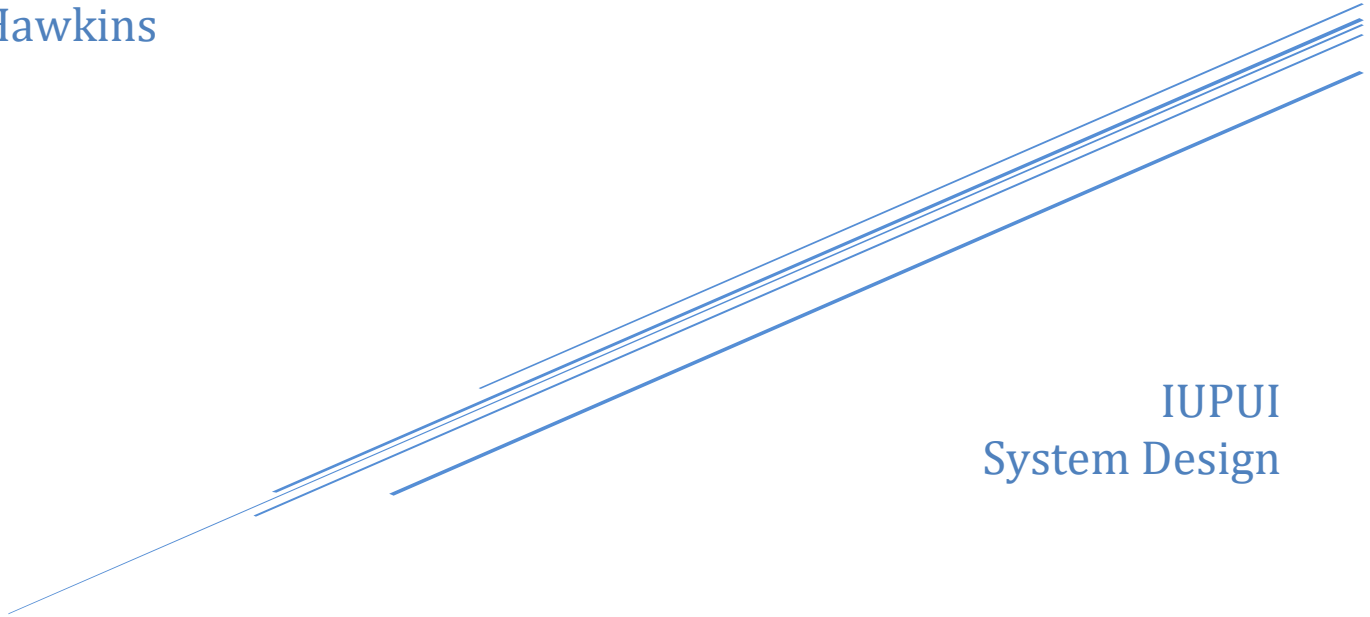
PARKING LOT MANAGEMENT SYSTEM

Elvis Ilor

Mason Hawkins



IUPUI

System Design

Contents

Traceability Matrix

System Requirements

No.	Priority Weight (1–5)	Description
REQ1	5	Generate reports for admin users.
REQ2	5	Drivers can pay for parking securely.
REQ3	4	Drivers receive notifications about reservations.
REQ4	4	Parking agents can issue tickets.
REQ5	3	Parking agents can manage payment disputes.

No.	Priority Weight (1–5)	Description
REQ6	3	System calculates billing accurately.

Use Cases

No.	Role	Description
UC1	Admin	Generate reports on parking lot performance, revenue, and utilization.
UC2	Driver	Pay for parking securely through the system.
UC3	Driver	Receive notifications about reservations and updates.

No.	Role	Description
UC4	Parking Agent	Issue tickets for parking violations.
UC5	Parking Agent	Manage payment disputes efficiently.
UC6	System	Automatically calculates billing for drivers based on usage.

