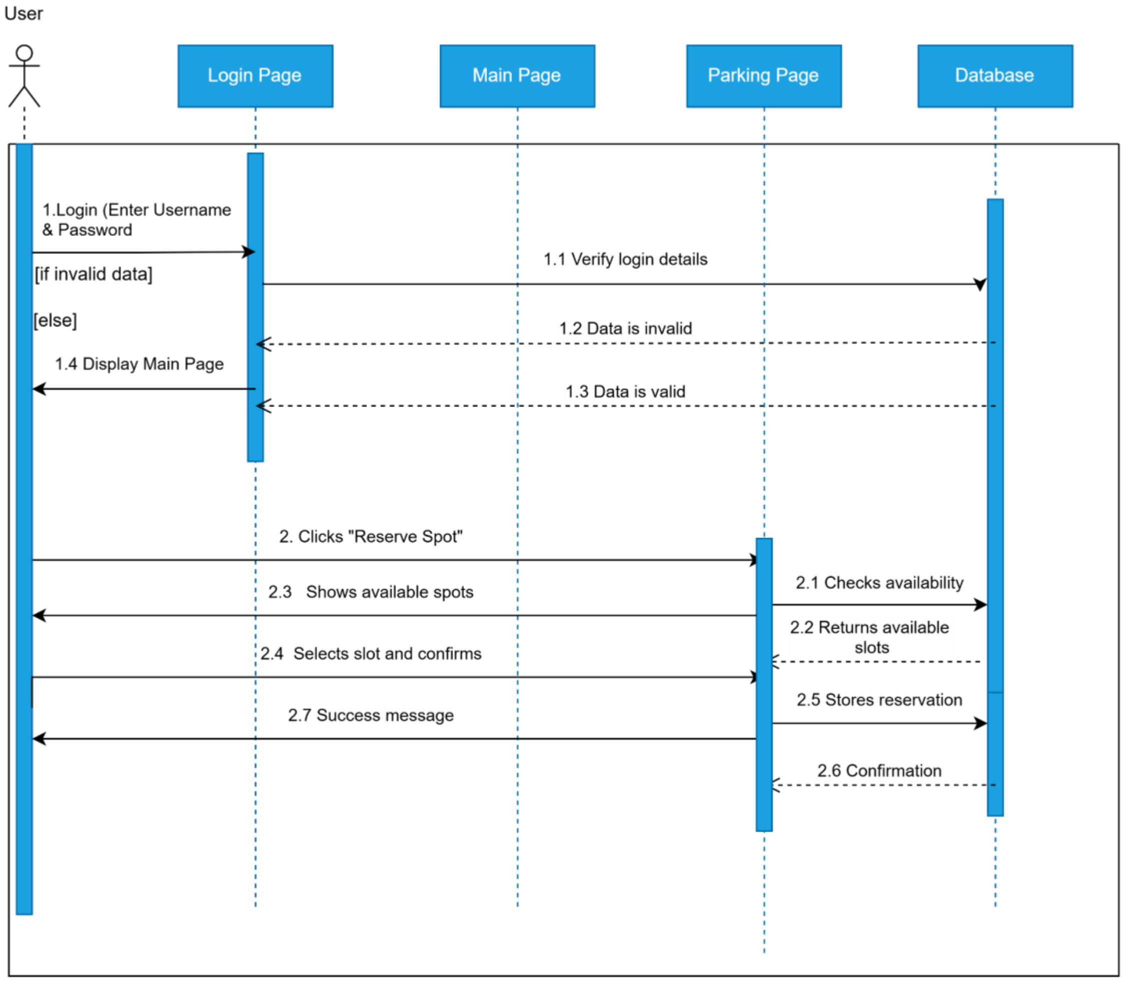


Figure 3.1.1.5: Sequence Diagram (Reserve Parking Spot)

3.1.1.6 Make Payment

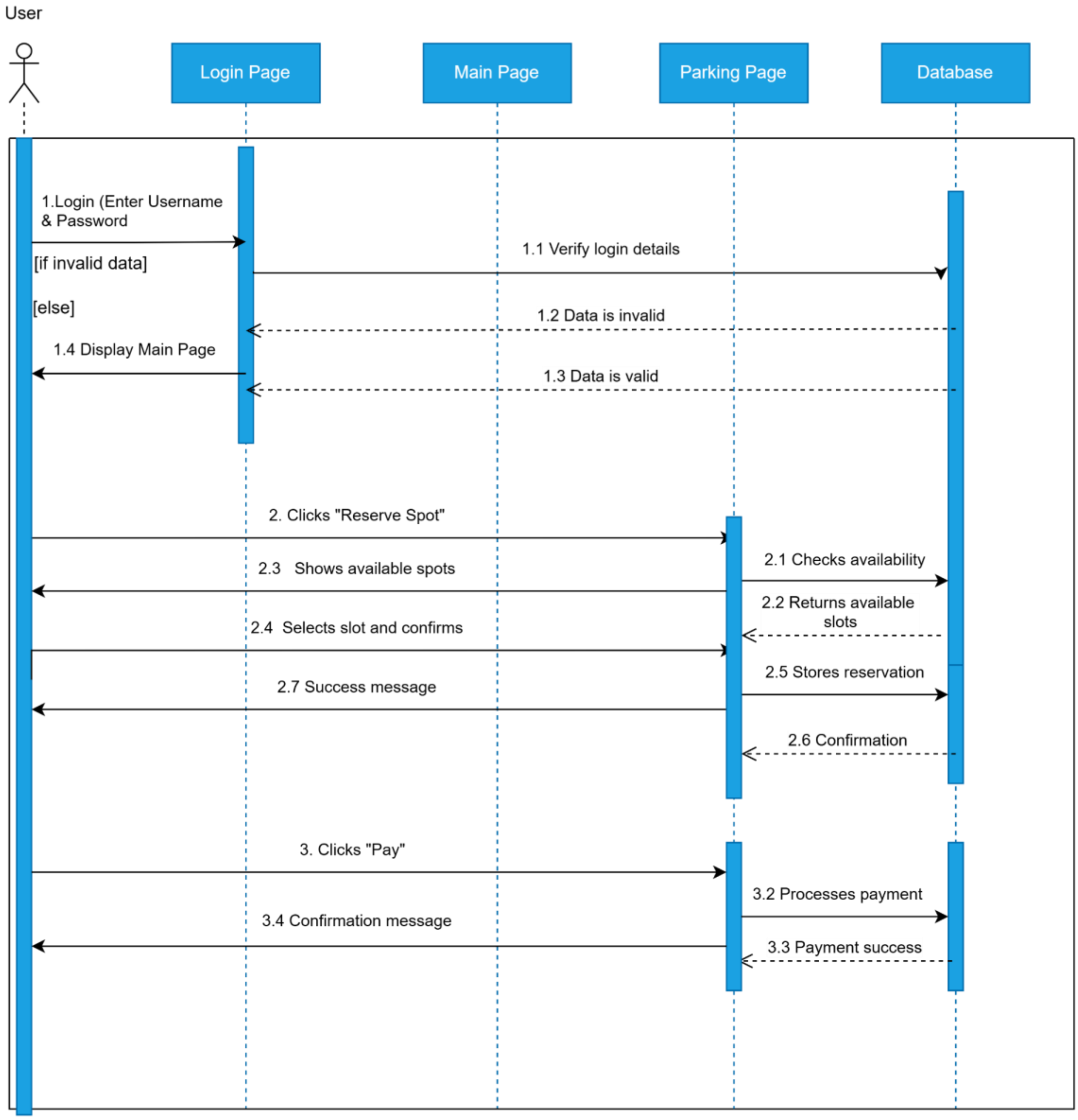


Figure 3.1.1.6: Sequence Diagram (Make Payment)

3.1.1.2 Leave Feedback

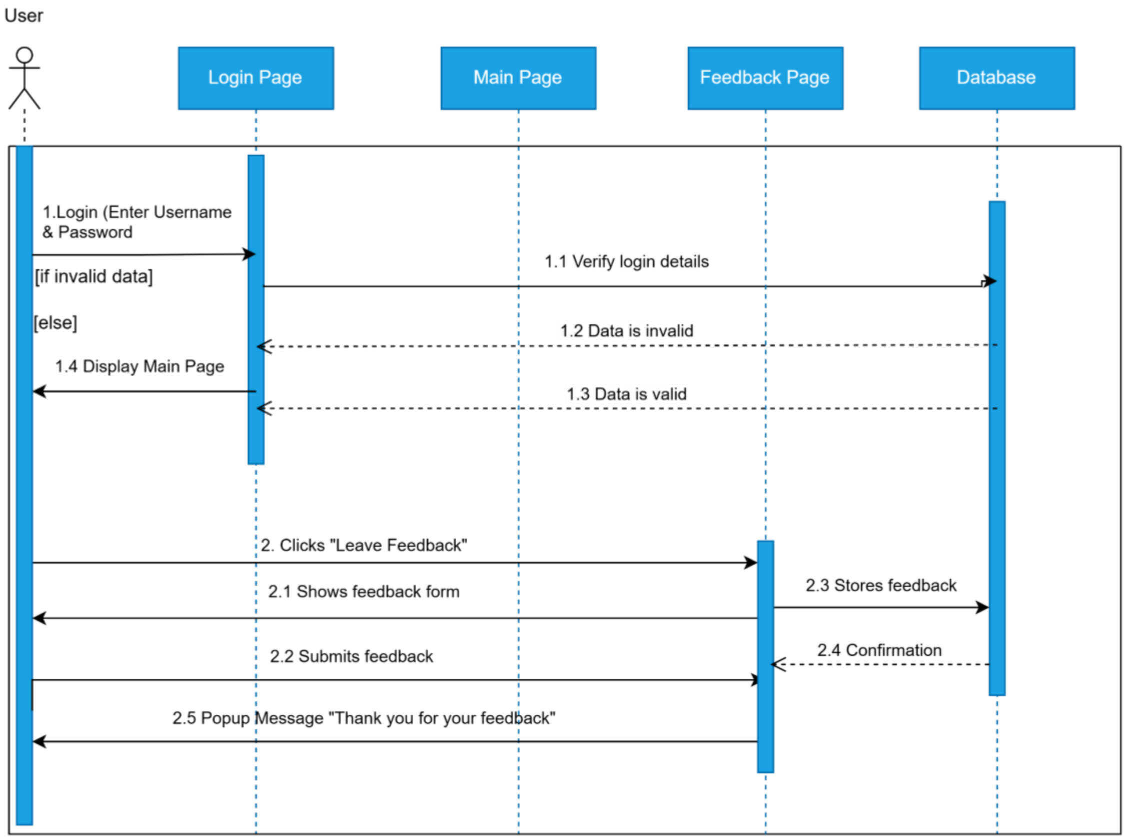


Figure 3.1.1.2: Sequence Diagram (Leave Feedback)

3.1.2 Administrator

3.1.2.1 Update & Store Users' Data

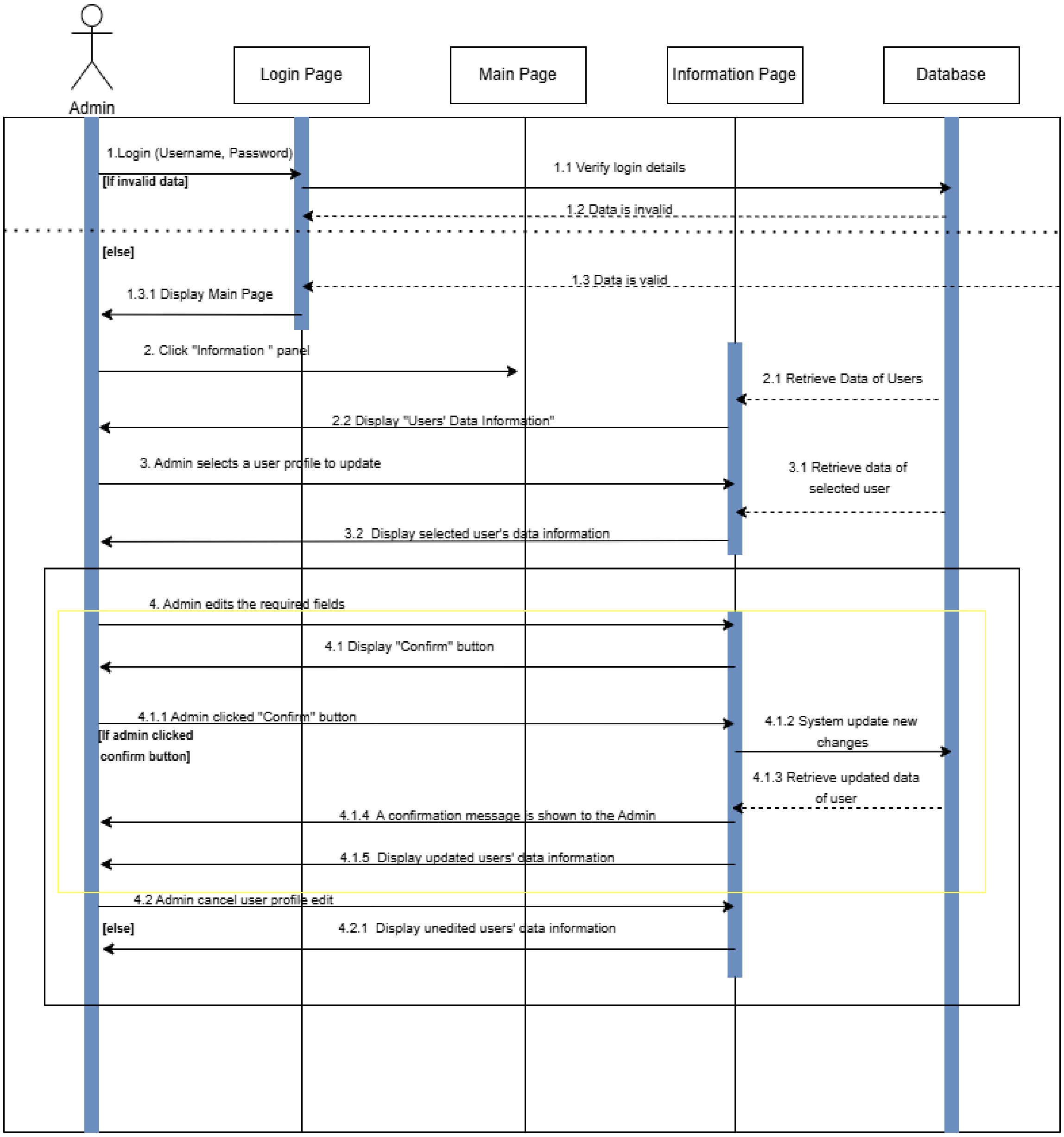

Admin
Login Page
Main Page
Information Page
Database
1.Login (Username, Password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display Main Page
2. Click "Information " panel
2.1 Retrieve Data of Users
2.2 Display "Users' Data Information"
3. Admin selects a user profile to update
3.1 Retrieve data of selected user
3.2 Display selected user's data information
4. Admin edits the required fields
4.1 Display "Confirm" button
4.1.1 Admin clicked "Confirm" button
[If admin clicked confirm button]
4.1.2 System update new changes
4.1.3 Retrieve updated data of user
4.1.4 A confirmation message is shown to the Admin
4.1.5 Display updated users' data information
4.2 Admin cancel user profile edit
[else]
4.2.1 Display unedited users' data information

Figure 3.1.2.1: Update & Store Users' Data Sequence Diagram

3.1.2.2 Review Carpool Request

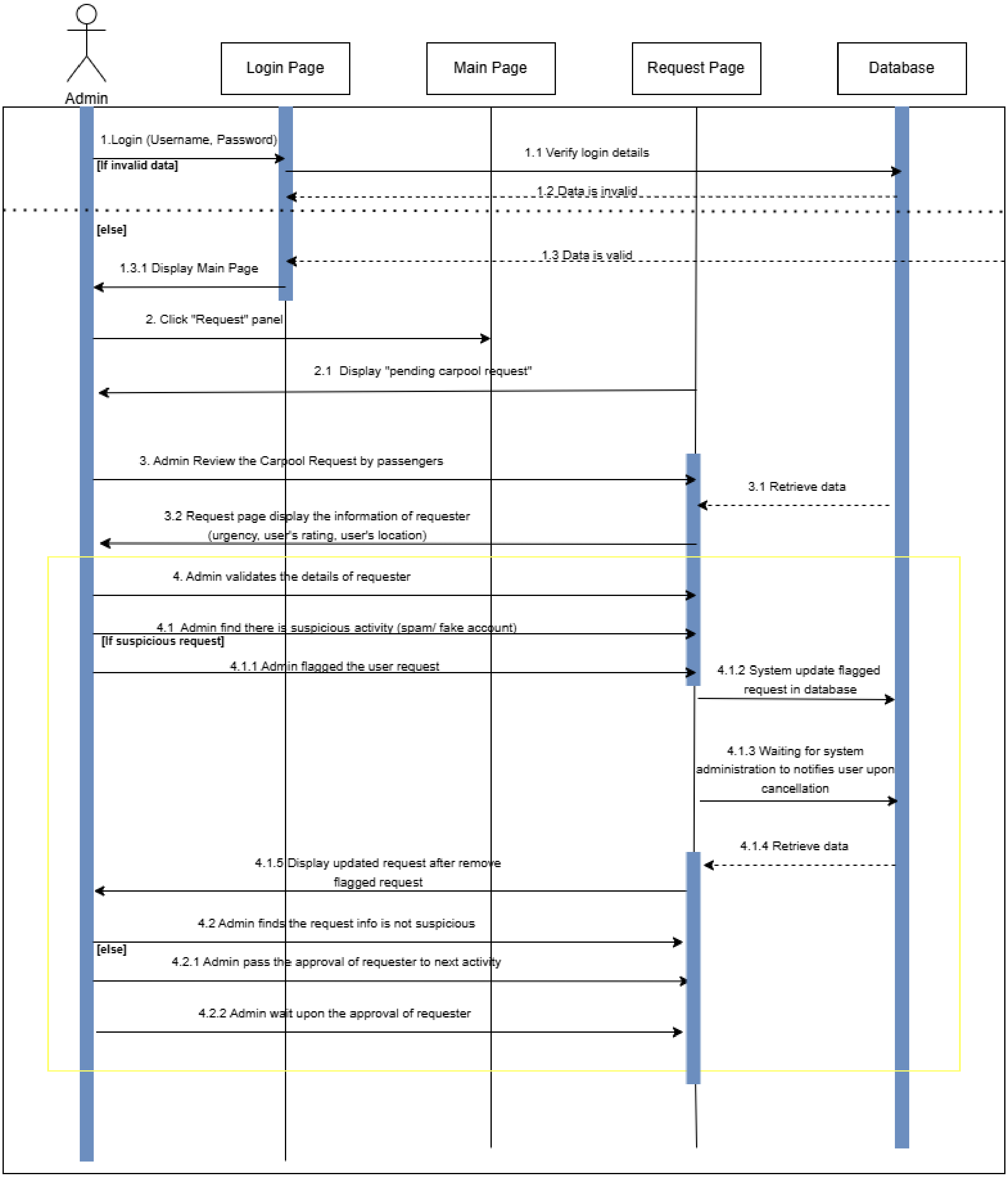

Admin
Login Page
Main Page
Request Page
Database
1.Login (Username, Password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display Main Page
2. Click "Request" panel
2.1 Display "pending carpool request"
3. Admin Review the Carpool Request by passengers
3.1 Retrieve data
3.2 Request page display the information of requester (urgency, user's rating, user's location)
4. Admin validates the details of requester
4.1 Admin find there is suspicious activity (spam/ fake account)
[If suspicious request]
4.1.1 Admin flagged the user request
4.1.2 System update flagged request in database
4.1.3 Waiting for system administration to notifies user upon cancellation
4.1.4 Retrieve data
4.1.5 Display updated request after remove flagged request
4.2 Admin finds the request info is not suspicious
[else]
4.2.1 Admin pass the approval of requester to next activity
4.2.2 Admin wait upon the approval of requester