Traceability Matrix

System Requirement ID	UC1	UC2	UC3	UC4	UC5	UC6	Max PW	Total PW
REQ1		✓					5	5

System Requirement ID	UC1	UC2	UC3	UC4	UC5	UC6	Max PW	Total PW
REQ2	✓						5	5
REQ3		✓					4	4
REQ4			✓				4	4
REQ5				✓			3	3
REQ6					✓		3	3

System Architecture and System Design

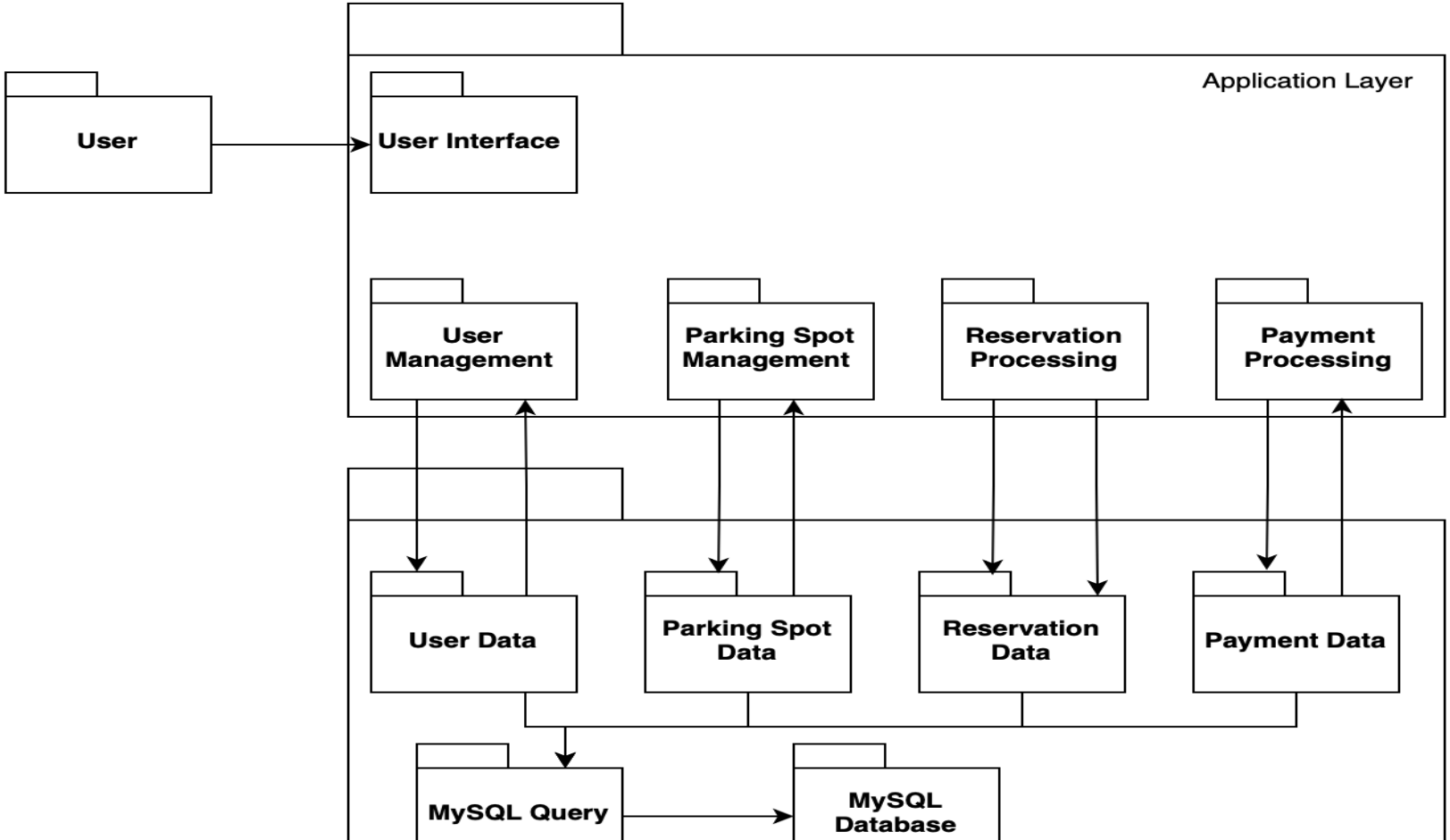
Architectural Styles

The Parking Lot Management System is a client-server architectural style. The client communicates with the server by sending requests and waiting for responses. The client does not need to understand the inner workings of the server to perform actions like reserving parking spaces or processing payments. The server is responsible for handling requests and delivering the necessary responses. We are using MySQL as the database to store persistent data, including user information, parking spots, reservations, and payments. Java, as the backend programming language, processes request and interacts with the database via JDBC. The user interface runs on the client side, and all data storage and logic processing occurs on the server side. The system will run on multiple machines, where users interact via a browser or application while the server handles backend processing and database interactions.

Identifying Subsystems

The UML package diagram below illustrates the subsystems of the Parking Lot Management System. This client-server system is divided into an application layer (client layer) and a database storage layer (server layer). The application layer includes packages for User Management, Parking Spot Management, Reservation Processing, and Payment Processing. The User Interface is built using JavaFX, allowing users to interact with the system through a graphical interface with buttons, forms, and tables. This interface accesses the Reservation and Parking Spot Management packages, which handle tasks like viewing availability, reserving spots, and updating statuses. These packages send SQL queries to the MySQL database to retrieve or update persistent data, such as reservation details or parking spot availability. The database responds with the requested data, which the system then displays to the user.

UML Package Diagram


User
User Interface
Application Layer
User Management
Parking Spot Management
Reservation Processing
Payment Processing
User Data
Parking Spot Data
Reservation Data
Payment Data
MySQL Query
MySQL Database

Persistent Data Storage

The Parking Lot Management System requires persistent storage to save data beyond a single execution. The persistent objects include Users (username, password, roles), Parking Spots (spot IDs, availability), Reservations (reservation IDs, user details, and payment status), and Payments (amounts, payment methods). The storage management strategy involves a relational database using MySQL, which ensures data integrity and efficient relational mapping between these entities.

Global Control Flow

Execution Orders

The Parking Lot Management System is an event-driven system that waits in a loop for user actions. Users interact with buttons and

forms on the interface to perform various actions, such as reserving a parking spot, making a payment, or freeing up a spot. These events trigger server-side actions such as processing SQL queries to update the database or returning updated data to the user interface. The system's flexibility allows users to perform actions in any order, enhancing user experience and functionality.

Time Dependency

The Parking Lot Management System does not rely on timers. It is an event-response system with no real-time constraints—actions such as viewing parking spot availability or making a reservation occur in response to user inputs. Data is updated dynamically on the server side and reflected immediately in the user interface upon the next interaction. This event-response model ensures smooth and asynchronous operation.

Hardware Requirements

The Parking Lot Management System depends on specific system resources to function:

- **Color Display:** Minimum resolution of 1280 x 720 pixels.
- **Client Device:** Desktop, laptop, or mobile device with browser support for HTML5 and JavaScript.
- **Memory:** At least 2 GB RAM.
- **Server Hardware:**
    - **Processor:** Multi-core CPU for concurrent threads.
    - **Memory:** Minimum 4 GB RAM.

    - **Storage:** At least 20 GB disk space for database and log files.

- **Network Bandwidth:** Minimum 1 Mbps for smooth interaction between client and server.

User Interface Design and Implementation

Overview

The Parking Lot Management System's user interface (UI) is designed with simplicity and usability in mind, ensuring that users can navigate the system efficiently. The UI is implemented using **JavaFX**, which offers a rich set of components and features for creating interactive and visually appealing interfaces.

UI Features

1. **Main Dashboard**:
    - Displays an overview of available, reserved, and occupied parking spots.
    - Provides access to different application sections based on user roles (Admin, Parking Agent, General User).

2. *Login and User Authentication:*

   - Secure login screen with role-based access. ○ Password encryption for security.

3. *User Functionalities:*

   - **View Available Parking Spots**:

- A dynamic table or grid layout showing available spots.
- Filters for location, size, and availability.