Figure 3.1.2.2: Review Carpool Request Data Sequence Diagram

3.1.2.3 Validate Ride Details

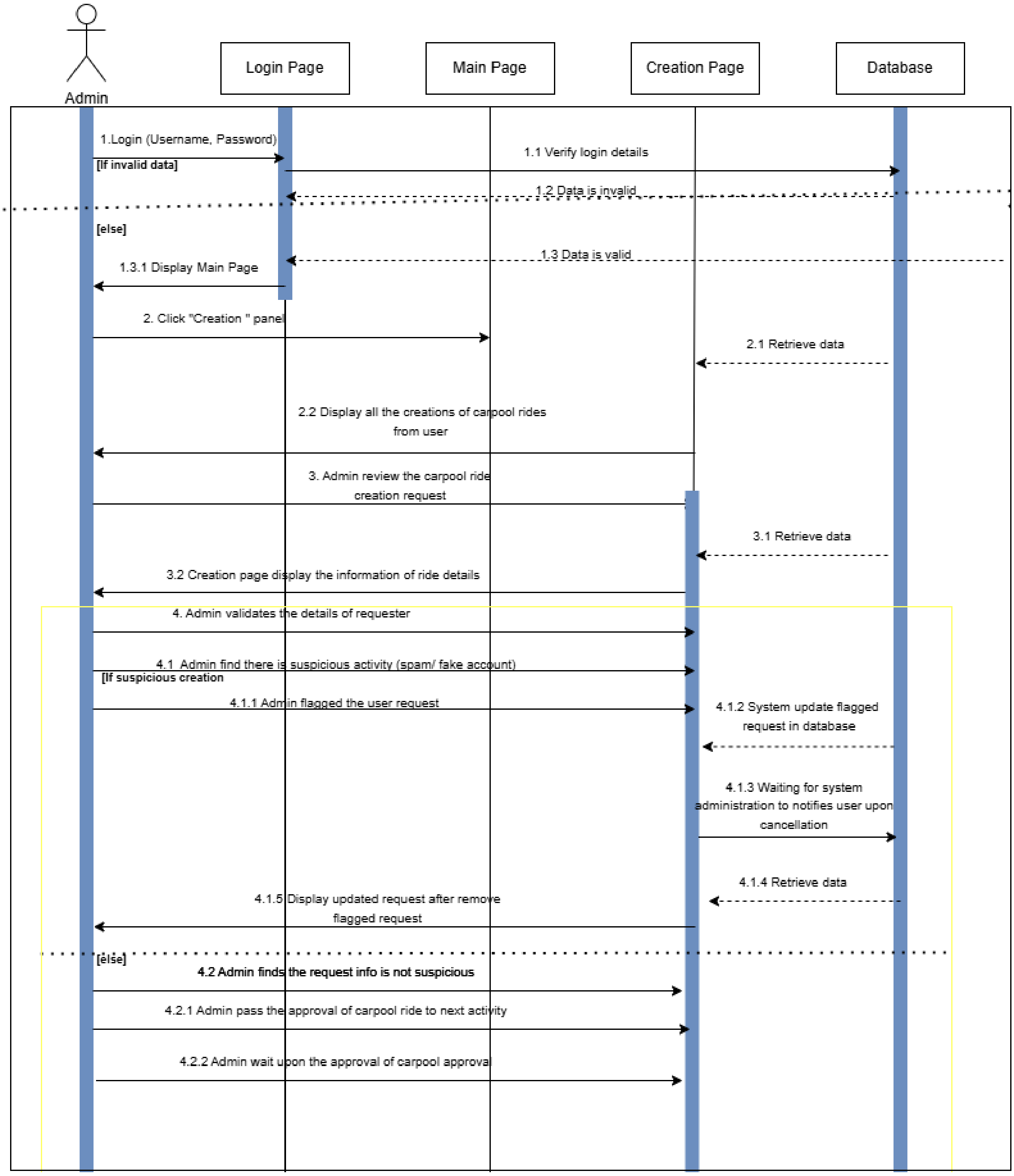

Admin
Login Page
Main Page
Creation Page
Database
1.Login (Username, Password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display Main Page
2. Click "Creation " panel
2.1 Retrieve data
2.2 Display all the creations of carpool rides from user
3. Admin review the carpool ride creation request
3.1 Retrieve data
3.2 Creation page display the information of ride details
4. Admin validates the details of requester
4.1 Admin find there is suspicious activity (spam/ fake account)
[If suspicious creation
4.1.1 Admin flagged the user request
4.1.2 System update flagged request in database
4.1.3 Waiting for system administration to notifies user upon cancellation
4.1.4 Retrieve data
4.1.5 Display updated request after remove flagged request
[else]
4.2 Admin finds the request info is not suspicious
4.2.1 Admin pass the approval of carpool ride to next activity
4.2.2 Admin wait upon the approval of carpool approva

Figure 3.1.2.3: Validate Ride Details Sequence Diagram

3.1.2.4 Approve/Reject Carpools Request

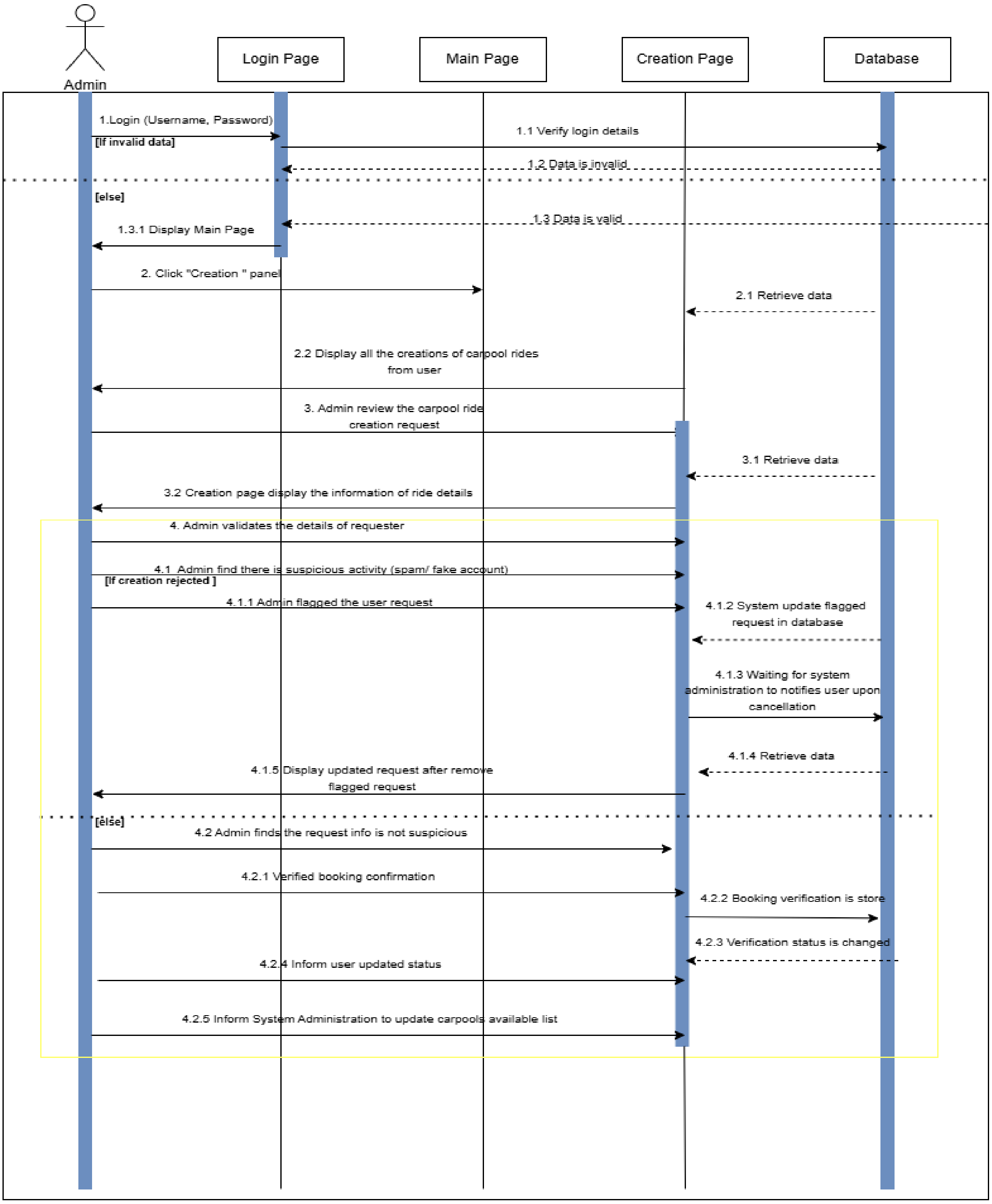

Admin
Login Page
Main Page
Creation Page
Database
1.Login (Username, Password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display Main Page
2. Click "Creation " panel
2.1 Retrieve data
2.2 Display all the creations of carpool rides from user
3. Admin review the carpool ride creation request
3.1 Retrieve data
3.2 Creation page display the information of ride details
4. Admin validates the details of requester
4.1 Admin find there is suspicious activity (spam/ fake account)
[If creation rejected]
4.1.1 Admin flagged the user request
4.1.2 System update flagged request in database
4.1.3 Waiting for system administration to notifies user upon cancellation
4.1.4 Retrieve data
4.1.5 Display updated request after remove flagged request
[else]
4.2 Admin finds the request info is not suspicious
4.2.1 Verified booking confirmation
4.2.2 Booking verification is store
4.2.3 Verification status is changed
4.2.4 Inform user updated status
4.2.5 Inform System Administration to update carpools available list

Figure 3.1.2.4: Approve/Reject Carpool Request Sequence Diagram

3.1.2.5 Receive Parking Approval

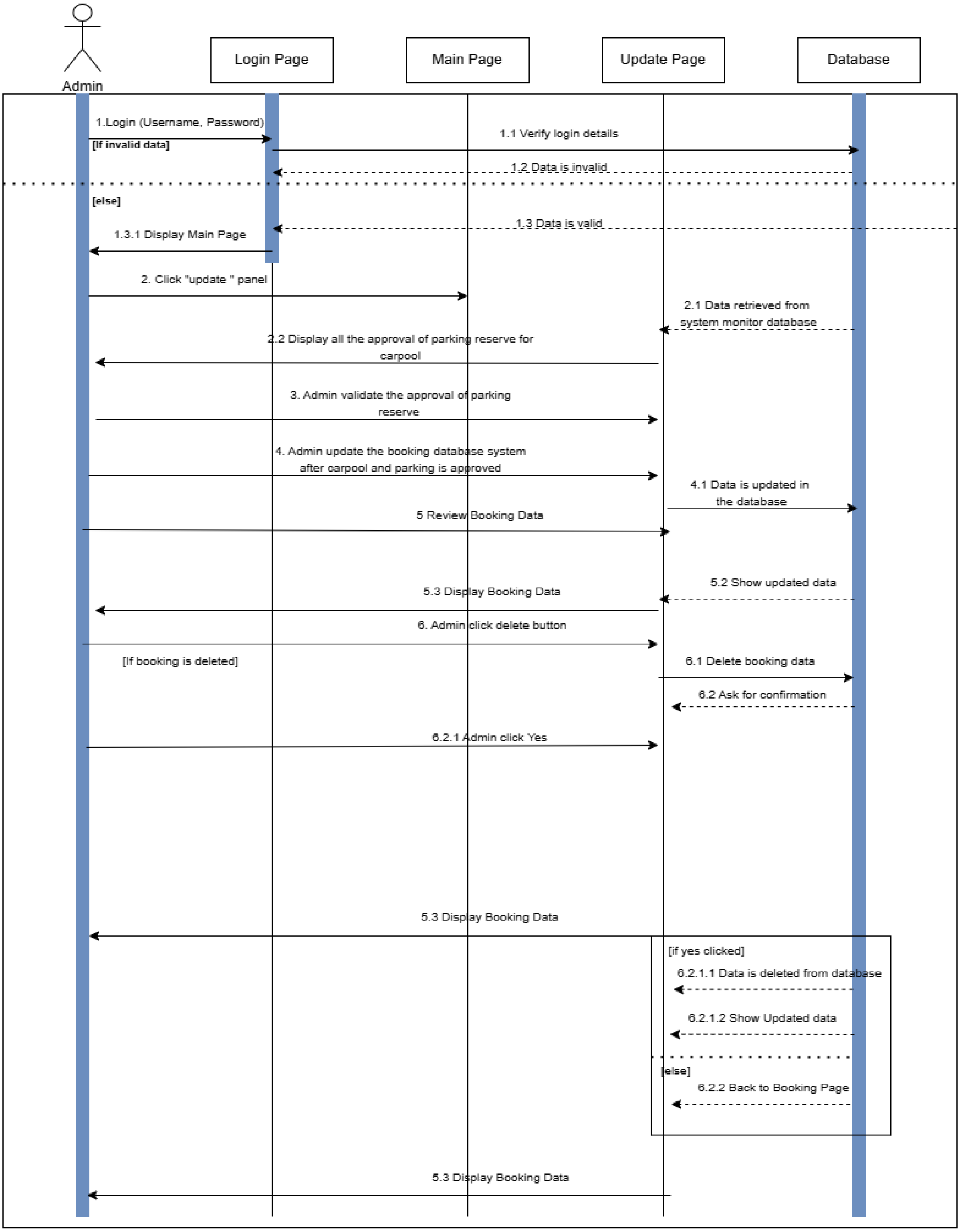

Admin
Login Page
Main Page
Update Page
Database
1.Login (Username, Password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display Main Page
2. Click "update " panel
2.1 Data retrieved from system monitor database
2.2 Display all the approval of parking reserve for carpool
3. Admin validate the approval of parking reserve
4. Admin update the booking database system after carpool and parking is approved
4.1 Data is updated in the database
5 Review Booking Data
5.2 Show updated data
5.3 Display Booking Data
6. Admin click delete button
[If booking is deleted]
6.1 Delete booking data
6.2 Ask for confirmation
6.2.1 Admin click Yes
5.3 Display Booking Data
[if yes clicked]
6.2.1.1 Data is deleted from database
6.2.1.2 Show Updated data
[else]
6.2.2 Back to Booking Page
5.3 Display Booking Data

Figure 3.1.2.5: Receive Parking Approval Sequence Diagram

3.1.2.6 Update & Store Booking Data into Database

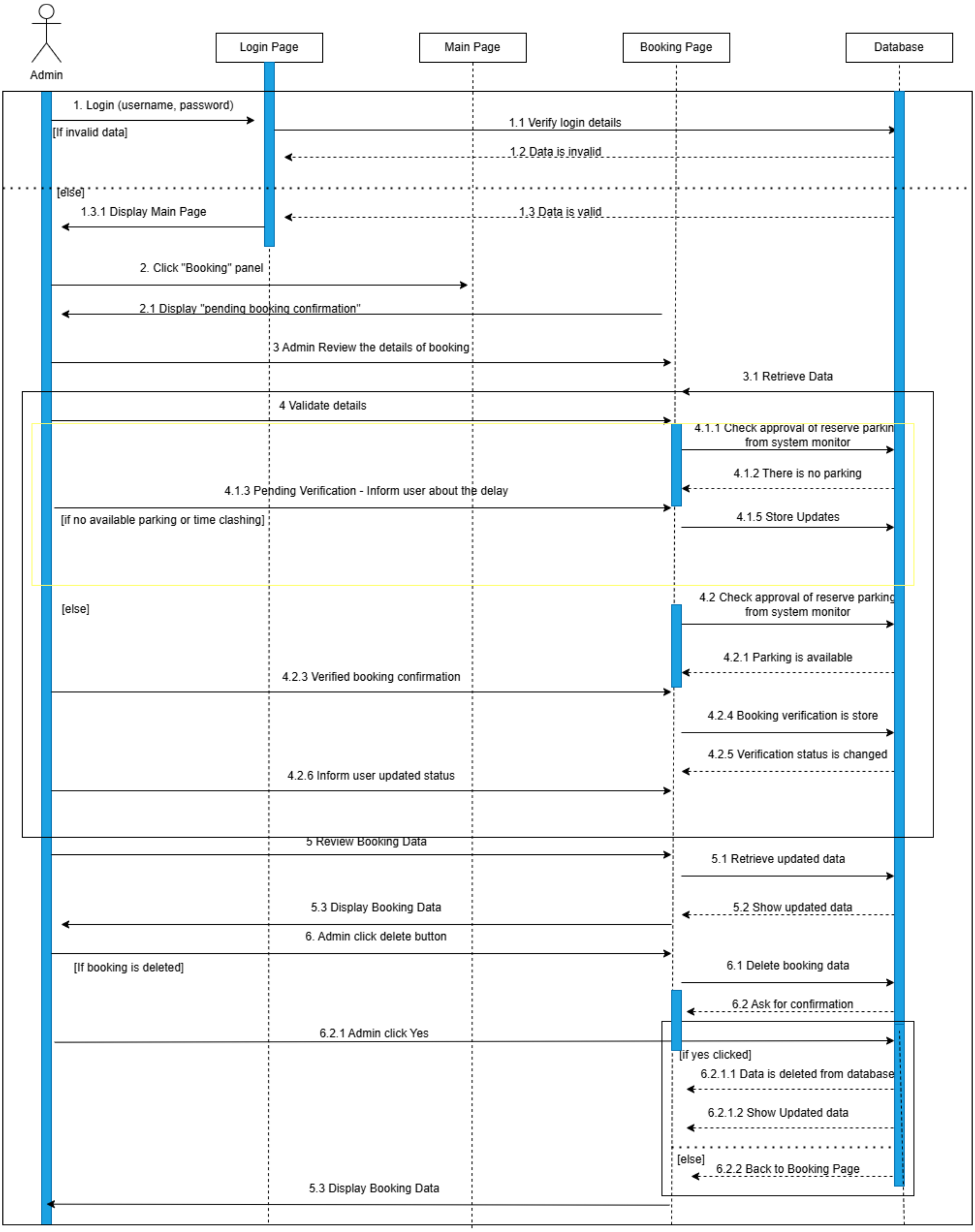

Admin
Login Page
Main Page
Booking Page
Database
1. Login (username, password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3.1 Display Main Page
1.3 Data is valid
2. Click "Booking" panel
2.1 Display "pending booking confirmation"
3 Admin Review the details of booking
3.1 Retrieve Data
4 Validate details
4.1.1 Check approval of reserve parkin from system monitor
4.1.2 There is no parking
4.1.3 Pending Verification - Inform user about the delay
[if no available parking or time clashing]
4.1.5 Store Updates
[else]
4.2 Check approval of reserve parking from system monitor
4.2.1 Parking is available
4.2.3 Verified booking confirmation
4.2.4 Booking verification is store
4.2.5 Verification status is changed
4.2.6 Inform user updated status
5 Review Booking Data
5.1 Retrieve updated data
5.3 Display Booking Data
5.2 Show updated data
6. Admin click delete button
[If booking is deleted]
6.1 Delete booking data
6.2 Ask for confirmation
6.2.1 Admin click Yes
[if yes clicked]
6.2.1.1 Data is deleted from database
6.2.1.2 Show Updated data
[else]
6.2.2 Back to Booking Page
5.3 Display Booking Data

Figure 3.1.2.6: Update & Store Booking Data into Database Sequence Diagram

3.1.2.7 Review & Store Comments or Feedback

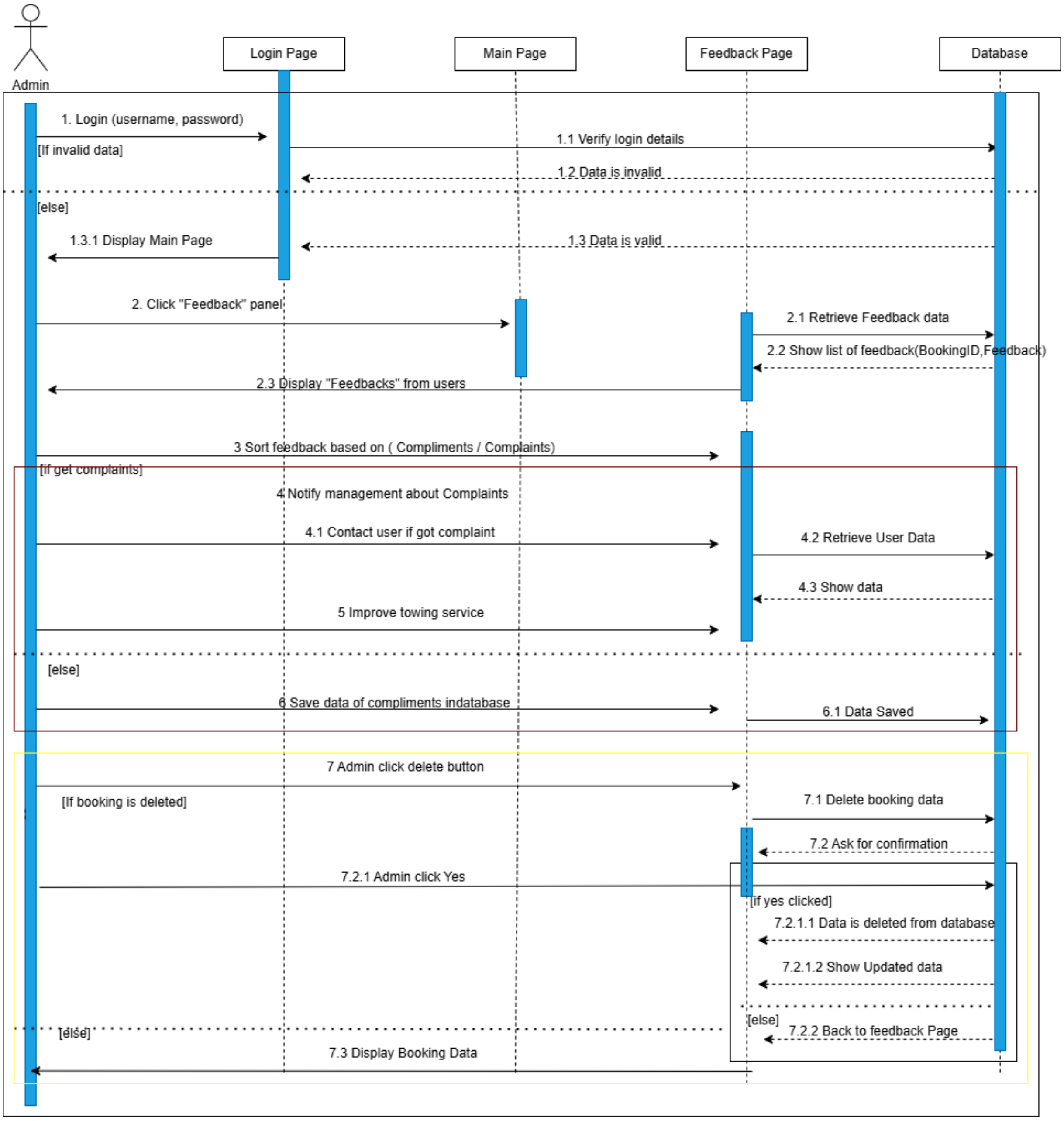

Admin
Login Page
Main Page
Feedback Page
Database
1. Login (username, password)
[If invalid data]
1.1 Verify login details
1.2 Data is invalid
[else]
1.3.1 Display Main Page
1.3 Data is valid
2. Click "Feedback" panel
2.1 Retrieve Feedback data
2.2 Show list of feedback(BookingID,Feedback)
2.3 Display "Feedbacks" from users
3 Sort feedback based on (Compliments / Complaints)
[if get complaints]
4 Notify management about Complaints
4.1 Contact user if got complaint
4.2 Retrieve User Data
4.3 Show data
5 Improve towing service
[else]
6 Save data of compliments indatabase
6.1 Data Saved
7 Admin click delete button
[If booking is deleted]
7.1 Delete booking data
7.2 Ask for confirmation
7.2.1 Admin click Yes
[if yes clicked]
7.2.1.1 Data is deleted from database
7.2.1.2 Show Updated data
[else]
7.2.2 Back to feedback Page
[else]
7.3 Display Booking Data

Figure 3.1.2.7: Review & Store Comments or Feedbacks Sequence Diagram

3.1.2.8 Future Enhancement: Voice BookingAdd commentMore actions

Description:

In the future, the platform will include a **voice booking** functionality. This feature will allow users to make carpool and parking reservations via voice commands, enhancing the accessibility and convenience of the platform. Voice booking will integrate with popular voice assistants, such as **Amazon Alexa**, **Google Assistant**, or **Siri**, allowing users to interact with the system hands-free.