- **Reserve Parking Spot**:

- Form for selecting a parking spot and entering user details.
- Real-time availability check.

- **Payment Integration**:

- Payment form with fields for card details or PayPal integration.
- Confirmation message upon successful payment.

4. *Admin Functionalities:*

- **Manage Parking Spots**:

- Add, edit, or remove parking spots.

- Mark spots as under maintenance.

- **View All Reservations**:

- Table with details like user ID, reservation time, and payment status.

- **Generate Reports**:

- Export data in formats like CSV or PDF.

5. **Parking Agent Functionalities**:

    - **Assigning Vehicles to Spots**:

- Interface to select a parking spot and assign a vehicle.
- **Mark Spots as Free**:
- Button to update the status of a spot when a vehicle leaves.

6. **Error Handling**:
    - Alerts and notifications for invalid inputs.
    - Loading spinners and success messages for better user experience.

Design of Tests

Testing Methodologies

The testing process ensures that the application meets functional and non-functional requirements. Two primary testing methodologies are applied:

1. **Unit Testing**: Testing individual components of the application.

2. **Integration Testing**: Verifying interactions between multiple modules.

Test Scenarios and Cases

1. **Login and Authentication**:

    - **Scenario**: User enters valid credentials.

- **Expected Result**: Redirected to the appropriate dashboard based on the role.
    - **Scenario**: User enters invalid credentials.
    - **Expected Result**: Error message is displayed.
2. **View Available Parking Spots**:
    - **Scenario**: User filters spots by size.
    - **Expected Result**: Display of spots matching the filter.
    - **Scenario**: User views details of an available spot.

- **Expected Result**: Accurate information is displayed.

3. **Reserve Parking Spot**:
    - **Scenario**: User selects an already reserved spot.
- **Expected Result**: Error message and prompt to choose another spot.
- **Scenario**: User completes reservation.
- **Expected Result**: Reservation is saved, and confirmation is displayed.

4. **Payment Integration**:

- **Scenario**: User enters valid payment details.

- **Expected Result**: Payment is processed successfully, and confirmation is shown.

- **Scenario**: User enters invalid payment details.

- **Expected Result**: An error message is displayed.

5. *Admin Functionalities:*

- **Scenario**: Admin adds a new parking spot.

- **Expected Result**: Spot is added to the database and appears on the dashboard.

- **Scenario**: Admin removes an existing parking spot.
  - **Expected Result**: Spot is deleted, and users are notified.

6. **Parking Agent Functionalities**:
   - **Scenario**: Agent assigns a vehicle to a spot.
     - **Expected Result**: Spot status updates to "Occupied." ○ **Scenario**: Agent marks a spot as free.
       - **Expected Result**: Spot status updates to "Available."

Tools and Frameworks

- **JUnit**: For unit testing of the Java application.

- **Selenium**: For testing the UI functionalities.
- **MySQL Workbench**: For verifying database operations.

Test Execution

1. **Test Environment**:
    - OS: macOS 12.0+
    - Database: MySQL
    - Development Environment: IntelliJ IDEA
2. **Testing Workflow**:

- Prepare test cases and scripts.

- Execute test cases for each module.

- Log defects and retest after fixes. ○ Generate test reports.

3. **Metrics test**:

   - Test Coverage: Percentage of application features covered by tests.

   - Defect Density: Number of defects per module.

   - Pass Rate: Percentage of test cases that passed successfully.

Conclusion

The user interface design and testing strategy ensures a seamless and reliable user experience. The iterative approach to design and testing allows for continuous improvement, aligning with project goals and user needs.

User Interface Specifications

Use Case: Reserving a Parking Spot

Flow of Events:

1. **Click ‘Reserve Spot’**: The user clicks the "Reserve Spot" button on the dashboard.

2. **Enter Vehicle Details**: The user enters their license plate number and enters the desired parking time from into a text field.

3. **Select Parking Spot**: The user is shown available spots in the chosen lot, and they select one from a map.

4. **Confirm Reservation**: The user reviews the details and clicks the "Confirm Reservation" button.

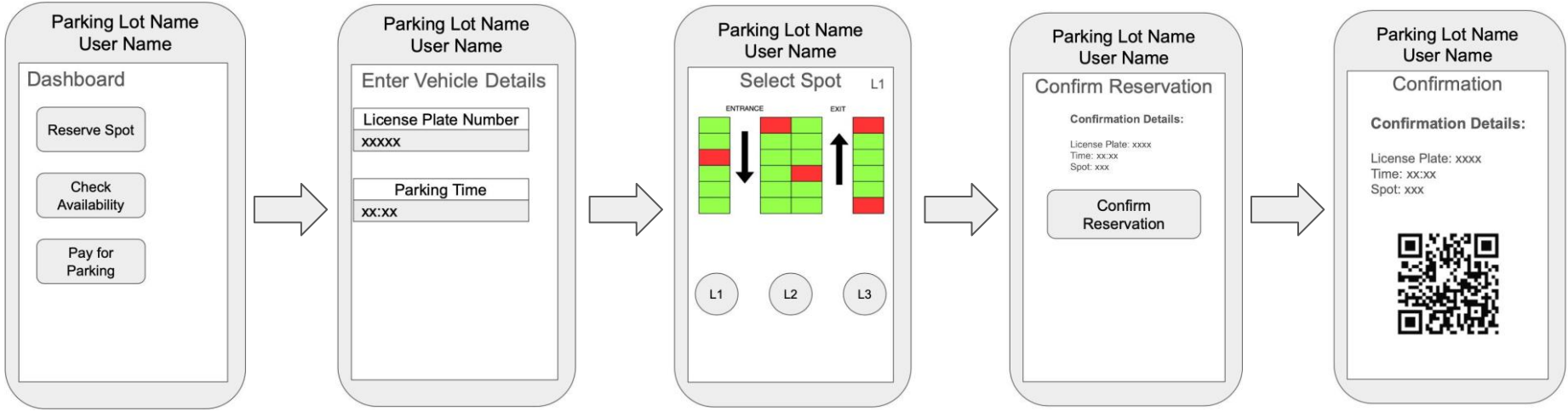
5. **Receive Confirmation**: A confirmation screen appears, displaying the reservation details and a QR code for parking access.

Navigation Path:

- Dashboard → Reserve Spot → Enter Vehicle Details → Select Spot → Confirmation Screen **User Effort Estimation:**

●**Mouse clicks**: 5 (Click ‘Reserve Spot’, select time, choose spot, confirm, back to dashboard) **Keystrokes**: 8 (Enter license plate number)

Use Case: Reserving a Parking Spot



Click Reserve Spot	Enter Vehicle Details	Select an available spot (green)	Confirm Reservation	Confirmation Screen

Use Case: Checking Parking Availability

Flow of Events:

1. **Click ‘Check Availability’**: The user clicks the "Check Availability" button on the dashboard.
2. **Select Parking Lot**: A drop-down menu appears, and the user selects the parking lot they want to check.
3. **View Available Spots**: The system displays a map or list showing the available parking spots in the selected lot.
4. **Return to Main Menu**: The user clicks "Back" to return to the dashboard.