Justification:

The inclusion of voice commands will cater to users who prefer a hands-free experience while driving or multitasking. It will also increase the platform's accessibility, allowing users with disabilities or those in situations where typing or using a screen is difficult to access the system more easily.

Scope:

The voice booking feature will be implemented as a **future enhancement** once the core functionalities of the system (user registration, carpooling, parking reservation, and payment) are fully established. This feature will be available on both mobile and desktop platforms once integrated with the corresponding voice assistants.

Future Actions:

Voice recognition integration: Integration with voice assistant APIs.
User interface modifications: Adding voice command options in the UI.
Testing and validation: Ensuring that the voice commands work accurately in different environments and languages.

3.1.3 System Administrator

3.1.3.1 Update Status to Users

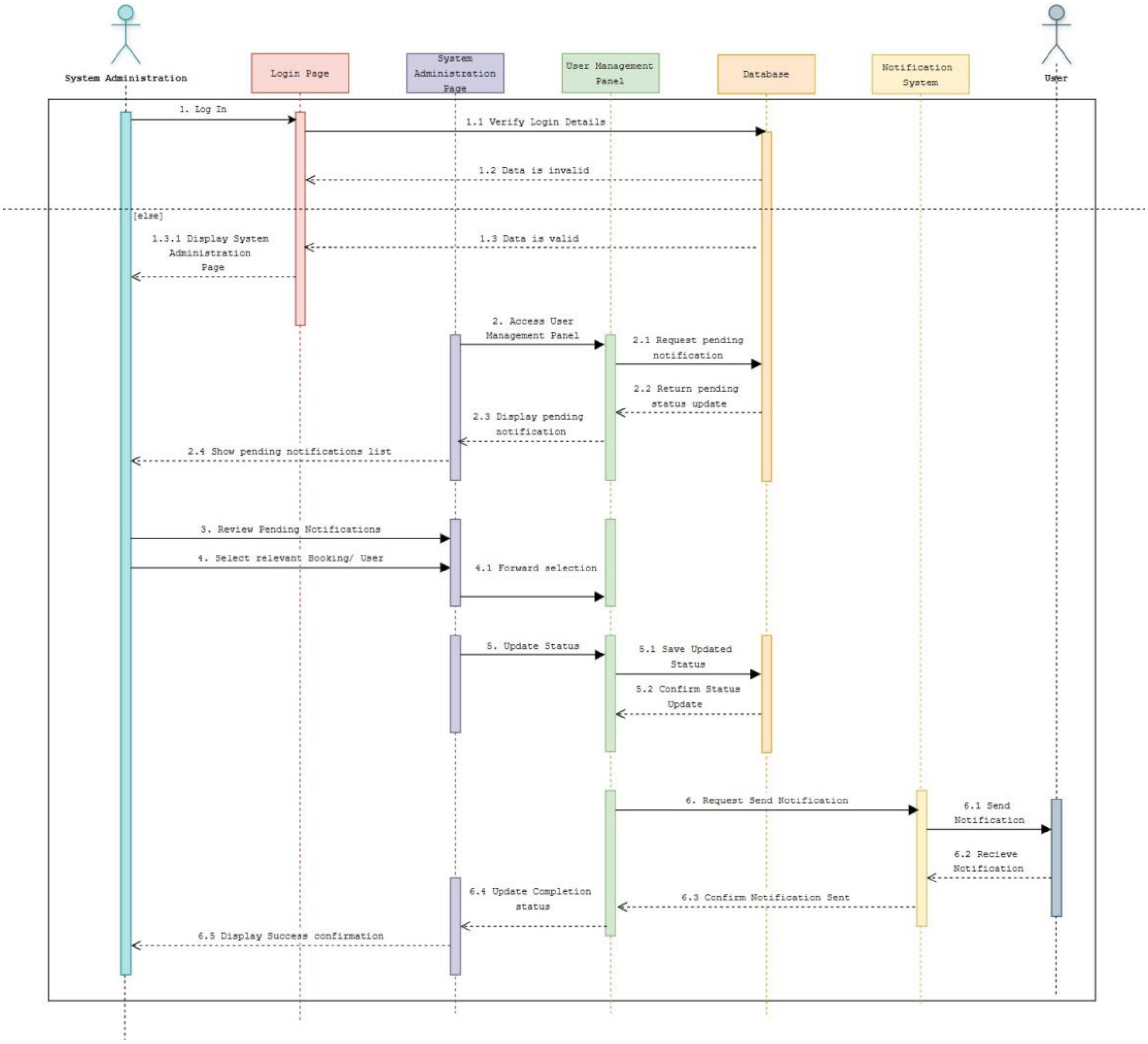

System Administration
Login Page
System Administration Page
User Management Panel
Database
Notification System
User
1. Log In
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display System Administration Page
2. Access User Management Panel
2.1 Request pending notification
2.2 Return pending status update
2.3 Display pending notification
2.4 Show pending notifications list
3. Review Pending Notifications
4. Select relevant Booking/ User
4.1 Forward selection
5. Update Status
5.1 Save Updated Status
5.2 Confirm Status Update
6. Request Send Notification
6.1 Send Notification
6.2 Recieve Notification
6.3 Confirm Notification Sent
6.4 Update Completion status
6.5 Display Success confirmation

Figure 3.1.3.1: Update Status to Users Sequence Diagram

3.1.3.2 Retrieve Availability

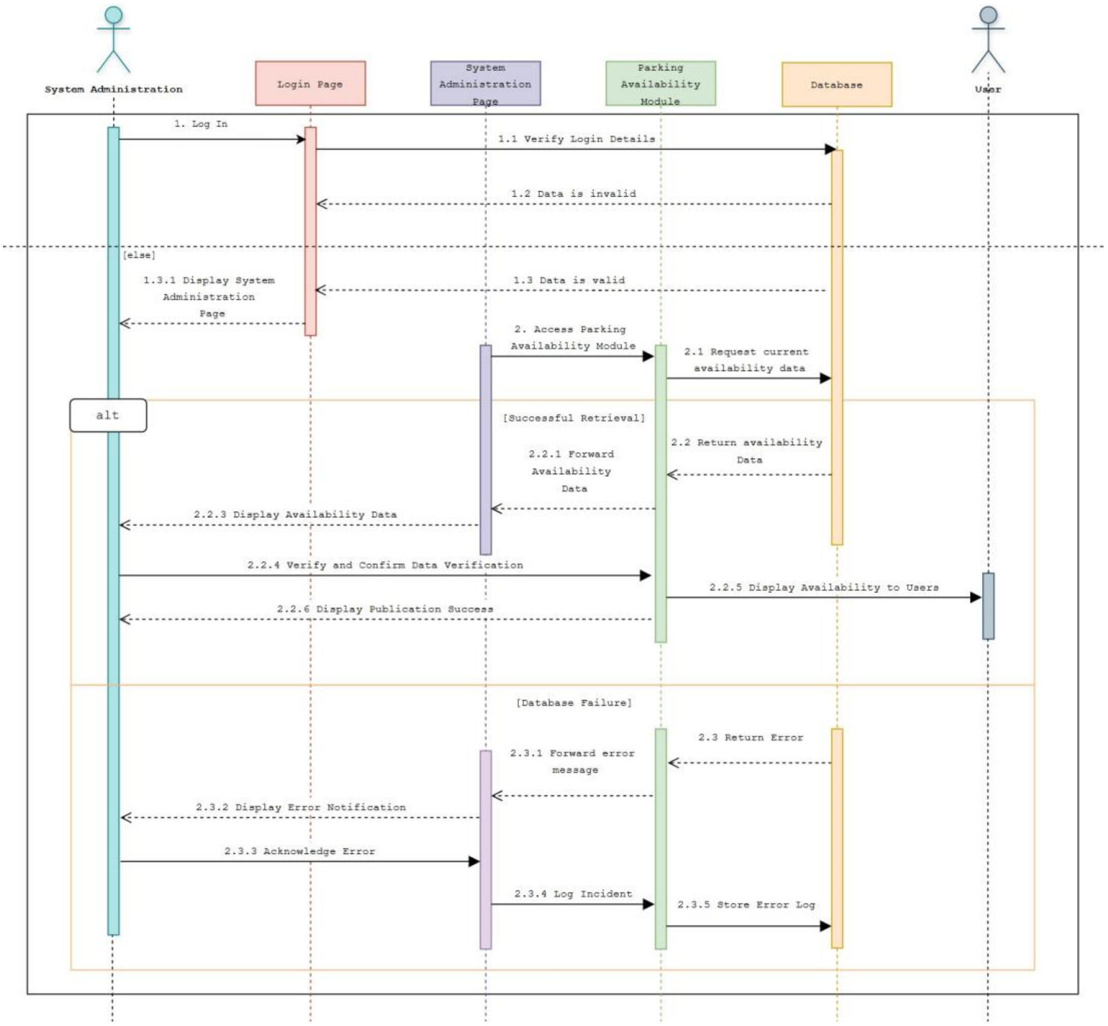

System Administration
Login Page
System Administration Page
Parking Availability Module
Database
User
1. Log In
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3.1 Display System Administration Page
1.3 Data is valid
2. Access Parking Availability Module
2.1 Request current availability data
alt
[Successful Retrieval]
2.2 Return availability Data
2.2.1 Forward Availability Data
2.2.3 Display Availability Data
2.2.4 Verify and Confirm Data Verification
2.2.5 Display Availability to Users
2.2.6 Display Publication Success
[Database Failure]
2.3 Return Error
2.3.1 Forward error message
2.3.2 Display Error Notification
2.3.3 Acknowledge Error
2.3.4 Log Incident
2.3.5 Store Error Log

Figure 3.1.3.2: Retrieve Availability Sequence Diagram

3.1.3.3 Display Slot

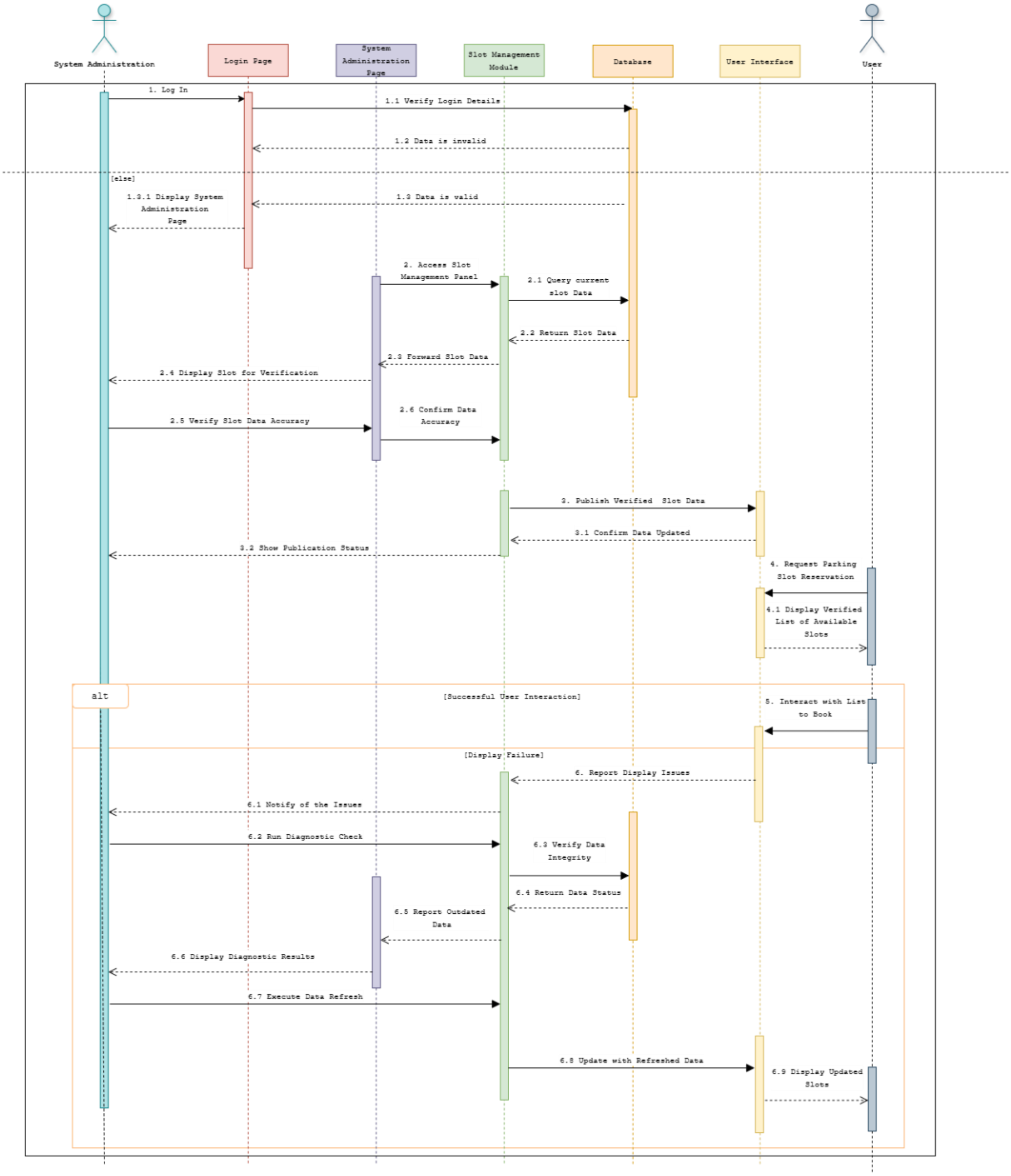

System Administration
Login Page
System Administration Page
Slot Management Module
Database
User Interface
User
1. Log In
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display System Administration Page
2. Access Slot Management Panel
2.1 Query current slot Data
2.2 Return Slot Data
2.3 Forward Slot Data
2.4 Display Slot for Verification
2.5 Verify Slot Data Accuracy
2.6 Confirm Data Accuracy
3. Publish Verified Slot Data
3.1 Confirm Data Updated
3.2 Show Publication Status
4. Request Parking Slot Reservation
4.1 Display Verified List of Available Slots
alt
[Successful User Interaction]
5. Interact with List to Book
[Display Failure]
6. Report Display Issues
6.1 Notify of the Issues
6.2 Run Diagnostic Check
6.3 Verify Data Integrity
6.4 Return Data Status
6.5 Report Outdated Data
6.6 Display Diagnostic Results
6.7 Execute Data Refresh
6.8 Update with Refreshed Data
6.9 Display Updated Slots

Figure 3.1.3.3: Display Slot Sequence Diagram

3.1.3.4 Calculate Total Cost

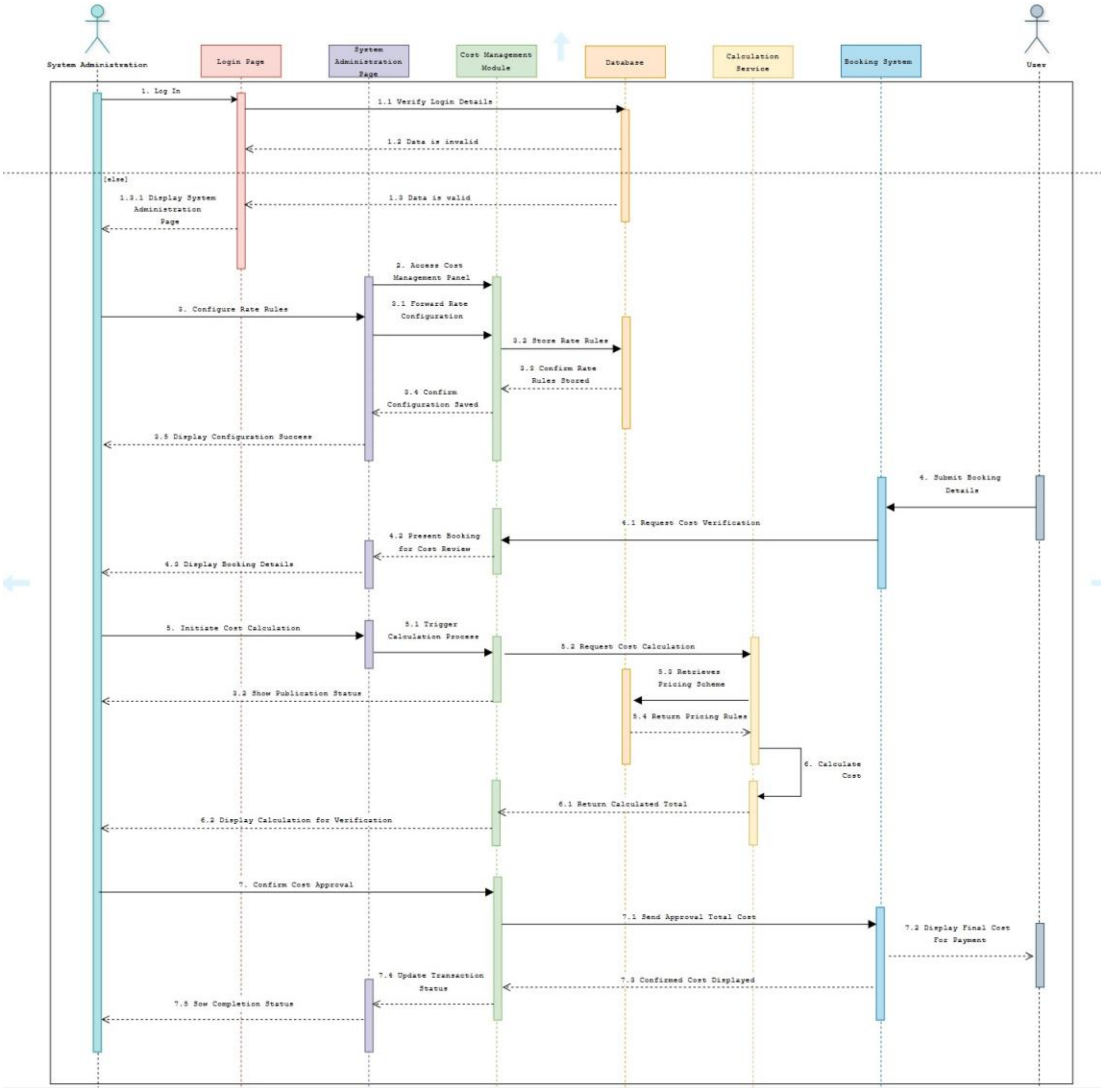

System Administration
Login Page
System Administration Page
Cost Management Module
Database
Calculation Service
Booking System
User
1. Log In
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3.1 Display System Administration Page
1.3 Data is valid
2. Access Cost Management Panel
3. Configure Rate Rules
3.1 Forward Rate Configuration
3.2 Store Rate Rules
3.3 Confirm Rate Rules Stored
3.4 Confirm Configuration Saved
3.5 Display Configuration Success
4. Submit Booking Details
4.1 Request Cost Verification
4.2 Present Booking for Cost Review
4.3 Display Booking Details
5. Initiate Cost Calculation
5.1 Trigger Calculation Process
5.2 Request Cost Calculation
5.3 Retrieves Pricing Scheme
3.2 Show Publication Status
5.4 Return Pricing Rules
6. Calculate Cost
6.1 Return Calculated Total
6.2 Display Calculation for Verification
7. Confirm Cost Approval
7.1 Send Approval Total Cost
7.2 Display Final Cost For Payment
7.4 Update Transaction Status
7.3 Confirmed Cost Displayed
7.5 Sow Completion Status