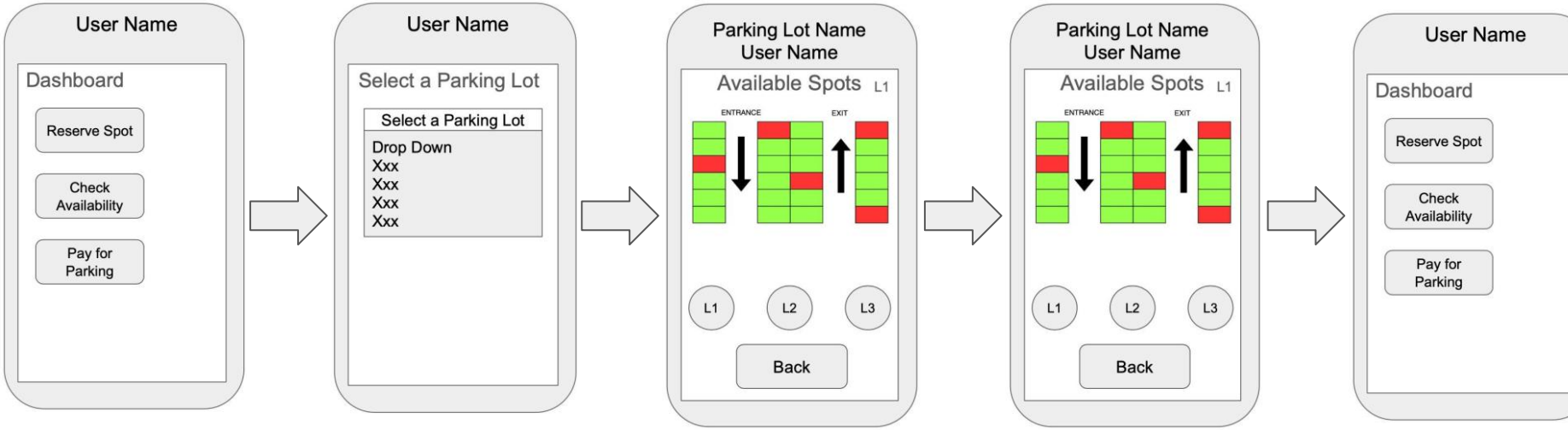
Navigation Path:

- Dashboard → Check Availability → Select Parking Lot → Available Spots Display → Back to Dashboard

User Effort Estimation:

: 4 (Click ‘Check Availability’, select lot, back to dashboard) : 0

Use Case: Checking Parking Availability



Click Check Availability

Select Parking Lot

View Available Spots

Click Back to Return to Dashboard

Use Case: Paying for Parking

Flow of Events:

1. **Click ‘Pay for Parking’**: The user clicks the "Pay for Parking" button on the dashboard.

2. **Enter Payment Details**: The user selects their payment method (e.g., credit card or mobile payment) and enters the necessary details.

3. **Confirm Payment**: After reviewing the payment summary, the user clicks the "Confirm Payment" button.

4. **Receive Payment Confirmation**: The user receives a confirmation message and receipt.

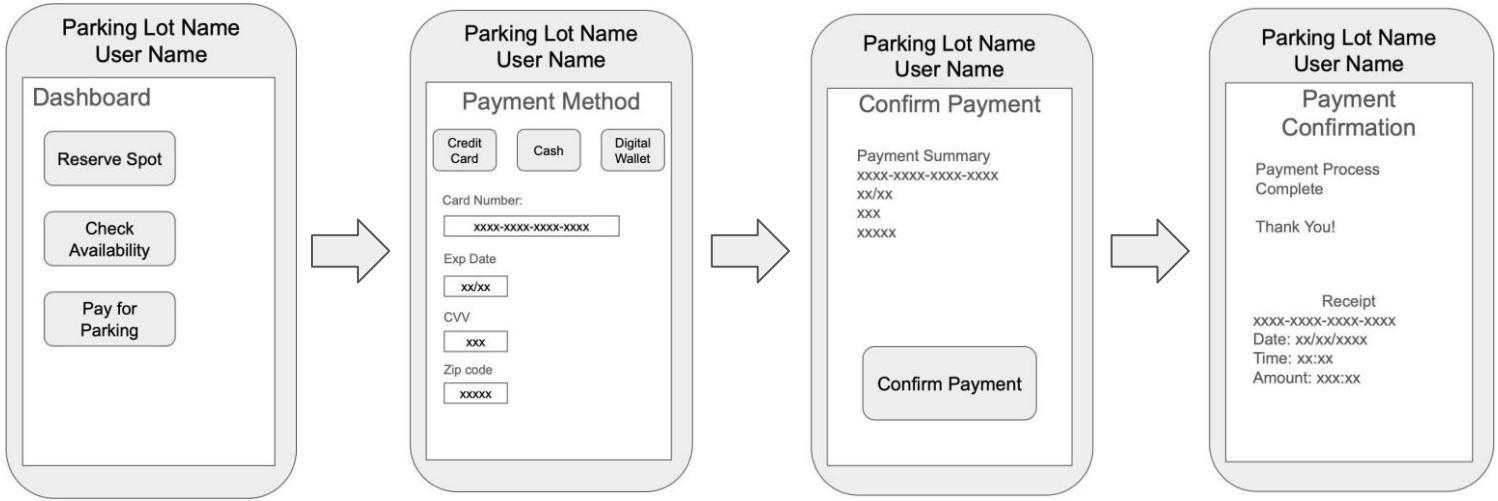
Navigation Path:

- Dashboard → Pay for Parking → Enter Payment Details → Confirm Payment → Payment Confirmation Screen

User Effort Estimation:

: 4 (Click ‘Pay for Parking’, select a payment method, confirm payment, back to the dashboard) : 28 (Enter payment details)

Use Case: Paying for Parking



Click Pay for Parking	Select Payment Method **Credit card**: enter details **Digital Wallet**: takes the user straight to confirmation screen **Cash**: Prompts the user to go to gate	Click Confirm Payment	Receive Payment Confirmation and Receipt

Use Case: Admin: Viewing Parking Lot Status

Flow of Events:

1. **Click ‘View Lot Status’**: The admin clicks the "View Lot Status" button.

2. **Select Parking Lot**: A dropdown menu allows the admin to select which parking lot's status they wish to view.

3. **View Lot Status**: A dashboard shows real-time data on lot availability, occupancy rates, and other metrics.

4. **Return to Admin Dashboard**: The admin can return to the main dashboard by clicking "Back".

Navigation Path:

- Admin Dashboard → View Lot Status → Select Lot → View Data → Back to Admin Dashboard

User Effort Estimation:

: 3 (Click 'View Lot Status', select the lot, back to the dashboard) : 0

Use Case: Admin: Viewing Parking Lot Status

Click View Lot Status

Select Parking Lot

View of real time

Admin Dashboard

Parking Lot data

Back Returns to Admin

Dashboard

Use Case: Admin: Managing Parking Rates

Flow of Events:

1. **Click ‘Manage Rates:** The admin clicks the "Manage Rates" button on the dashboard.
2. **Select Lot**: A dropdown menu allows the admin to select the parking lot for which they want to adjust rates.
3. **Enter New Rate**: The admin enters the new hourly or daily parking rate.
4. **Confirm Changes**: The admin clicks the "Confirm" button to apply the new rates.
5. **Receive Confirmation**: A confirmation message appears indicating the rate changes have been applied successfully.

Navigation Path:

- Admin Dashboard → Manage Rates → Select Lot → Enter New Rate → Confirm Changes → Confirmation Screen

User Effort Estimation:

: 4 (Click ‘Manage Rates’, select a lot, confirm changes, back to the dashboard) : 4 (Enter new rate)

Use Case: Admin: Managing Parking Rates

Click Manage Parking Rates

Select Parking Lot

Enter new Parking Rates and Select

Green Confirmation verifying changes

	Confirm Changes to save	saved, Back Button returns to Admin Dashboard

User Effort Estimation

Use Case	Clicks	Keystroke	Navigation path
Reserving a Parking Spot	5	8	Dashboard → Reserve Spot → Enter Vehicle Details → Select Spot → Confirmation Screen
Checking Parking Availability	4	0	Dashboard → Check Availability → Select Parking Lot → Available Spots Display → Back to Dashboard
Paying for Parking	4	28	Dashboard → Pay for Parking → Enter Payment Details → Confirm Payment → Payment Confirmation Screen
Admin: Viewing Parking Lot Status	3	0	Admin Dashboard → View Lot Status → Select Lot → View Data → Back to Admin Dashboard

Admin: Managing Parking Rates	4	4	Admin Dashboard → Manage Rates → Select Lot → Enter New Rate → Confirm Changes → Confirmation Screen