Figure 3.1.3.4: Calculate Total Cost Sequence Diagram

3.1.3.5 Verified Payment

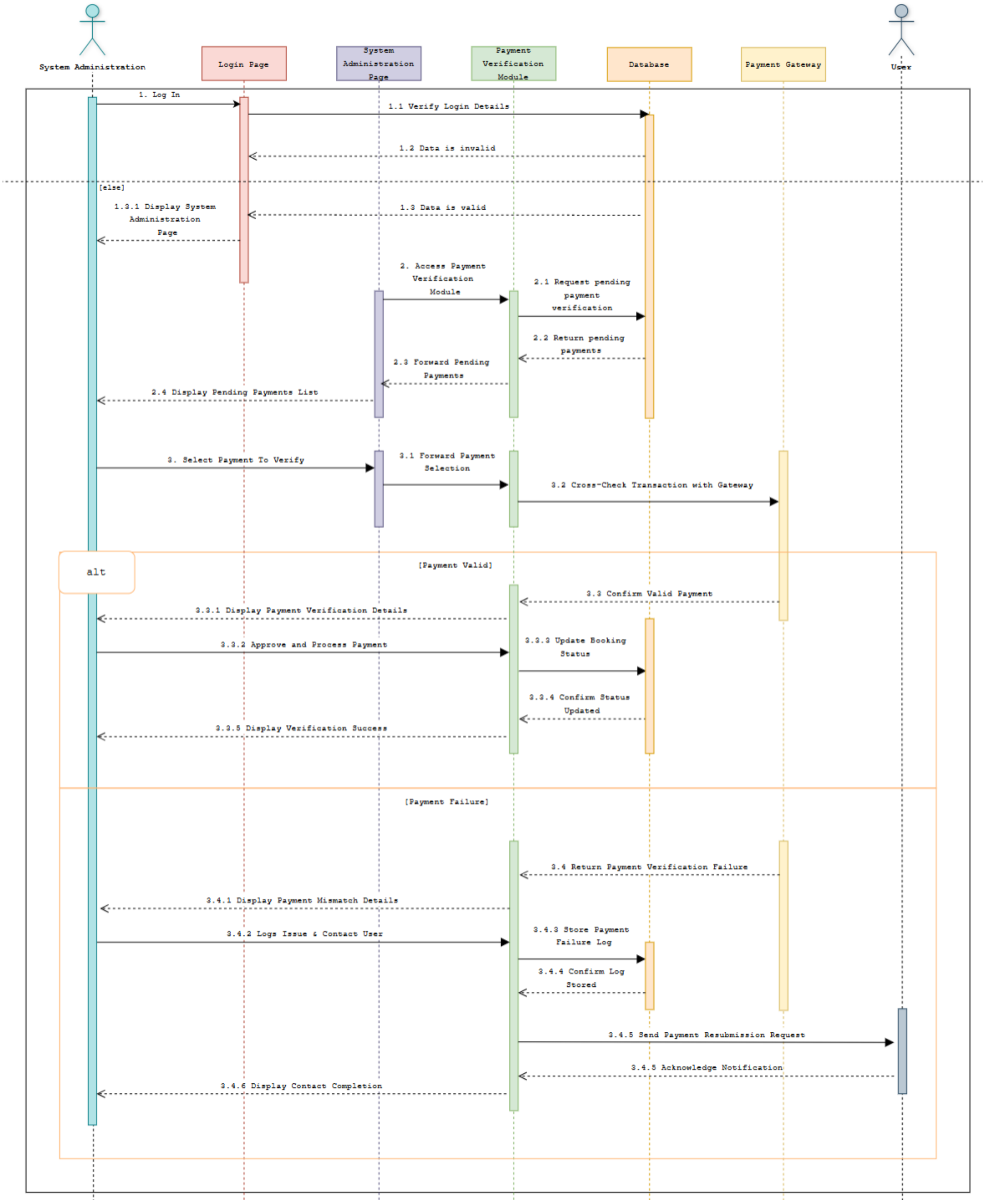

System Administration
Login Page
System Administration Page
Payment Verification Module
Database
Payment Gateway
User
1. Log In
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display System Administration Page
2. Access Payment Verification Module
2.1 Request pending payment verification
2.2 Return pending payments
2.3 Forward Pending Payments
2.4 Display Pending Payments List
3. Select Payment To Verify
3.1 Forward Payment Selection
3.2 Cross-Check Transaction with Gateway
alt
[Payment Valid]
3.3 Confirm Valid Payment
3.3.1 Display Payment Verification Details
3.3.2 Approve and Process Payment
3.3.3 Update Booking Status
3.3.4 Confirm Status Updated
3.3.5 Display Verification Success
[Payment Failure]
3.4 Return Payment Verification Failure
3.4.1 Display Payment Mismatch Details
3.4.2 Logs Issue & Contact User
3.4.3 Store Payment Failure Log
3.4.4 Confirm Log Stored
3.4.5 Send Payment Resubmission Request
3.4.5 Acknowledge Notification
3.4.6 Display Contact Completion

Figure 3.1.3.5: Verified Payment Sequence Diagram

3.1.3.6 Notify User with Invoice

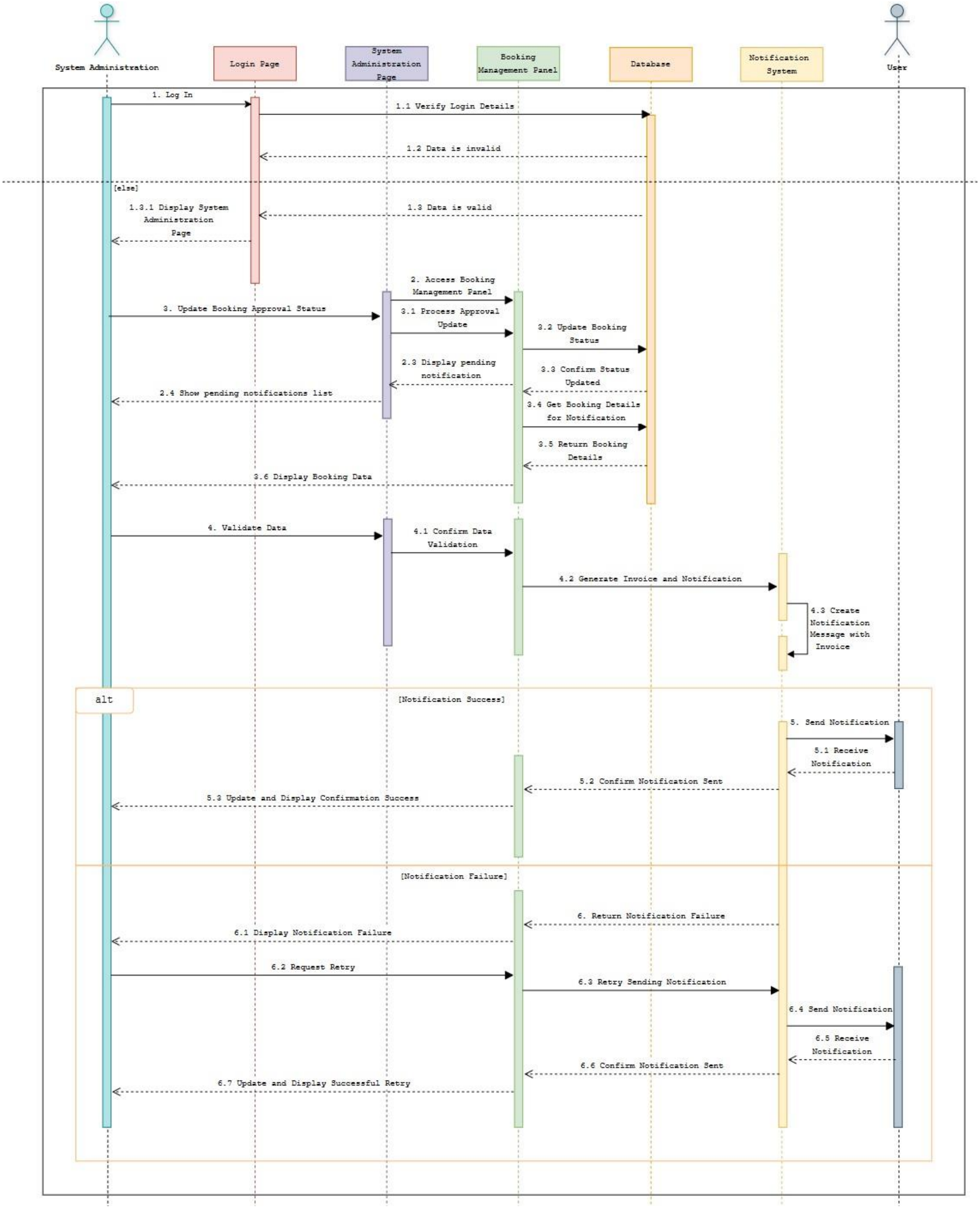

System Administration
Login Page
System Administration Page
Booking Management Panel
Database
Notification System
User
1. Log In
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3 Data is valid
1.3.1 Display System Administration Page
2. Access Booking Management Panel
3. Update Booking Approval Status
3.1 Process Approval Update
3.2 Update Booking Status
2.3 Display pending notification
3.3 Confirm Status Updated
2.4 Show pending notifications list
3.4 Get Booking Details for Notification
3.5 Return Booking Details
3.6 Display Booking Data
4. Validate Data
4.1 Confirm Data Validation
4.2 Generate Invoice and Notification
4.3 Create Notification Message with Invoice
alt
[Notification Success]
5. Send Notification
5.1 Receive Notification
5.2 Confirm Notification Sent
5.3 Update and Display Confirmation Success
[Notification Failure]
6. Return Notification Failure
6.1 Display Notification Failure
6.2 Request Retry
6.3 Retry Sending Notification
6.4 Send Notification
6.5 Receive Notification
6.6 Confirm Notification Sent
6.7 Update and Display Successful Retry

Figure 3.1.3.6: Notify User with Invoice Sequence Diagram

3.1.3.7 Monitor Backend Log

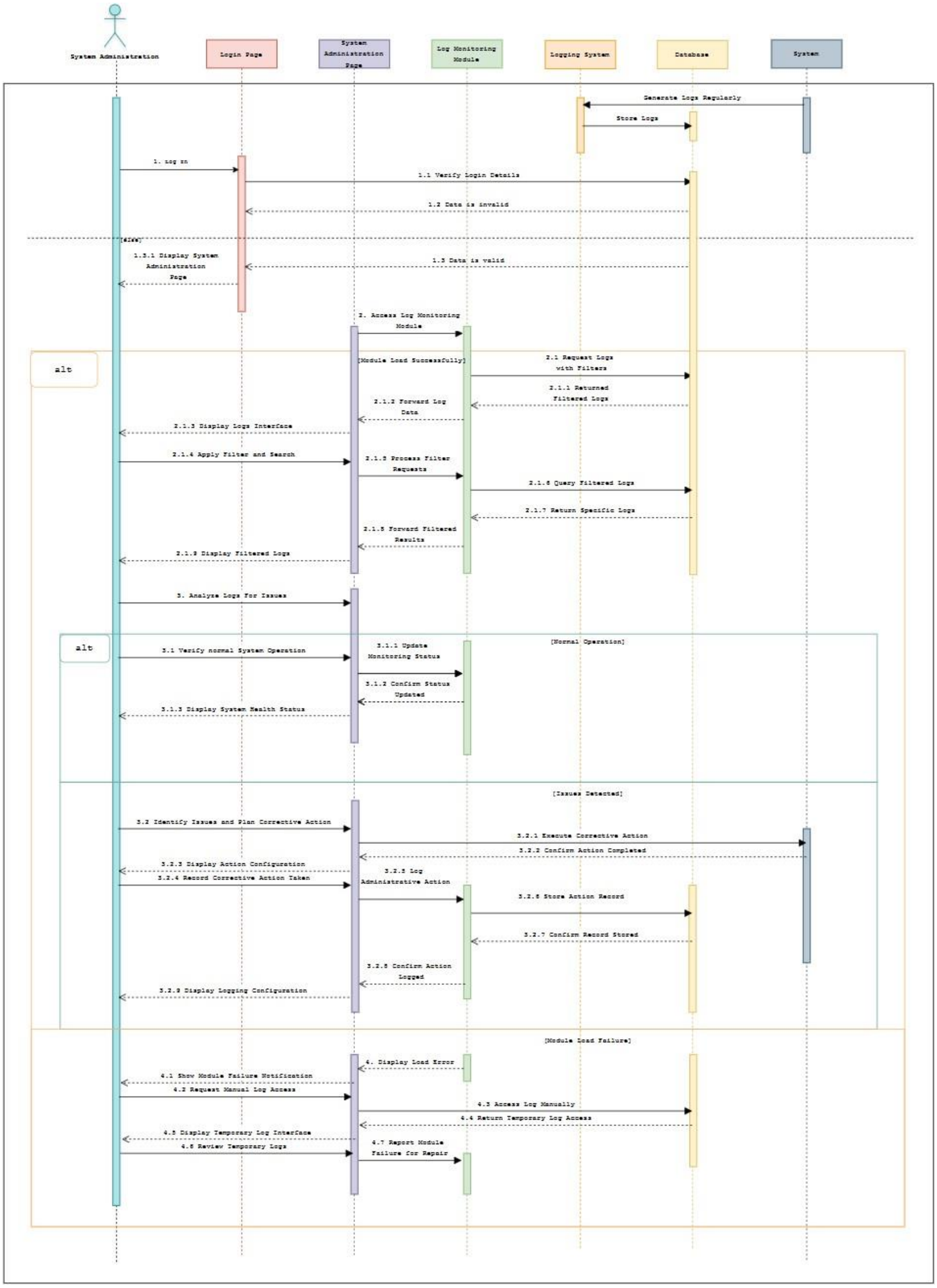

System Administration
Login Page
System Administration Page
Log Monitoring Module
Logging System
Database
System
Generate Logs Regularly
Store Logs
1. Log in
1.1 Verify Login Details
1.2 Data is invalid
[else]
1.3.1 Display System Administration Page
1.3 Data is valid
2. Access Log Monitoring Module
alt
[Module Load Successfully]
2.1 Request Logs with Filters
2.1.1 Returned Filtered Logs
2.1.2 Forward Log Data
2.1.3 Display Logs Interface
2.1.4 Apply Filter and Search
2.1.5 Process Filter Requests
2.1.6 Query Filtered Logs
2.1.7 Return Specific Logs
2.1.8 Forward Filtered Results
2.1.9 Display Filtered Logs
3. Analyze Logs For Issues
alt
[Normal Operation]
3.1 Verify normal System Operation
3.1.1 Update Monitoring Status
3.1.2 Confirm Status Updated
3.1.3 Display System Health Status
[Issues Detected]
3.2 Identify Issues and Plan Corrective Action
3.2.1 Execute Corrective Action
3.2.2 Confirm Action Completed
3.2.3 Display Action Configuration
3.2.4 Record Corrective Action Taken
3.2.5 Log Administrative Action
3.2.6 Store Action Record
3.2.7 Confirm Record Stored
3.2.8 Confirm Action Logged
3.2.9 Display Logging Configuration
[Module Load Failure]
4. Display Load Error
4.1 Show Module Failure Notification
4.2 Request Manual Log Access
4.3 Access Log Manually
4.4 Return Temporary Log Access
4.5 Display Temporary Log Interface
4.6 Review Temporary Logs
4.7 Report Module Failure for Repair

Figure 3.1.3.7: Monitor Backend Log Sequence Diagram

3.1.4 System Monitor

3.1.4.1 Monitor Parking Data Feed

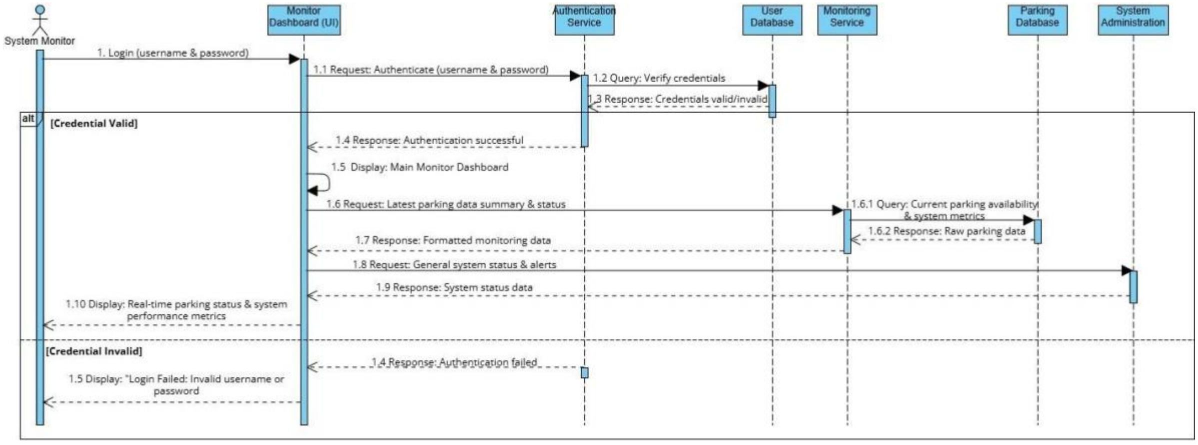

System Monitor
Monitor Dashboard (UI)
Authentication Service
User Database
Monitoring Service
Parking Database
System Administration
1. Login (username & password)
1.1 Request: Authenticate (username & password)
1.2 Query: Verify credentials
1.3 Response: Credentials valid/invalid
alt
[Credential Valid]
1.4 Response: Authentication successful
1.5 Display: Main Monitor Dashboard
1.6 Request: Latest parking data summary & status
1.6.1 Query: Current parking availability & system metrics
1.6.2 Response: Raw parking data
1.7 Response: Formatted monitoring data
1.8 Request: General system status & alerts
1.9 Response: System status data
1.10 Display: Real-time parking status & system performance metrics
[Credential Invalid]
1.4 Response: Authentication failed
1.5 Display: "Login Failed: Invalid username or password

Figure 3.1.4.1: Monitor Parking Data Feed Sequence Diagram

3.1.4.2 Confirm & Update Available Slots

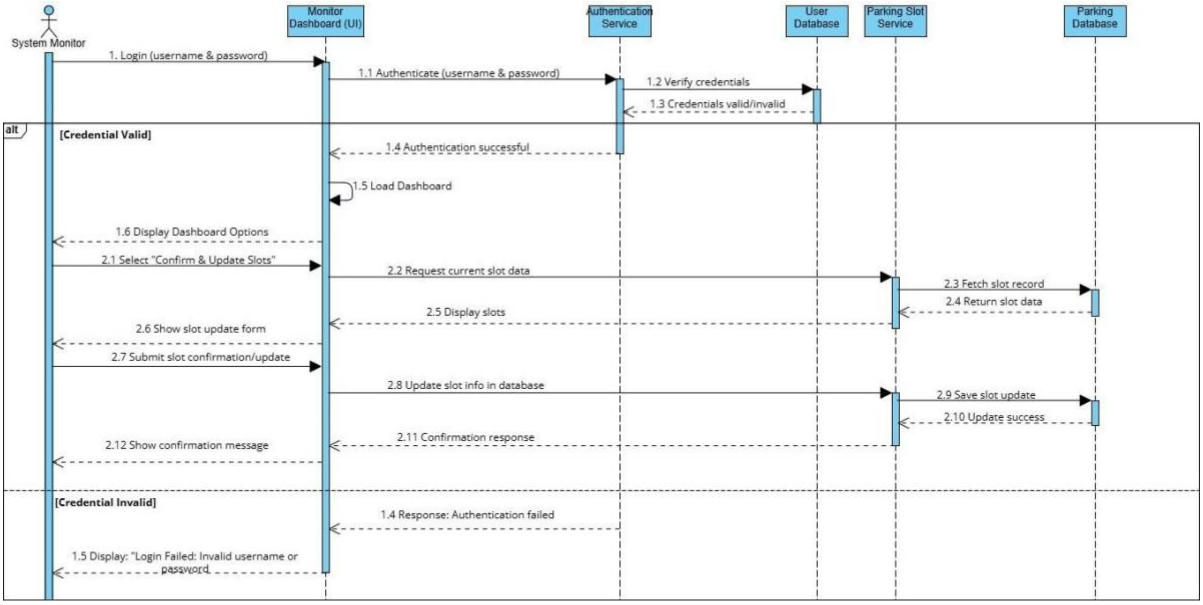

System Monitor
Monitor Dashboard (UI)
Authentication Service
User Database
Parking Slot Service
Parking Database
1. Login (username & password)
1.1 Authenticate (username & password)
1.2 Verify credentials
1.3 Credentials valid/invalid
alt
[Credential Valid]
1.4 Authentication successful
1.5 Load Dashboard
1.6 Display Dashboard Options
2.1 Select "Confirm & Update Slots"
2.2 Request current slot data
2.3 Fetch slot record
2.4 Return slot data
2.5 Display slots
2.6 Show slot update form
2.7 Submit slot confirmation/update
2.8 Update slot info in database
2.9 Save slot update
2.10 Update success
2.11 Confirmation response
2.12 Show confirmation message
[Credential Invalid]
1.4 Response: Authentication failed
1.5 Display: "Login Failed: Invalid username or password

Figure 3.1.4.2: Confirm & Update Available Slots Sequence Diagram

3.1.4.3 Receive User's Parking Booking

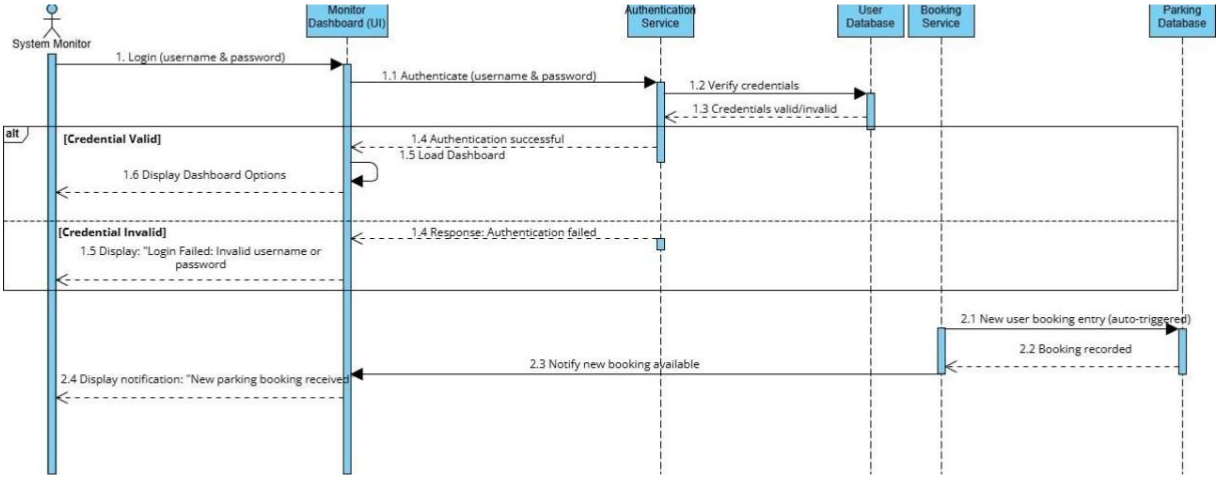

System Monitor
Monitor Dashboard (UI)
Authentication Service
User Database
Booking Service
Parking Database
1. Login (username & password)
1.1 Authenticate (username & password)
1.2 Verify credentials
1.3 Credentials valid/invalid
alt
[Credential Valid]
1.4 Authentication successful
1.5 Load Dashboard
1.6 Display Dashboard Options
[Credential Invalid]
1.4 Response: Authentication failed
1.5 Display: "Login Failed: Invalid username or password
2.1 New user booking entry (auto-triggered)
2.2 Booking recorded
2.3 Notify new booking available
2.4 Display notification: "New parking booking received

Figure 3.1.4.3: Receive User's Parking Booking Sequence Diagram

3.1.4.4 Review Parking Booking

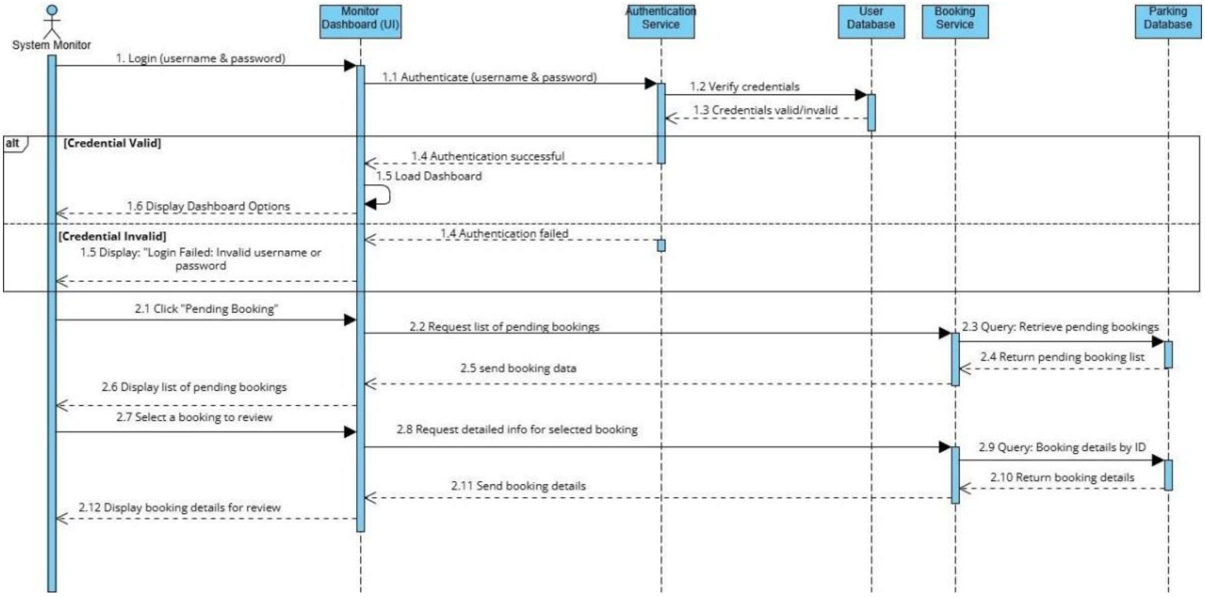

System Monitor
Monitor Dashboard (UI)
Authentication Service
User Database
Booking Service
Parking Database
1. Login (username & password)
1.1 Authenticate (username & password)
1.2 Verify credentials
1.3 Credentials valid/invalid
alt
[Credential Valid]
1.4 Authentication successful
1.5 Load Dashboard
1.6 Display Dashboard Options
[Credential Invalid]
1.4 Authentication failed
1.5 Display: "Login Failed: Invalid username or password
2.1 Click "Pending Booking"
2.2 Request list of pending bookings
2.3 Query: Retrieve pending bookings
2.4 Return pending booking list
2.5 send booking data
2.6 Display list of pending bookings
2.7 Select a booking to review
2.8 Request detailed info for selected booking
2.9 Query: Booking details by ID
2.10 Return booking details
2.11 Send booking details
2.12 Display booking details for review

Figure 3.1.4.4: Review Parking Booking Sequence Diagram

3.1.4.5 Approve Parking Booking

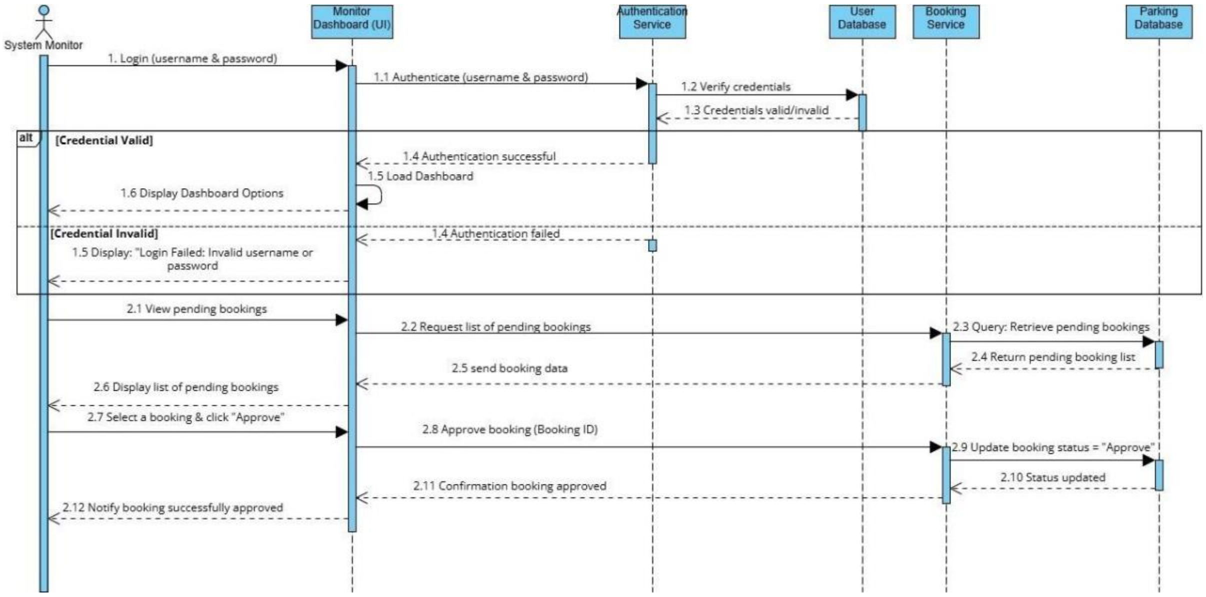

System Monitor
Monitor Dashboard (UI)
Authentication Service
User Database
Booking Service
Parking Database
1. Login (username & password)
1.1 Authenticate (username & password)
1.2 Verify credentials
1.3 Credentials valid/invalid
alt
[Credential Valid]
1.4 Authentication successful
1.5 Load Dashboard
1.6 Display Dashboard Options
[Credential Invalid]
1.4 Authentication failed
1.5 Display: "Login Failed: Invalid username or password
2.1 View pending bookings
2.2 Request list of pending bookings
2.3 Query: Retrieve pending bookings
2.4 Return pending booking list
2.5 send booking data
2.6 Display list of pending bookings
2.7 Select a booking & click "Approve"
2.8 Approve booking (Booking ID)
2.9 Update booking status = "Approve"
2.10 Status updated
2.11 Confirmation booking approved
2.12 Notify booking successfully approved

Figure 3.1.4.5: Approve Parking Booking Sequence Diagram

3.1.4.6 Update Data of Parking Reserved

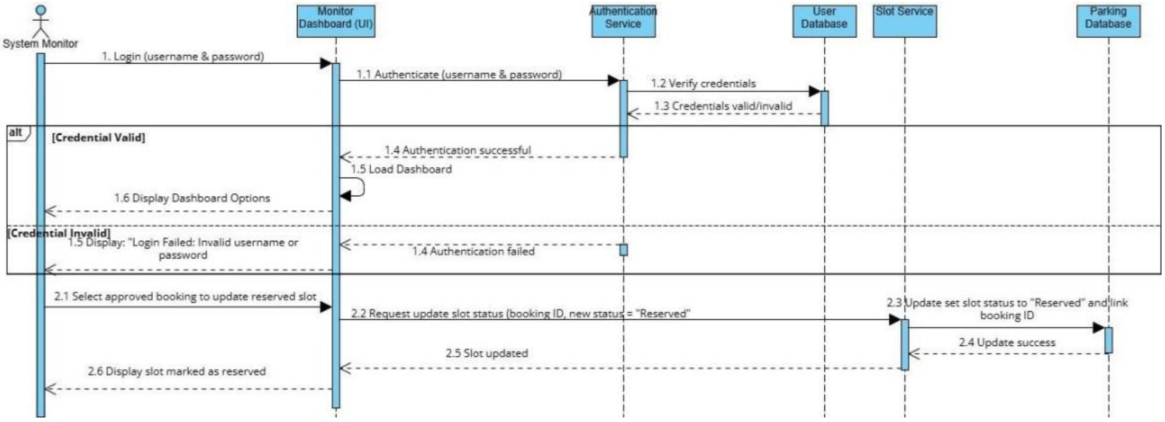

System Monitor
Monitor Dashboard (UI)
Authentication Service
User Database
Slot Service
Parking Database
1. Login (username & password)
1.1 Authenticate (username & password)
1.2 Verify credentials
1.3 Credentials valid/invalid
alt
[Credential Valid]
1.4 Authentication successful
1.5 Load Dashboard
1.6 Display Dashboard Options
[Credential Invalid]
1.5 Display: "Login Failed: Invalid username or password
1.4 Authentication failed
2.1 Select approved booking to update reserved slot
2.2 Request update slot status (booking ID, new status = "Reserved"
2.3 Update set slot status to "Reserved" and link booking ID
2.4 Update success
2.5 Slot updated
2.6 Display slot marked as reserved

Figure 3.1.4.6: Update Data of Parking Reserved Sequence Diagram

3.2 Performance requirements

3.2.1 User

Performance Req No.	Description
PR1	The system must authenticate users and load the dashboard within 2 seconds after successful login.
PR2	Users must be able to fill and submit their personal and ride-related information with confirmation received within 3 seconds.
PR3	Users must be able to create a carpool group, and the request should be submitted and acknowledged by the system within 4 seconds.
PR4	When joining a carpool group, the system should check for availability and respond within 3 seconds.
PR5	Users must be able to view and reserve available parking slots, and get confirmation within 3 seconds.
PR6	Payment processing should be completed and confirmation shown to the user within 5 seconds.
PR7	Feedback submitted by the user should be stored and confirmation displayed within 2 seconds.

Table 3.2.1: Performance Requirement for User

3.2.2 Administrator

Performance Req No.	Description	Metric
PR8	The administrator must update and store users’ data (name, contact, status) securely in the database.	≤ **2 seconds** for single-user updates.
PR9	The administrator must review user-submitted carpool requests, including requester details, date, time, and locations.	≤ **3 seconds** to load and display request data.
PR10	The administrator must validate ride details for completeness and accuracy before approval.	≤ **4 seconds** to complete validation workflow.
PR11	The administrator must approve/reject carpool requests based on submitted ride details.	≤ **2 seconds** to update request status.
PR12	The administrator must receive and view parking approval requests linked to carpool bookings.	≤ **1 second** latency for real-time notifications.
PR13	The administrator must update booking details and store them securely in the database.	≤ **3 seconds** for transaction completion.
PR14	The administrator must review and store user comments/feedback for future reference.	≤ **2 seconds** to log and archive feedback.

Table 3.2.2 : Performance Requirement for Administrator

3.2.3 System Administrator

Performance Req No.	Description
PR15	The system must allow the System Administrator to update a user's status (e.g., booking, payment, approval) and send notifications within 2 seconds of initiating the action
PR16	Real-time slot availability data must be retrieved and displayed to the System Administrator within 1 second of request, assuming a responsive database.
PR17	Users must be able to see the displayed list of slots updated by the System Administrator within 2 seconds of availability confirmation
PR18	The system must calculate the total cost of a booking based on parameters (e.g., time, slot type, carpooling status) within 1 second of receiving user inputs.
PR19	Payment verification must be completed within 2 seconds, including cross-checking with the payment gateway.
PR20	The system must send booking confirmations or invoices via email or platform notifications to users within 1 second of validation
PR21	System logs must be accessible to the System Administrator with filter features (by date, action, type) within 3 seconds, even under high load.

Table 3.2.3 : Performance Requirement for System Administration

3.2.4 System Monitor

Performance Req No.	Description
PR22	The System Monitor must receive, process, and validate incoming parking data feeds within 1 second of input.
PR23	The System Monitor needs to detect significant data changes and verify updates internally within 2 seconds.
PR24	The System Monitor must approve or reject parking booking requests after validation within 1 second of review completion.
PR25	The System Monitor must update the Parking Database with slot availability changes within 1 second of verification.
PR26	Notifications for exceptional events (i.e., parking full, sensor faults) must be delivered to the System Administrator within 1 second of event detection.
PR27	For concurrent streams of data from up to X parking zones, the System Monitor must complete processing and updating the databases within 2 seconds for each area.

Table 3.2.4 : Performance Requirement for System Monitor

3.3 Usability Requirements

The system shall be user-friendly and easy to navigate for all types of users, including admins, users, and system monitors. All buttons, fields, and menus should have clear labels and instructions, and the interface should be simple and easy to use. To guarantee usability across platforms, the system needs to be responsive and available on desktop and mobile devices. After completing tasks like making requests, altering data, or getting approvals, users ought to get prompt response or confirmation messages. The system should minimize human input by providing features like dropdown menus and auto-fill where appropriate in order to lessen user effort. When mistakes occur, the system must provide concise, informative notifications together with easy-to-follow instructions on how to fix the problem. Additionally, the interface must comply with WCAG 2.1 AA standards to support users with different needs, and overall system response time should not exceed 2 seconds to ensure a smooth and efficient user experience.

3.4 Interface Requirements

3.4.1 External Interface

This section defines the engagement of the system with external world entities, including other software systems, users, hardware, and communication interface. This defines how the system interacts with external services such as authentication systems and sensor networks, how the users engage the system via different devices, and how data sharing is managed to deliver security, accuracy, and reliability. All these interfaces are essential to facilitate seamless integration and effective functioning of the system in its environment.

3.4.1.1 System Interface

The system will interact with several other systems to provide smooth functionality. It will be integrated into MMU's centralized authentication system to authenticate user credentials and provide secure access to the system. The system will communicate with the Parking Management System through well-documented APIs, providing real-time sharing of parking availability, booking status, and updates. The system will also be integrated with digital identification verification services for user identification validation during the ride-sharing and parking reservation operations. Cloud messaging services will be utilized to offer timely notices and alerts to users and managers. The interfaces will be in harmony with standard communication protocols and data forms to facilitate secure and strong data transfer to all connected elements.

3.4.1.2 User Interface

The system shall provide an intuitive and responsive interface accessible through both desktop and mobile browsers. The interface will vary slightly depending on the role (User, Administrator, System Administrator, System Monitor), ensuring role-based access and operations and the interface must comply with WCAG 2.1 AA for accessibility

3.4.1.2.1 General Layout

- The main layout will include a **dashboard**, **navigation panel**, **main content area**, and **notifications** section.
- Consistent color themes and clear typography will be used for readability.
- Responsive design will adjust layout and components according to screen size.

3.4.1.2.2 Interfaces per Role

User Interface

- **Login/Register Page** – Standard fields for authentication.
- **Dashboard** – Displays current ride bookings, payment status, notifications.
- **Carpool Module** – Form for creating/joining carpools, viewing carpool status.
- **Parking Slot Booking** – Real-time availability map and reservation form.
- **Payment Page** – Displays booking summary and allows secure payment.
- **Feedback Form** – Allows users to submit ratings and comments.

Administrator Interface

- **Login Page** – Administrator authentication.
- **User Management** – Table view with update/edit capabilities.
- **Carpool Review Page** – View carpool request details, approve/reject.
- **Booking Update Module** – Edit booking data, view logs.
- **Feedback Viewer** – Sorted user feedback with filtering.

System Administrator Interface

- **Dashboard** – Summary of system operations and alerts.
- **Status Update Panel** – Allows editing and notifying booking/payment statuses.
- **Parking Slot Viewer** – Displays current availability and controls updates.
- **Cost Calculator Tool** – Display of calculated booking cost.
- **Payment Verification** – Secure connection to payment gateway and transaction log.
- **Invoice Notification System** – Sends and views notification records.
- **System Log Monitor** – Filterable logs by date, status, or module.

System Monitor Interface

- **Live Feed Dashboard** – Monitors real-time data from parking zones.
- **Anomaly Detection Alerts** – Notification section for faults or full lots.
- **Database Update Panel** – Displays most recent updates to slot data.

3.4.1.3 Hardware Interface

The Campus Ride-Sharing Platform will operate on standard user devices such as desktops, laptops, and smartphones with internet access and modern browsers. The system's backend will be hosted on a secure server infrastructure with at least an 8-core CPU, 16 GB RAM, SSD storage, and a stable network interface to support real-time processing and database access. For parking management, the system will integrate with IoT-based parking sensors installed at physical parking zones, capable of detecting vehicle presence using ultrasonic or infrared technology. These sensors will transmit real-time data to the system monitor via Wi-Fi or LTE protocols. Additionally, the system will interface with external notification services such as email and SMS gateways for delivering confirmations and alerts to users and administrators.

3.4.1.4 Software Interface

Table 3.4.1.4 **s**hows the software interfaces through which the users can make use of the platform. These interfaces facilitate the communication between the user interface, backend systems, authentication services, and third-party tools like payment gateways and notification services. They make sure everything is interacting appropriately across different parts.

Interface Name	Description
Web Frontend	The system provides a responsive web-based user interface built with HTML5, CSS3, JavaScript, and frameworks like Bootstrap or React.
Backend Server	Handles logic and data processing. Built on PHP, Node.js, or Python frameworks. Communicates with the database and API layers.
MMU Authentication System	Integrates with MMU's Single Sign-On (SSO) or OAuth2 authentication system to verify user credentials securely.
Parking Management API	Interfaces with the real-time parking availability system to fetch data and confirm reservations.
RESTful API Layer	All actions such as ride creation, joining, profile updates, and booking are handled via RESTful APIs using JSON over HTTPS.
Payment Gateway API	Supports secure transactions via FPX, Stripe, or other PCI-compliant APIs. Processes parking and ride payments.
Notification Service	Connects to cloud messaging services or SMTP for email/SMS alerts about booking confirmations, ride status, or feedback requests.
Validation & Error Handling	Both client-side (JavaScript) and server-side (backend scripts) validation ensure clean, accurate data input and error messages for user guidance.

Table 3.4.1.4: Software Interface Requirements for User

3.4.1.5 Communication Interface

Table 3.4.1.5 defines the communication interfaces that handle the data transfer between the user and internal/external systems. These comprise secure protocols, API communications, session management, and messaging services for facilitating real-time updates, confirmations, and notifications necessary for an error-free user experience.

Interface Name	Description
HTTPS Protocol	All user-to-server communications use HTTPS encryption to protect sensitive data like credentials and payment info.
MMU Authentication API	Facilitates login and logout requests through a secure OAuth2 or SAML-based Single Sign-On (SSO) service.
RESTful Communication API	User actions are sent via RESTful API endpoints using JSON. These include login, profile updates, bookings, and feedback submissions.
SMTP/Email API	The platform communicates via email for confirmations, feedback requests, and announcements using an SMTP or email API service.
Cloud Messaging (e.g., Firebase)	Enables real-time notifications (e.g., booking status, chat requests) to be sent directly to user devices or browsers.
Payment Gateway Channel	Secure payment data is sent to a third-party provider, and responses (success/failure) are received and processed immediately.
Session Tokens/Cookies	User sessions are managed using encrypted tokens or cookies to track activity securely during active periods.
Error & Status Feedback Protocol	In case of failed communication (e.g., timeouts, denied requests), the system returns detailed status codes and messages for user and debugging.

Table 3.4.1.5: Communication Interface Requirements for User

3.5 Logical Database Requirements

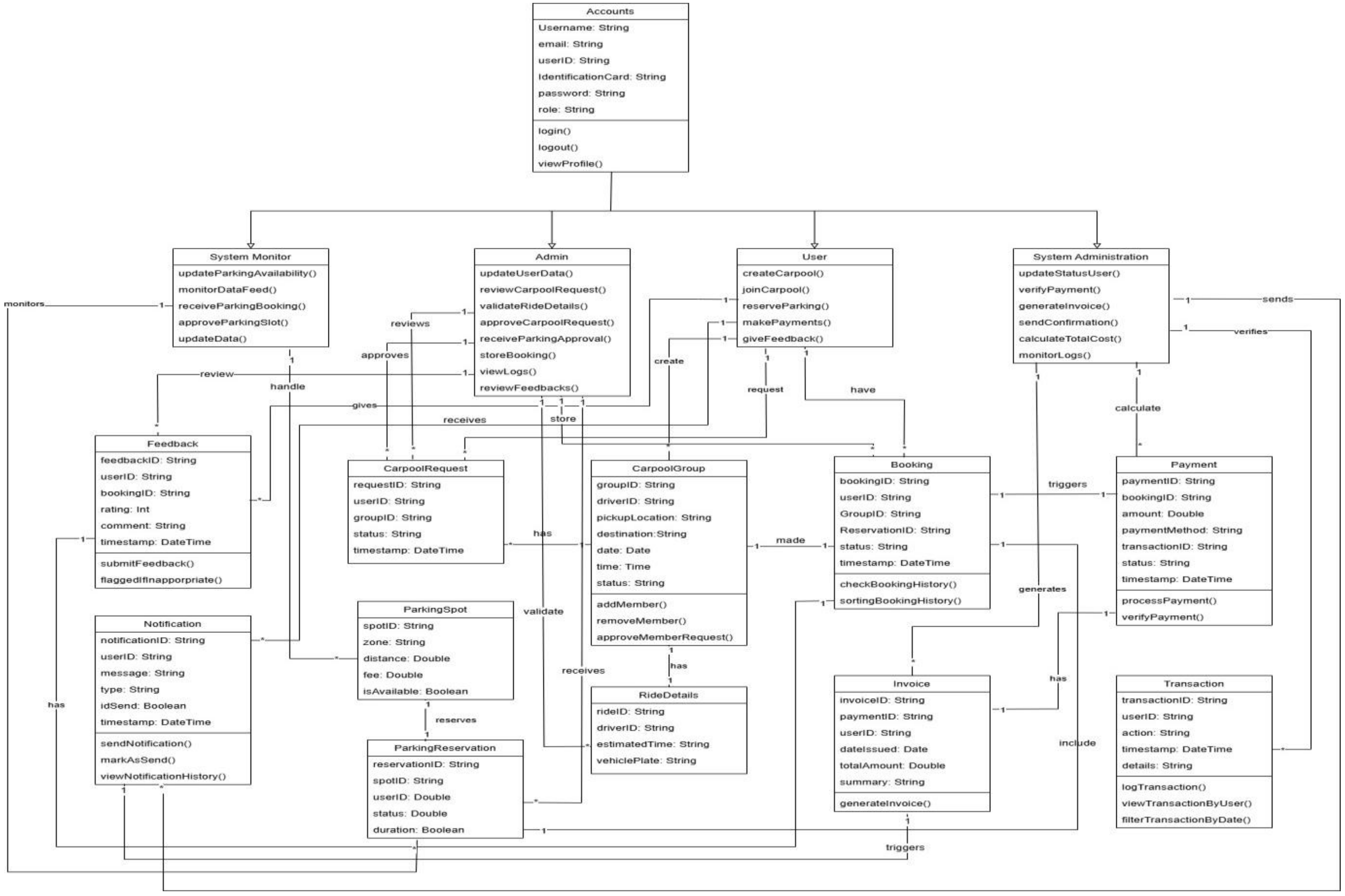


Figure 3.5: Logical Database Requirements (Class Diagram)

The UML class diagram for the Campus Ride-Sharing Platform With Parking System Integration models the core entities and relationships required for a seamless ride-sharing experience with mandatory parking reservations. At the foundation lies the abstract class UserAccount, which is inherited by four concrete user roles: User, Administrator, SystemAdmin, and SystemMonitor. Each User can create or join multiple CarpoolGroups, submit CarpoolRequests, and make Bookings for rides. Every Booking is tightly coupled with a ParkingReservation and a Payment, forming a composition relationship since these entities cannot exist independently once a booking is made. The ParkingReservation is linked to a ParkingSpot, ensuring that a specific location is secured.

Payments are processed through the Payment class and result in the generation of an Invoice. Users can also provide Feedback after each completed ride, and each Booking is associated with one such feedback entry. Additionally, Notifications are sent to users regarding system updates, carpool approvals, and reminders, reflecting a one-to-many association between UserAccount and Notification. Administrator manages approvals for carpool requests and parking reservations, while the SystemAdmin oversees platform-level verifications and payment audits. The SystemMonitor continuously observes Transactions and user activities to ensure transparency and integrity. Dependencies are modeled between classes like Administrator and CarpoolRequest, and between Booking and CarpoolGroup, signifying temporary usage without strong ownership.

3.6 Design Constraints

3.6.1 User Interface Constraints

- The system should be designed in such a way that focuses on the user's needs and preferences first. It should be easily accessible by all users including users with disabilities.
- The system should have consistent design elements like colors, typography and layout helps users to navigate properly and also enhances the system authority.
- The system should focus on delivering real value to users by making tasks easier, more efficient, or more enjoyable. The design's advantages ought to be obvious, significant, and consistent with user expectations.

3.6.2 Hardware Constraints

The system should operate on general-purpose hardware devices without requiring specialized or high-end infrastructure. The following are the hardware constraints:

1. Client-Side Devices

- The system should be available on typical user devices such as:
    - Smartphones and Tablets (iOS and Android)
    - Laptops and Desktops (Windows, macOS, or Linux)
- They ought to support a modern web browser and constant internet connectivity.
- Minimum device requirements:
    - 2 GB RAM
    - Dual-core processor
    - 720p screen resolution

2. Server Infrastructure

- The backend infrastructure will be run on a centralized server or cloud environment with a minimum:
    - Quad-core CPU (2.4 GHz and above)
    - 16 GB RAM
    - 100 GB SSD storage
    - Reliable internet connection with 1 Gbps bandwidth

- Secure access through HTTPS is required of the server and should provide continuous uptime for 24/7 availability.

3. Administrative Terminals

- Admins and system monitors will connect to the system using standard office desktop or laptop computers.
- Recommended specs:
  - Minimum 4 GB RAM
  - Full HD (1920×1080) screen
  - Wired or wireless internet connection

3.6.3 Software Constraints

The system must follow specific software-specific limitations and dependencies for the sake of compatibility, maintainability, and security:

- Operating System Compatibility: The server environment must be able to host Linux (Ubuntu 20.04 or later) or Windows Server 2019/2022 for backend deployment.
- Database System: The platform will utilize PostgreSQL or MySQL as the relational database system. Limitations include schema compatibility, ACID compliance, and secure user authentication support.
- Browser Compatibility: Front-end must be compatible with the latest major versions of popular browsers (Chrome, Firefox, Safari, Edge) and mobile-access responsive design.
- Third-party Integrations: Integrations to MMU's Central Authentication System, Parking Management APIs, and cloud messaging services must follow their respective SDKs, endpoints, and security protocols.
- Development Frameworks:
  - I. Backend: Django or Express.js (final stack dependent).
  - II. Frontend: React.js or Vue.js.
  - III. All frameworks and libraries must be properly maintained and must use the MIT, Apache 2.0, or GPL licenses.
- Security Standards: The system must use OWASP Top 10 security standards to prevent SQL injection, XSS, and CSRF attacks.

3.6.4 Communication Constraints

The system must enable secure connection between users and the server by implementing HTTPS protocols. Internally, RESTful APIs with JSON formatting will be used to standardize communication across components such as System Administrator, Monitor, and Payment Gateway. Real-time status updates must be transmitted with a latency of no more than one second. In the event of message delivery problems (e.g., email or notification difficulties), the system must contain a retry mechanism to ensure vital messages are received.

3.6.5 Data Management Constraints

The system will use a centralized relational database, such as MySQL or PostgreSQL, with ACID compliance to ensure data consistency. All system records, including bookings, payments, and logs, must be time stamped and kept safe. To avoid data loss, automated database backups will be performed on a daily basis. Access to sensitive data will be restricted using role-based access control (RBAC), and all communications and failures will be recorded in a separate module for monitoring and debugging.

3.6.6 Memory Constraints

Memory limitations describe limits relating to how much data can be stored or processed by a system/user via their runtime. Memory limitations will help manage the performance of devices via student and staff use.

- The client-side application (browser) should run efficiently utilizing only 512 MB of available system memory to support low-end student laptops or mobile devices.

- Caching commonly used information (such as user profile, carpool list, parking slot status) on the frontend, using the browser's local/session storage, should limit caching of any single item to 10 MB to maintain storage within each browser's memory limitations.

- Backend operations, for each user (such as login session, form submission, booking update) must be memory allocated no more than 256 MB for each request on the server every user consumes in a request must stay within 256 MB of memory allocation.

- For any uploaded files (for example, profile picture or documents if we allow this in a future iteration) should have a restriction in size to 5 MB each file.

3.6.7 Operational Constraints

Operational constraints establish the parameters in which the user side of the platform must be able to operate effectively and reliably.

- The system must be available via modern web browsers (Chrome v12.0+, Firefox v11.5+, Safari v17+, Edge v12.0+), with JavaScript enabled.
- A user must have internet access (minimum 1 Mbps) at the time of the interaction for successful interaction with real-time modules, including parking availability and chat.
- The system must be available 24 hours a day, 7 days a week, with reception windows (if available) in place, and must notify users through email or on-site message at a minimum of 24 hours before any closure.
- The system must allow for multiple users to log in from different devices at the same time, but restrict duplicate actions (e.g. booking) from multiple different sessions of the same user account to avoid conflict.
- Sessions will timeout after 15 minutes of inactivity, but all user accounts must receive a warning before they timeout and they must re-authenticate to access user accounts, for security reasons.

3.7 Software System Attributes

3.7.1 Reliability

The system shall operate consistently with the uptime of 99.9%, and perform its intended functions under defined conditions without failure. It should be able to seamlessly recover from unforeseen problems and manage several user requests without crashing. To guarantee system dependability, regular testing—including unit, integration, and system testing—must be carried out. If something goes wrong, the system should record it and alert the administrator so that it may be fixed right away.

3.7.2 Availability

The system will be accessible around-the-clock with the exception of planned maintenance times and the system must support failover clustering to ensure 99.9% availability. It should be built to reduce downtime and guarantee consumers' constant access. The system must preserve user data integrity and deliver the proper messages in the event of a server or network failure. Implementing system monitoring tools will guarantee high availability and prompt problem identification.

3.7.3 Security

The system prioritizes security by utilizing MMU's centralized authentication for secure and role-based logins and access control to restrict functions based on roles. Data transmissions are protected by HTTPS, and sensitive data are securely stored using encryption. Input validation helps to avoid attacks such as SQL injection, XSS, and CSRF. The platform is also integrated with MMU's digital ID verification systems for user authentication. Activity logs and real-time alerts enable monitoring and quick response to suspicious activities, ensuring compliance with data protection regulations like PDPA.

3.7.4 Maintainability

The system will be created with maintainability in mind, ensuring that future changes, enhancements, and bug repairs may be completed effectively. It will employ a modular architecture in which each component (such as the booking system, notification service, payment gateway, and user administration) is loosely connected and well-documented. This enables developers to isolate and resolve bugs without disrupting other modules.

Consistent naming conventions, extensive inline documentation, and devotion to best procedures (e.g., SOLID principles) shall be observed across the project. A Git will be used for version control, with changelog.md tracking all changes will be used to utilized for tracking source code modifications and enable rollbacks to stable versions as needed. Logs will be centralized and available to administrators, allowing for more effective troubleshooting and performance monitoring. Configuration parameters (such as database credentials and API keys) will be saved independently from the code to allow for easy updates without requiring code modifications. Maintenance processes and apprised schedules will be detailed for administrators.

3.7.5 Portability

The system should be able to run across platforms and environments with little changes. The system should operate on any OS (windows – linux) that they choose and a variety of web browsers (chrome, firefox, edge, etc.). The application should have the ability to transition to a different platform in the future (mobile, cloud, etc). Documentation and modular code should allow for easy migration and reconfiguration to facilitate reuse and lessen the effort to "move" the system to a new environment.

3.8 Supporting Information

3.8.1 Campus Ride-Sharing Platform Survey form

As part of the data collection process for the Campus Ride-Sharing Platform with Parking System Integration project, a survey consisting of eight questions was distributed to university students to better understand their commuting habits. A total of **20 respondents** participated in the survey.

Figure 3.8.1.1 shows one of the key questions asked was, *"How often do you drive to campus?"* The responses showed a diverse range of driving frequencies. Notably, **6 out of 20 respondents (30%) drive to campus daily**, while **5 respondents (25%) drive a few times a week**. Additionally, **4 respondents (20%) drive once a week**, whereas **5 respondents (25%) rarely or never drive to campus**. These findings suggest that a significant portion of the campus population does commute by car regularly, highlighting the relevance and potential impact of a ride-sharing platform integrated with parking management, particularly for those who travel frequently and may benefit from more organized and efficient parking and carpooling options.

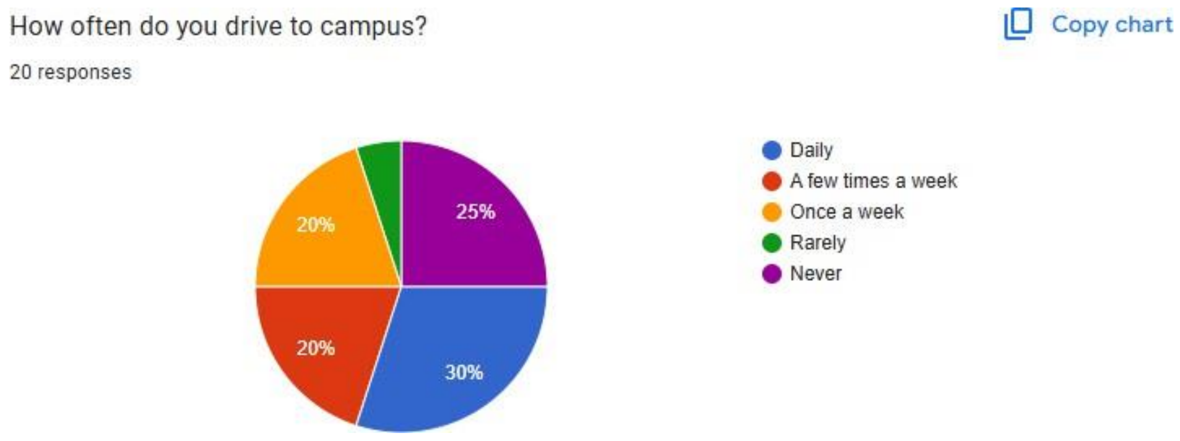


Figure 3.8.1.1: Result of How Often Do You Drive to Campus Survey

Figure 3.8.1.2 shows the current commuting methods of 20 respondents. **45%** use **personal vehicles**, making it the most common option. **Public transportation** is used by **20%**, followed by **ride-hailing apps** like Grab (**15%**). **Carpooling** and **walking/biking** are the least common, each with **10%**. These results highlight a strong dependence on private vehicles, indicating the potential benefit of a campus ride-sharing and parking system.

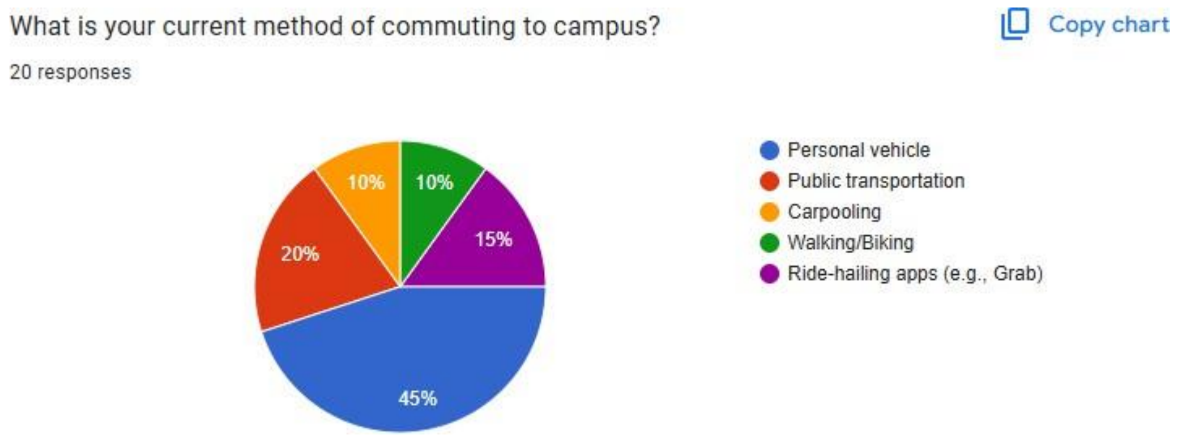


Figure 3.8.1.2: Result of What is Your Current Method of Commuting to Campus Survey

Figure 3.1.8.3 illustrates user satisfaction with current campus parking availability, based on a scale of 1 (very satisfied) to 5 (very dissatisfied). Out of 20 respondents, **35% rated 5**, showing strong dissatisfaction. **35% rated 3** as neutral response, **15% rated 4** indicating moderate dissatisfaction, while only **15% rated 1 or 2**, reflecting moderate satisfaction. These results suggest that most users are unhappy with the current parking situation, supporting the need for a better-managed parking and ride-sharing system.

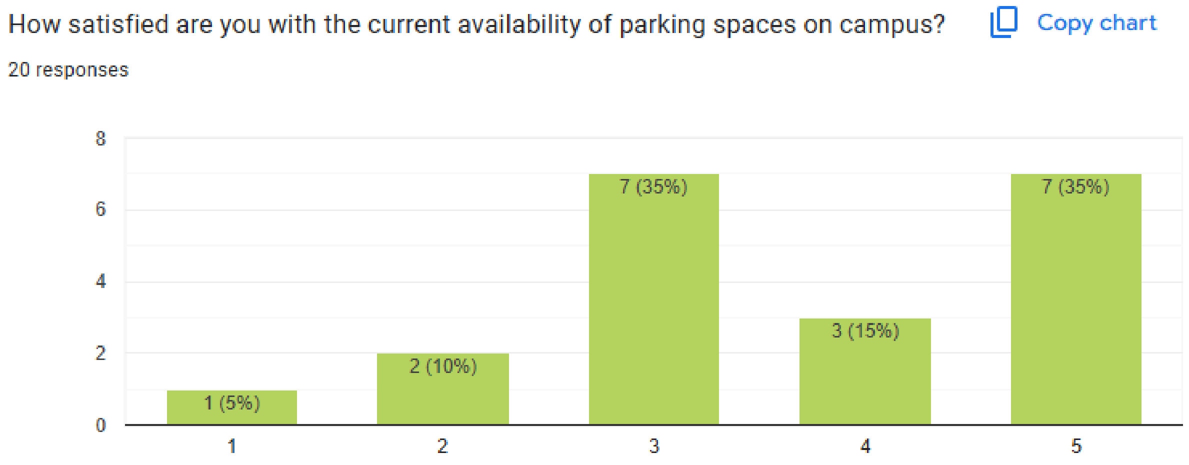
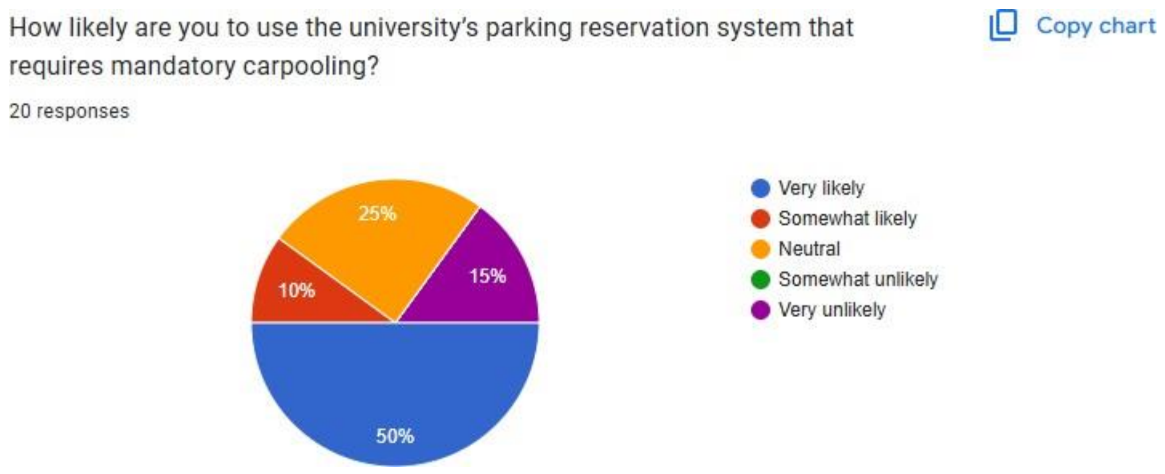


Figure 3.8.1.3: Result of How Satisfied Are You With The Current Availability of Parking Spaces in Campus Survey

Figure 3.8.1.4 illustrates the respondents' likelihood of using the university's parking reservation system that requires mandatory carpooling. Based on the responses from 20 individuals, **50%** indicated they are very likely to use the system, reflecting strong interest in the proposed solution. Meanwhile, **25%** of respondents remained neutral, and **10%** reported being somewhat likely to use it. On the other hand, **15%** expressed that they are very unlikely to adopt the system. Overall, the data suggests a generally positive outlook, with the majority of participants showing a willingness to engage with a carpool-based parking reservation approach.



How likely are you to use the university's parking reservation system that requires mandatory carpooling?

20 responses

Figure 3.8.1.4: Result of How Likely Are You to Use The University's Parking Reservation System That Requires Mandatory Carpooling Survey

Figure 3.8.1.5 illustrates the incentives that would motivate the respondents to carpool. In 20 people's responses, "Saving on parking fees" and "Parking spots available only for carpools" were the strongest incentives with 14 respondents (70%) and 13 respondents (65%) respectively. Followed closely by 13 respondents (65%), the "Convenience of ride-sharing" was an incentive. "Environmental concerns" and "Social interaction" were also important drivers, both of which were captured by 12 participants (60%). As a whole, the findings suggest that financial incentives and specialist infrastructure, with convenience and social/environmental factors, are essential drivers for the take-up of carpooling amongst the study group.

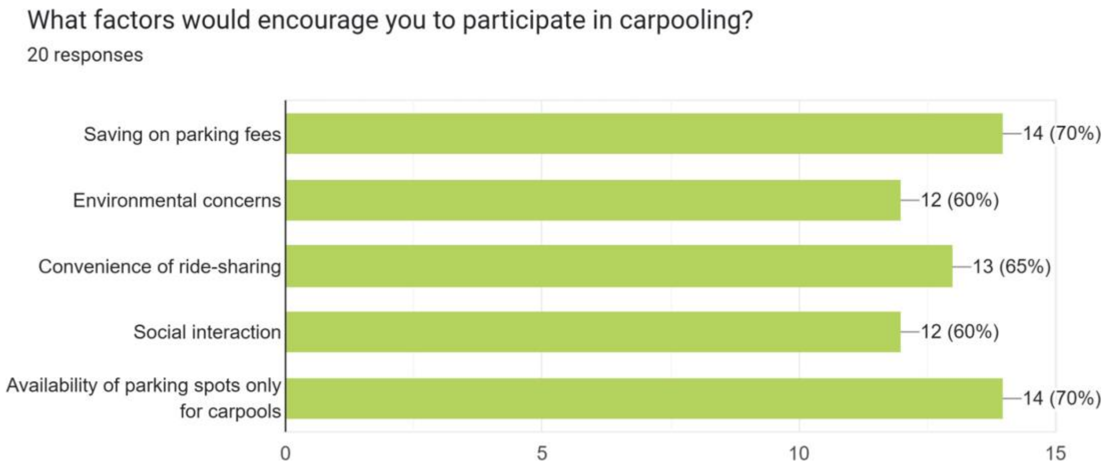


Figure 3.8.1.5: Result of What factors would encourage you to participate in carpooling

Figure 3.8.1.6 illustrates the opinions of the respondents in regard to the need to form or join a carpool group before being allowed to reserve a parking spot. Out of 20 answers collected, there was a wide variety of opinions, with the majority of individual opinions held by only one of the respondents, representing 5% of all respondents. The most frequently observed individual opinion was "Mmu student" and was recorded by 2 respondents (10%). The other miscellaneous 5% of the responses were "Reduces congestion.", "Good initiative", "Have fees", "I do think that it could help in optimizing the parking space", "Must be mmu students", "Sounds fair", "Very good", and one that was an opinion concerning promoting sustainability and utilization of parking space to its best. The diversity and scattered nature of the above answers indicate that there is no common opinion on making the compulsory carpooling condition mandatory for booking parking space.

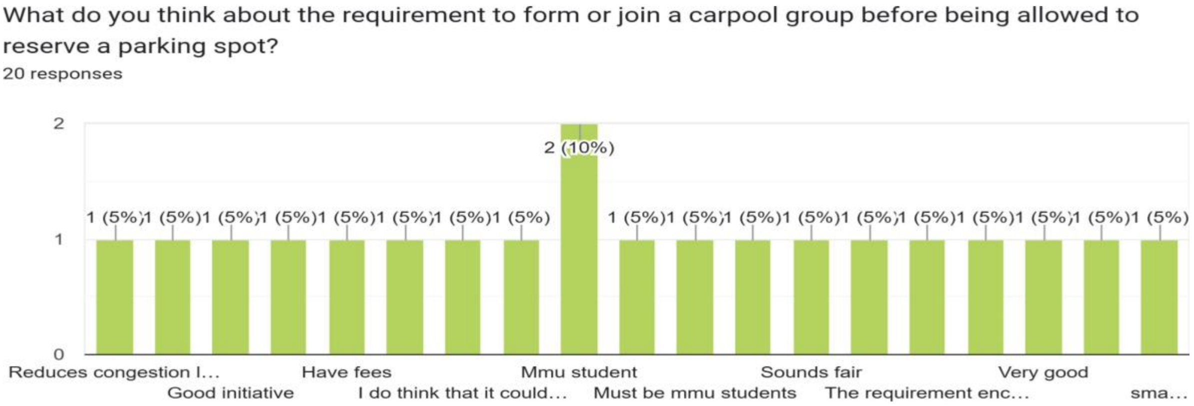
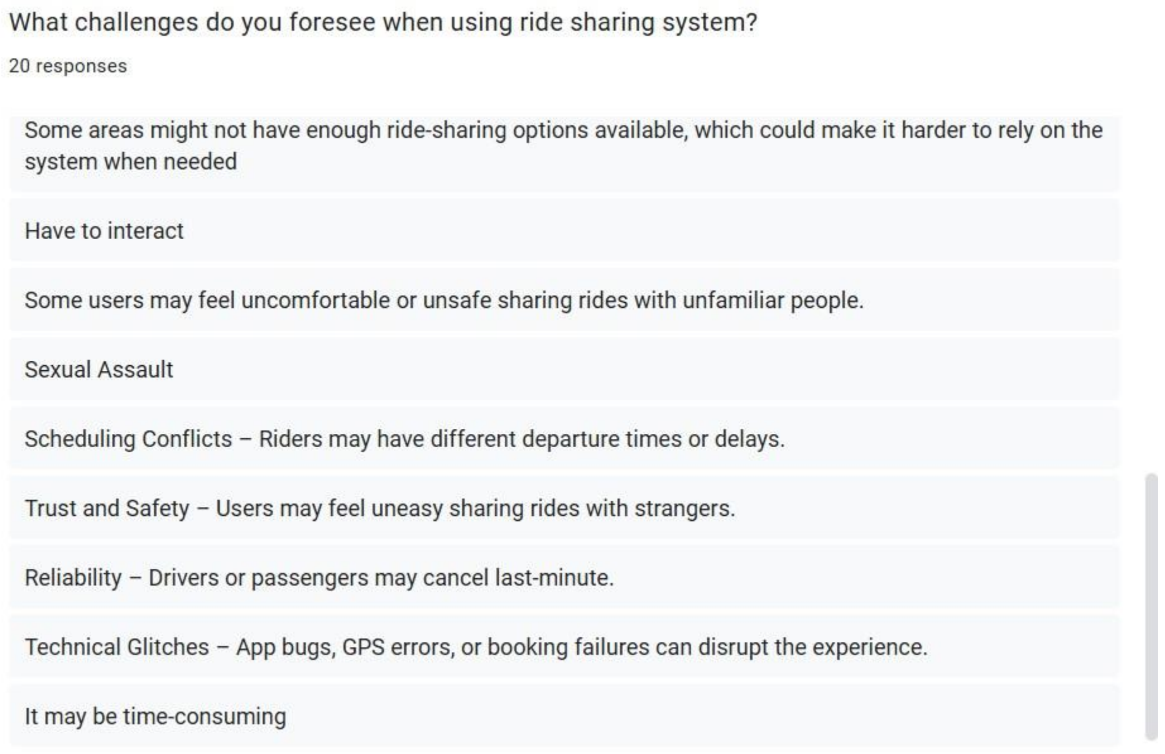


Figure 3.8.1.6: Result of What do you think about the requirement to form or join a carpool group before being allowed to reserved parking spot

Figure 3.8.1.7 illustrates the issues respondents anticipate when having to use a ride-sharing system. Out of 20 responses, a number of potential issues were enumerated, running the gamut from logistical to interpersonal, safety, and technical concerns. Respondents suggested some locations would lack sufficient choices for ride-sharing, which could compromise dependence upon the system when necessary, and also mentioned the necessity of coping with others. Safety-related issues were brought up as significant concerns, where users may feel uneasy or insecure with strangers and the threat of "Sexual Assault" was even brought up in so many words. Logistical concerns included "Scheduling Conflicts," where the riders might have different pickup times or be stuck waiting, and general "Reliability" issues like cancellations of drivers or passengers at the last minute. Furthermore, "Technical Glitches" like app bugs, GPS failures, or booking errors were also expected as disruptions to the experience, and the system was viewed as potentially time-consuming. Overall, the responses indicate that users anticipate a broad range of issues, from functional operation difficulties to critical safety and comfort issues, when considering the use of a ride-sharing system



What challenges do you foresee when using ride sharing system?

20 responses

Some areas might not have enough ride-sharing options available, which could make it harder to rely on the system when needed

Have to interact

Some users may feel uncomfortable or unsafe sharing rides with unfamiliar people.

Sexual Assault

Scheduling Conflicts – Riders may have different departure times or delays.

Trust and Safety – Users may feel uneasy sharing rides with strangers.

Reliability – Drivers or passengers may cancel last-minute.

Technical Glitches – App bugs, GPS errors, or booking failures can disrupt the experience.

It may be time-consuming

Figure 3.8.1.7: Result of What challenges do you foresee when using ride sharing system

Figure 3.8.1.9 presents the features that would significantly increase respondents' likelihood of using a parking reservation system with ride-sharing, as stated by 20 responses. The top feature was "Real-time availability updates," chosen by 16 respondents (80%), followed by an "Easy-to-use mobile app," the "Ability to choose carpool riders," and "Integration with class schedules" as equally important, each chosen by 14 respondents (70%). In addition, "Guaranteed parking spots for carpoolers" was wished for by 13 individuals (65%), and "Incentives (e.g., discount, reward points)" were wished for by 11 individuals (55%). In total, the data indicate that transparency, simplicity of use, freedom over carpooling plans, and assured parking benefits are core to motivating use of such a system.

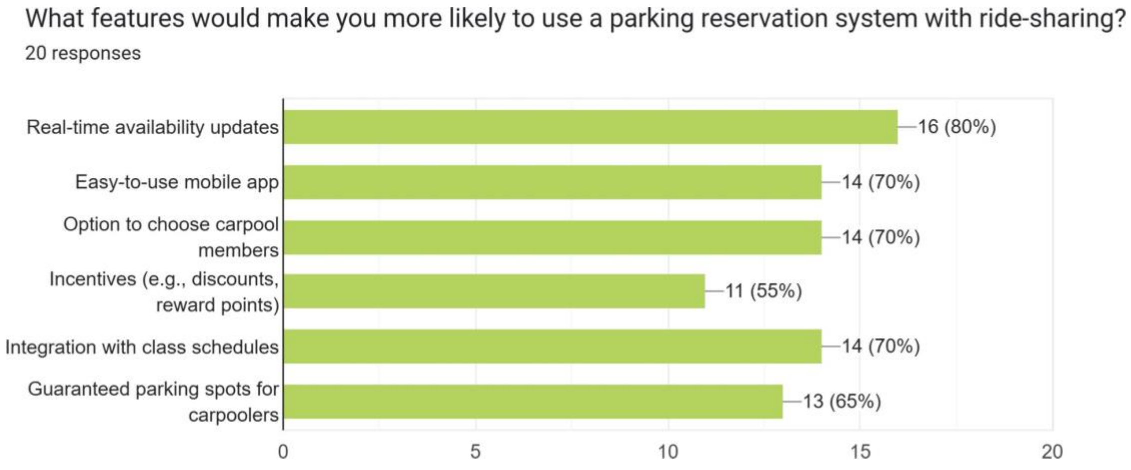


Figure 3.8.1.8: Result of What features would make you more likely to use a parking reservation system with ride-sharing

3.8.2 Brainstorming

Brainstorming meetings for the Campus Ride-Sharing Platform were held with a diverse group of participants, including students (possible users), system administrators, administrator and system monitoring staff. These sessions sought to identify major areas in current campus transportation, desirable features, and potential improvements towards parking and carpool services.

During brainstorming, participants highlighted:

- The need for a real-time parking slot display system.
- A simple way for users to create or join carpool groups.
- Automated cost calculation for rides based on duration and group size.
- Instant notifications and invoice delivery post-booking or payment.
- Administrator-side monitoring tools for logs, availability, and approvals.

Benefits & Limitation

Benefits:

- **Better Parking Use**

  Live updates g the amount of available space bald lot limits dangerous driving, angry frustration, and overuse of space.

- **Cost and Resource Sharing**

  Carpooling not only reduces costs to the individual, it services the greater good of our environment.

- **Simplified Communication**

  Instant notifications and invoices inform users of parked cars and receiver hybrid business communication.

- **Monitor Tools for Admins**

  Monitor tools for admins provide important info on the hybrid infrastructure and fixed systems, while also having the ability to clarify issues easily while enforcing app rules fairly.

- **Building an Ideal Community**

  Carpooling helps build cooperation and connection between students and staff. They're a part of your higher education system building, or a useful slant to your scheme.

Limitation:

- **User Engagement**

  It is important for the system to depend on user engagement, most importantly for carpooling.

- **Scheduling Conflicts**

  Scheduling conflicts are challenges users will experience when trying to arrange ride times with those they are meeting, limiting pool viability.

- **Privacy/Safety Concerns**

  Accepting rides from unknown individuals may lead to personal safety friction or issues of comfort.

- **Technology Access**

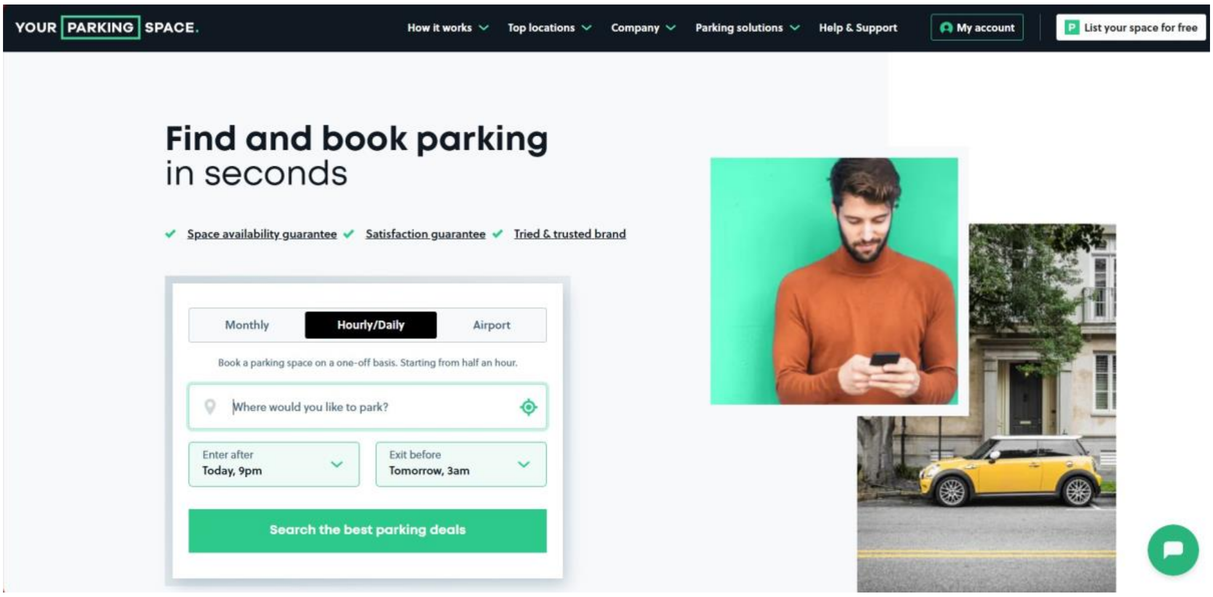
  Users may be unfamiliar with devices or applications, or may also lack most access to the web, leading to challenges on the platform.

- **Technical Issues/Errors**

  Technical issues may occur at various times affecting notifications, payments, or bookings, which only leads to distrust.

3.8.3 Existing System

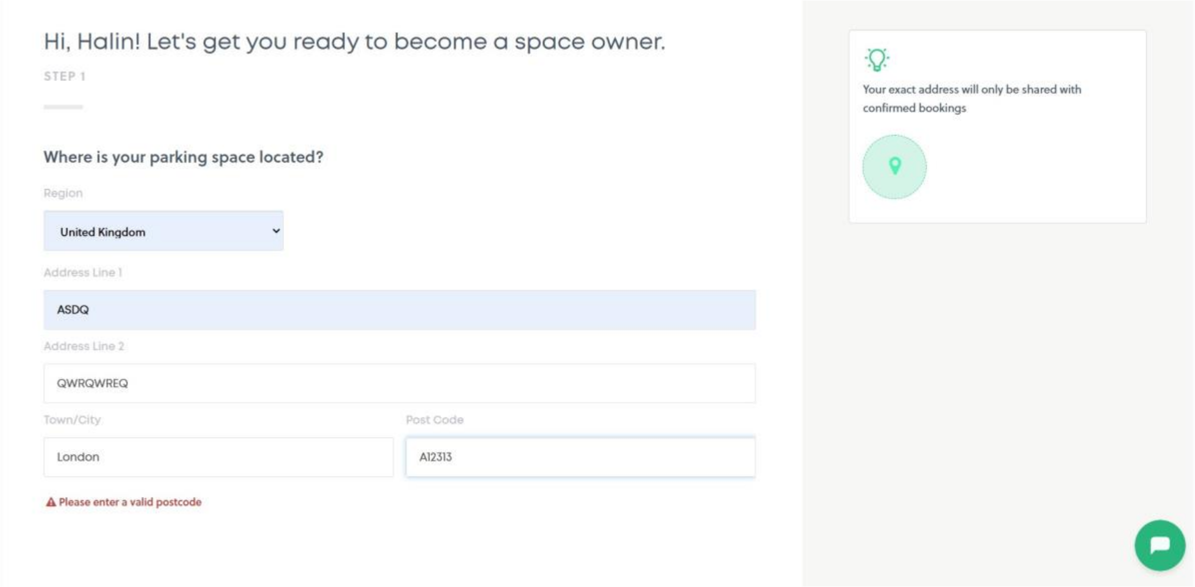
The dashboard in the YourParkingSpace system provides a broad view for users operating parking bookings across the UK. With this dashboard, users can view real-time availability of parking spaces, upcoming bookings, and parking history in selected timeframes. The interface also gives access to user accounts, payment summaries, as well as personalized recommendations from favorite parking spots. A sidebar menu allows drivers to transition smoothly from the whole process—searching and reserving a space, managing vehicle data, and looking at history—to a complete end-to-end parking experience.


YOUR PARKING SPACE.
How it works
Top locations
Company
Parking solutions
Help & Support
My account
List your space for free
Find and book parking in seconds
Space availability guarantee
Satisfaction guarantee
Tried & trusted brand
Monthly
Hourly/Daily
Airport
Book a parking space on a one-off basis. Starting from half an hour.
Where would you like to park?
Search the best parking deals

Enter after: Today, 9pm

Exit before: Tomorrow, 3am

Figure 3.8.3.1: User Dashboard



Hi, Halin! Let's get you ready to become a space owner.

STEP 1

Where is your parking space located?

Region: United Kingdom

Address Line 1: ASDQ

Address Line 2: QWRQWREQ

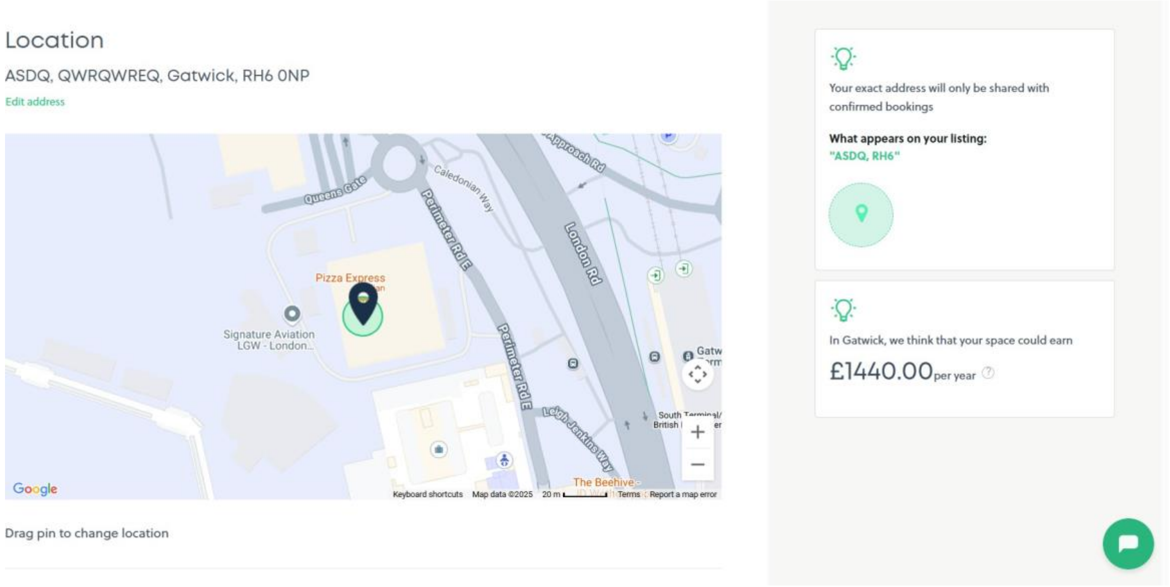
Town/City: London

Post Code: A12313

⚠ Please enter a valid postcode

Your exact address will only be shared with confirmed bookings

Figure 3.8.3.2: User Registration Form



Location

ASDQ, QWRQWREQ, Gatwick, RH6 0NP

Edit address

Drag pin to change location

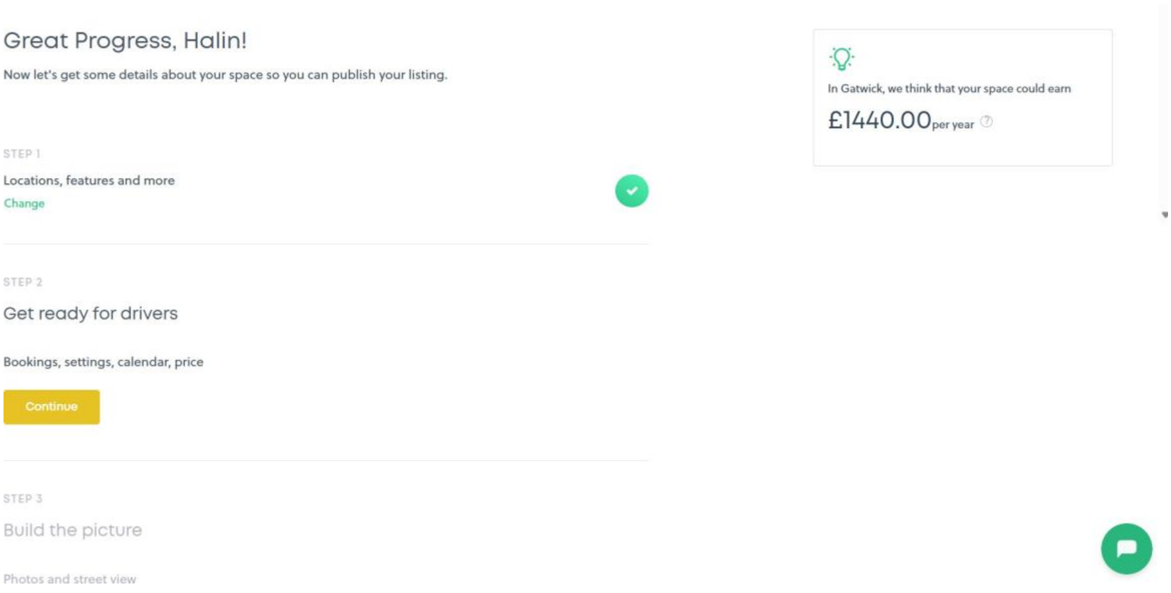
Your exact address will only be shared with confirmed bookings

What appears on your listing:
"ASDQ, RH6"

In Gatwick, we think that your space could earn

£1440.00per year

Figure 3.8.3.3: Parking Cost Calculator



Great Progress, Halin!

Now let's get some details about your space so you can publish your listing.

STEP 1

Locations, features and more ✓

Change

STEP 2

Get ready for drivers

Bookings, settings, calendar, price

Continue

STEP 3

Build the picture

Photos and street view

In Gatwick, we think that your space could earn

£1440.00per year

Figure 3.8.3.4: Notifications Steps

3.8.4 Validation Session

Session ID	Date and Time	Technique	Section Reviewed	Participant & Role	No. of Defect
VS-01	2/6/2025 6.00 PM	Inspection	3.2.2 Admin Performance Requirements	Lau Kaixuan, Alsaman Leen, Thong Yun Peng	4
VS-02	5/6/2025 8.00 PM	Walkthrough	3.4.1.4 User Interface Requirements	Lau Kaixuan, Alsaman Leen, Thong Yun Peng, Monish	3
VS-03	8/6/2025 8.00 PM	Prototyping	3.7.2 System Availability	Lau Kaixuan, Thong Yun Peng, Monish	2
VS-04	12/6/2025 6.00 PM	Risk Analysis	3.6.5 Data Management Constraints	Lau Kaixuan, Alsaman Leen, Monish	5
VS-05	15/6/2025 8.00 PM	Inspection	3.10 Feedback Prioritization	Lau Kaixuan, Alsaman Leen, Thong Yun Peng, Monish	3

3.8.4Defects Summary

A. Content Defect

Req ID	Validation and Defect Description	Detected By	Comment/Suggested Fix	Session ID	Severity (1–5)
REQ-001	Missing performance metrics in Admin section (3.2.2)	Lau Kaixuan	Replace with actual performance requirements	**VS-01**	5
REQ-004	Admin Module Requirement 6 ("securely store in the database") lacks encryption details (Section 3.9)	Lau Kaixuan	Specify encryption standards (e.g., AES-256)	**VS-03**	3
REQ-007	User Interface Requirement: Missing accessibility standards (WCAG compliance) (Section 3.4.1.2)	Lau Kaixuan	Add: "Interface must comply with WCAG 2.1 AA."	**VS-02**	3
REQ-010	Missing carpool group size limit (Section 1.3.2.1.3 and 3.1.1.3)	MONISH	Add: "Maximum 4 users per group for carpools."	**VS-04**	3
REQ-013	Undefined cost formula (Section 1.3.2.3.4 and 3.1.3.4)	MONISH	Add: how actually the ride cost is calculated where both ride and parking fee combined	**VS-05**	4
REQ-016	Browser compatibility still states "modern browsers" instead of specific versions (Section 3.6.7)	Lau Kaixuan	Replace with: "Chrome v12.0+, Firefox v11.5+, Safari v17+, Edge v12.0+"	**VS-01**	2
REQ-019	Feedback prioritization (High/Medium/Low tags) is missing (Section 3.10)	Lau Kaixuan	Add: "Feedback must be tagged with priority levels."	**VS-05**	3
REQ-022	Vague reliability metrics	Lau Kaixuan	Add quantifiable metrics (e.g., "99.9% uptime").	**VS-05**	4

B. Documentation Defect

Page No.	Validation and Defect Description	Detected By	Comment/Suggested Fix	Session ID	Severity (1–5)
Throughout	Inconsistent terms: Standardize "System Administration" (Sections 1.3.2.3, 3.1.3)	MONISH	Standardize term throughout documentation.	VS-09	2
72	Ambiguous term "basic accessibility standards" in UI requirements (Section 3.3)	Lau Kaixuan	Replace with WCAG 2.1 AA standards.	VS-10	2
83	"Accessible around-the-clock" lacks failover mechanisms for 99.9% uptime (Section 3.7.2)	Lau Kaixuan	Add: "System must support failover clustering."	VS-11	3
84	Maintainability description lacks version control references (Git workflow) (Section 3.7.4)	Lau Kaixuan	Add: "Git will be used for version control."	VS-12	4

C. Agreement Defect

Req ID	Validation Description/Stakeholder Concern	Mismatch	Detected By	Session ID	Severity (1–5)
REQ-025	**Voice Booking Missing – Stakeholders demand voice commands for reservations**	Missing voice command functionality	Thong Yun Peng	VS-02	4
REQ-026	**Mandatory Carpool Before Parking – Users want flexibility for short-term parking**	Carpooling requirement vs. stakeholder needs	Thong Yun Peng	VS-04	4
REQ-027	**Flexibility of Carpooling – Mandates carpooling for all users**	Mandatory carpooling vs. stakeholder flexibility	Thong Yun Peng	VS-04	4
REQ-028	Stakeholders require biometric login (fingerprint/facial recognition), but SRS only specifies password-based auth	Security feature mismatch	Lau Kaixuan	VS-01	4
REQ-029	Feedback submission lacks priority tagging (High/Medium/Low)	Functional requirement misalignment	Lau Kaixuan	VS-05	3
REQ-030	Real-time parking	Core	Lau Kaixuan	VS-03	4

	updates every 30 seconds (PDF Page 102) are missing	functionality vs. documented requirements			
REQ-031	Admin module performance metrics (e.g., "update user data within 2 seconds") are missing	Efficiency expectations vs. documentation	Lau Kaixuan	VS-01	4

3.8.5Conflict Analysis

Conflict ID	Conflict Description	Conflict Analysis	Stakeholders Involved	Session ID
CF-01	**Performance vs. Cost Tradeoff: Stakeholders demand 30-second parking updates and failover mechanisms, but development team cites resource constraints.**	**Interest conflict: QA/Dev prioritize reliability/speed; PO emphasizes cost control. Root cause: misaligned priorities.**	**PO, QA, Dev Team**	**VS-04**
CF-02	**Biometric Login Missing: Stakeholders expect biometric authentication, but SRS only specifies password-based access.**	**Requirement mismatch: Stakeholders prioritize modern security; developers focus on baseline compliance.**	**Security Team, PO**	**VS-01**
CF-03	**Admin Performance Metrics Missing: Stakeholders expect measurable Admin task efficiency (≤2 seconds for updates), but SRS lacks metrics.**	**Gap in stakeholder expectations vs. documentation. Admin efficiency impacts scalability.**	**Dev Team, PO**	**VS-01**

3.8.6Conflict Analysis and Resolution

Conflict ID	Conflict Resolution Strategy	Resolved (Y/N)	Outcome (If Resolved)	Justification
CF-01	**Structured negotiation facilitated by Scrum Master, including trade-off analysis and ROI demonstration.**	**Y**	**Prioritize performance: Implement 30-second parking updates in Phase 2; upgrade failover mechanisms post-MVP.**	**Long-term ROI of performance improvements justified incremental cost increases.**
CF-02	**Stakeholder compromise: Add biometric login as afuture enhancementin Appendix A.**	**Y**	**Biometric authentication will be added in Version 2.0 as an enhancement.**	**Immediate implementation would delay MVP; stakeholders agree to phased delivery.**
CF-03	**Collaborative review of Admin workflows and stakeholder feedback sessions.**	**Y**	**Add measurable Admin performance metrics (e.g., ≤ 2 seconds for user data updates)**	**Metrics ensure alignment with stakeholder efficiency expectations.**

3.8.7Change Log

Change ID	Req ID(s)	Summary of Change	Proposed By	Date
CH-01	**REQ-001, REQ-004, REQ-007**	**Replace with actual performance requirements; Specify encryption standards; Add WCAG 2.1 AA compliance**	**Lau Kaixuan**	**21/6/25**
CH-02	**REQ-025**	**Added Voice Booking as future enhancement**	**Thong Yun Peng**	**22/6/25**
CH-03	**REQ-026, REQ-027**	**Updated Reserve Parking Spot section for carpool flexibility (emergency/short-term/VIP**	**Thong Yun Peng**	**22/6/25**

		exceptions)		
CH-04	**REQ-022**	**Added quantifiable reliability metrics (99.9% uptime)**	**Lau Kaixuan**	**22/6/25**
CH-05	**REQ-010**	**Added max 4 users/group for carpools**	**Monish**	**22/6/25**
CH-06	**REQ-013**	**Updated cost calculation to include ride + parking fees**	**Monish**	**22/6/25**
CH-07	**-**	**Standardized "Admin" → "Administrator" throughout documentation**	**Monish**	**22/6/25**

3.8.8Requirements Traceability Matrix

Req ID	Requirement Description	Linked Goal(s)	Feature(s)	Use Case(s)	Traceability Score (1-4)
REQ-001	**System must respond within ≤ 2 seconds**	**G1 (User Experience)**	**F1 (Login)**	**UC-01 (Login)**	**4**
REQ-004	**Real-time parking updates every 30 seconds**	**G3 (Efficiency)**	**F5 (Parking)**	**UC-07 (Reserve Parking)**	**4**
REQ-012	**24/7 uptime with failover clustering**	**G4 (Security)**	**F8 (Failover)**	**UC-12 (System Monitoring)**	**4**
REQ-019	**Biometric login (future enhancement)**	**G5 (Modern Security)**	**F9 (Login)**	**UC-13 (Authentication)**	**3**
REQ-020	**Feedback prioritization (High/Medium/Low tags)**	**G6 (Feedback Management)**	**F10 (Feedback)**	**UC-14 (Leave Feedback)**	**4**

REQ-021	**Admin performance metrics (≤ 2 seconds for updates)**	**G7 (Admin Efficiency)**	**F11 (Admin Metrics)**	**UC-08 (Update User Data)**	**4**
REQ-022	**AES-256 encryption for database storage**	**G4 (Security)**	**F2 (Data Security)**	**UC-02 (Data Storage)**	**4**

Traceability Score	**Description**
1	Linked to only **1 artifact** (e.g., just a goal, or just a use case)
2	Linked to **2 artifacts** (e.g., goal + feature)
3	Linked to **3 artifacts**, but links may be **basic or unverified**
4	Linked to **3 artifacts** with **high confidence, correctness, and completeness** (e.g., validated relationships, clear traceability)

3.8.9 Role Requirements, Negotiation & Management

Student Name	Primary Responsibility	No. of Session Participated
Lau Kaixuan	**Context Validation, Traceability Matrix updates, GitHub Version Control, Conflict analysis, Documentation Review**	**5**
Alsaman Leen	**GitHub Version Control, Context Validation, Documentation**	**4**

	Review	
Monish	**GitHub Version Control, Cotext Validation, Documentation Review**	**3**
Thong Yun Peng	**GitHub Version Control, Context Validation, Documentation Review**	**4**

3.8.9Version Control & Configuration Summary

Repository Branch: Main

SRS.md = Working version of updated SRS

TT2L_G3_SRS.doc: Final version

Changelog.md: Record the history of the project

Commits Made by Monish: 3
Pull Requests Merged by Monish:
Change Log Entries Made by Monish:

Commits Made by Lau Kaixuan: 6
Pull Requests Merged by Lau Kaixuan: 5
Change Log Entries Made by Lau Kaixuan: 3

Commits Made by Alsaman Leen: 2
Pull Requests Merged by Alsaman Leen: 1
Change Log Entries Made by Alsaman Leen: 1

Commits Made by Thong Yun Peng: 5
Pull Requests Thong Yun Peng: 2
Change Log Entries Made By Thong Yun Peng: 2

3.9 Apportioning of Requirements

1. User Module

- **Requirement 1:** The system must authenticate users and load the dashboard within 2 seconds after successful login.
- **Requirement 2:** Users must be able to fill and submit their personal and ride-related information with confirmation received within 3 seconds.
- **Requirement 3:** Users must be able to create a carpool group, and the request should be submitted and acknowledged by the system within 4 seconds.
- **Requirement 4:** When joining a carpool group, the system should check for availability and respond within 3 seconds.
- **Requirement 5:** Users must be able to view and reserve available parking slots, and get confirmation within 3 seconds.
- **Requirement 6:** Payment processing should be completed and confirmation shown to the user within 5 seconds.
- **Requirement 7:** Feedback submitted by the user should be stored and confirmation displayed within 2 seconds.

2. Administrator Module

- **Requirement 1:** Able to update and store users' data such as name, contact information, and status.
- **Requirement 2:** Can review user-submitted carpool requests, including requester details, date, time, pickup, and drop-off locations.
- **Requirement 3:** Able to validate ride details submitted by users to ensure completeness and accuracy before approval.
- **Requirement 4:** Able to approve or reject carpool requests based on submitted ride details.
- **Requirement 5:** Able to receive and view parking approval requests linked to carpool bookings.
- **Requirement 6:** Able to update booking details and securely store them in the database and the data stored in the database must use AES-256 encryption.
- **Requirement 7:** Able to review user comments or feedback and store them for future reference.

3. System Administration

- **Requirement 1:** Able to update a user's status (booking, payment, approval) and trigger notifications within 2 seconds of initiating the action.
- **Requirement 2:** Able to retrieve and view real-time slot availability data within 1 second of request, assuming a responsive database.
- **Requirement 3:** Ensure that users see the updated list of slots within 2 seconds after availability is confirmed.
- **Requirement 4:** System must calculate the total cost of a booking (based on parameters like time, slot type, carpooling status) within 1 second of receiving inputs.
- **Requirement 5:** Payment verification must be completed within 2 seconds, including cross-checking with the payment gateway.
- **Requirement 6:** System must send booking confirmations or invoices via email or platform notifications to users within 1 second of validation.
- **Requirement 7:** Able to access system logs with filter features (by date, action, type) within 3 seconds, even under high load.

4. System Monitor

- **Requirement 1:** Must receive, process, and validate incoming parking data feeds within 1 second of input.
- **Requirement 2:** Must detect significant data changes and verify updates internally within 2 seconds.
- **Requirement 3:** Must update the Parking Database with slot availability changes within 1 second of verification.
- **Requirement 4:** Must deliver notifications for exceptional events (parking full, sensor faults) to the System Administrator within 1 second of event detection.
- **Requirement 5:** Must process concurrent data streams from up to X parking zones and update the databases within 2 seconds per zone.

3.10 Specified Requirements

This section describes the specific functional requirements and non-functional requirements for the Campus Ride-Sharing Platform with Parking System Integration. These requirements will establish the basis for the development of the system and make sure that all stakeholders have a shared, clear understanding of what the system has to do.

Functional Requirements:

1. **User Account Management**

- Users will be able to register, log in, and modify their profile using MMU credentials.
- The system will validate a user’s identity using the University's SSO system.

2. **Carpool**

- Users will be able to create a carpool group, by entering the route, time, and vehicle.
- Users will be able to search and join an existing carpool, subject to spot availability.
- Administrators must approve or reject carpool requests before they are deemed active.

3. **Parking Reservation**

- Only users belonging to an approved carpool group will be able to reserve parking spots.
- Parking slots availability must be shown in real time, and updated in real time.

4. **Payment Integration**

- The system must support online payment for all parking and carpool related fees via FPX or equivalent.
- Payment status must be verified and recorded prior to confirmation of booking.

5. **Feedback and Notifications**

- Users will be able to submit feedback and assess their ride experiences and the feedback must be tagged with priority levels for stakeholder review.
- The system will send notifications for requests, updates, and confirmations in real-time.

Non-Functional Requirements:

1. **Performance**

- The primary functionality for the system (login, booking, payment) will perform in <2–5 seconds.
- The system will be able to accommodate many users simultaneously with no reduction in service.

2. **Security**

- All data will be sent securely using HTTPS.
- Role-based Access Control (RBAC) will be provided for each user type.

3. **Usability**

- The system will be easy to use and provide a fully responsive UI for desktop and mobile devices.
- Users will be provided feedback, confirmation, and error messages where applicable during their interaction.

4. **Maintainability**

- The system will be modular and will allow for changes with minimal downtime.

5. **Portability**

- The application will run on all major browser platforms and will deploy on standard IT infrastructure that a university provides.

4 Verification

4.1 Verification Approach

A systematic verification process will be used to make sure the Campus Ride-Sharing Platform With Parking System Integration satisfies all functional and non-functional requirements. This comprises several testing tiers at various phases of development, each of which is mapped to distinct system components.

How:

Verification will be conducted using the following methods:

- **Unit Testing:** Each class (Booking, CarpoolGroup, ParkingReservation, Payment, Notification) will be individually tested to verify logic correctness, particularly with regard to relationships like composition and dependency.

- **Integration Testing:** Interactions between modules (ebooking with parking, payment with invoice, user with carpool creation) will be tested to ensure that communication between classes works as expected.

- **System Testing:** Full end-to-end scenarios, such as "Reserving a Parking Spot with Carpool Creation" or "Joining Existing Carpool and Completing a Ride," will be tested against the scenario flows.

- **Functional Testing:** All use cases will be tested, including user login, administrator approval, payment handling, and notification delivery.

- **Validation Against UML Design:** Verification will also ensure that the final implementation remains aligned with the class diagram and system design artifacts.

Who:

Verification responsibilities are divided as follows:

- **Development Team:** Responsible for unit and integration testing throughout the development process.
- **QA/Test Team:** In charge of functional and system-level testing, particularly on staging environments.
- **System Administration Team:** Will verify backend logs, monitor real-time parking data flows, and validate payment security.
- **Administrator and System Monitor Roles:** Participate in verification for request approval flows, feedback flagging, and system event logging.

When:

Verification will occur in iterative stages:

- **After Each Sprint:** Unit and integration tests will be run at the end of every sprint cycle.
- **After Major Features Are Built:** Once features like booking, payment, and parking modules are complete, full system tests will be executed.
- **Before Deployment:** Complete system verification will be performed before staging and final production deployment.

Where:

Verification activities will take place in:

- **Local Development Environments:** For unit and initial integration tests.
- **QA Testing Environment (Staging Server):** For full system and functional testing.
- **Monitoring Dashboards:** For live verification and backend performance assessment after deployment.

4.2 Verification Criteria

- **Login & Authentication:**
  The system must verify user identity through university ID and password. Unauthenticated users must be denied access.

- **Carpool Creation & Request:**
  A carpool group must not be created unless at least one other participant has been invited. Join requests must be reviewed and approved by the carpool creator or administrator within 24 hours.

- **Parking Reservation:**
  Parking reservation should only be enabled after a carpool is approved. The system must check real-time slot availability before confirming.

- **Ride-Matching Algorithm:**
  The system must assign drivers to riders within 10 seconds after carpool and parking approval.

- **Booking Confirmation:**
  The system must send a confirmation notification (email and in-app) within 5 seconds of booking approval.

- **Payment Processing:**
  Payment must be processed securely via a simulated payment gateway, with a success/failure response within 5 seconds. A transaction ID must be generated for each successful booking.

- **Response Time:**
  All user actions (e.g., searching carpools, booking ride, loading parking zones) should complete in under 3 seconds under normal server load.

- **Administrator Actions:**
  Administrator should be able to view, approve, or reject any booking or carpool request in under 5 seconds.

- **System Monitoring:**
  Parking availability must update in real time every 30 seconds, monitored and refreshed by the System Monitor module.

- **Error Handling:**
  If any component fails (ride matching or payment), an error log should be generated and a notification must be sent to System Administration within 1 minute.

- **Feedback Submission:**
  Users must be able to submit ride feedback after marking the ride as “Completed,” and the data should be saved and viewable in the system.

5 Appendices

5.1 Assumptions and Dependencies

5.1.1 Assumptions

1. **Stable Internet Connection:**
   It is assumed that users and system components have consistent internet access for real-time operations.

2. **University Authentication Integration:**
   All users (students, staff, administrator) are assumed to log in using valid MMU university credentials.

3. **Accurate Real-time Data Feeds:**
   It is assumed that the parking data feed and user booking information are accurate and updated in real time.

4. **Payment Gateway Availability:**
   The third-party payment gateway service is assumed to be operational and capable of processing transactions securely and efficiently.

5. **Email/SMS Notifications:**
   The system assumes the availability of a functioning notification system (e.g., SMTP or messaging API) to send booking confirmations or status updates.

6. **Database Integrity:**
   It is assumed that the database remains consistent, with valid data models and properly indexed tables for efficiency.

7. **Logging and Monitoring System:**
   The logging system is assumed to be operational and capable of tracking all backend actions for security and audit purpose

5.1.2 Dependencies

1. **University Authentication Server:**

   The platform depends on MMU's identity management system for login and authentication.

2. **Real-Time Parking Feed System:**

   The availability and accuracy of parking slots rely on the data provided by sensors or manual updates from system monitors.

3. **Third-party Payment Gateway:**

   The system relies on an external payment gateway (e.g., FPX, Stripe, etc.) for processing and verifying user payments.

4. **Email/SMS Notification Service:**

   Notifications (like booking confirmations, status updates) depend on third-party services or university APIs.

5. **Web Hosting & Server Infrastructure:**

   The system is dependent on reliable hosting infrastructure (e.g., AWS, university servers) for uptime and performance.

6. **Browser Compatibility:**

   Users depend on modern web browsers (Chrome, Firefox, Edge) to access the platform effectively.

7. **Database Management System:**

   The backend relies on a functioning and optimized DBMS (e.g., MySQL, PostgreSQL) to store and retrieve system data efficiently.

5.2 Acronyms and Abbreviations

The table below contains a variety of acronyms and abbreviations that are included in this Software Requirement Specification (SRS) document. The terms provided below are included to suggest meaning, promote common understanding by all stakeholders and avoid the redundancy of lengthy terms. Each term contains its full form and a short statement about its relevance to the system.

Acronym/ Abbreviation	Full Form	Description
SRS	Software Requirements Specification	A document that outlines the software system's functional and non-functional requirements.
UI	User Interface	The interface through which users interact with the system.
DB	Database	A structured collection of data stored and accessed electronically.
SSO	Single Sign-On	An authentication process that allows a user to access multiple applications with one set of login credentials.
MMU	Multimedia University	The institution for which the ride-sharing and parking platform is developed.
ID	Identification	A unique identifier used to authenticate or represent a user in the system.
FPX	Financial Process Exchange	A Malaysian online payment gateway system.
SMTP	Simple Mail Transfer Protocol	A protocol used for sending emails.
API	Application Programming Interface	A set of tools and definitions for building and interacting with software applications.

UX	User Experience	The overall experience a user has when interacting with the platform.
OS	Operating System	System software that manages hardware and software resources.

Table 5.2: Acronyms and Abbreviations