Graphical user interface

Description automatically generated with medium confidence

**Project Report**

**An Assignment presented to**

**Ma’am Rabail Zahid**

**In partial fulfillment**

**of the requirement for the course of**

***Software Engineering CS-3009***

**By**

**Munhim Hussain**

**(22i-1021)**

**Hamza Imran**

**(22i-0865)**

**Umer Khan**

**(22i-0780)**

Contents

[***1.******Project Introduction*** 3](#_Toc196781539)

[**1.1 Purpose** 3](#_Toc196781540)

[**1.2 Intended Audience** 4](#_Toc196781541)

[***2. Functional and Non Functional Requirements*** 4](#_Toc196781542)

[**2.1 Functional Requirements** 4](#_Toc196781543)

[**2.2 Non Functional Requirements** 4](#_Toc196781544)

[***3. User Stories*** 5](#_Toc196781545)

[**3.1 User Story 1: Vehicle Owner Registration** 5](#_Toc196781546)

[**3.2 User Story 2: Vehicle Registration** 6](#_Toc196781547)

[**3.3 User Story 3: Admin Parking Space Setup** 6](#_Toc196781548)

[**3.4 User Story 4: Book a Parking Space** 6](#_Toc196781549)

[**3.5 User Story 5: Manage Reservations** 7](#_Toc196781550)

[**3.6 User Story 6: Admin Parking Space Monitoring** 7](#_Toc196781551)

[**3.7 User Story 7: Change Password** 7](#_Toc196781552)

[**3.8 User Story 8: Logging out of Account** 8](#_Toc196781553)

[**3.9 User Story 9: Setup parking rules** 8](#_Toc196781554)

[**3.10 User Story 10: See Profile details** 8](#_Toc196781555)

[**3.11 User Story 11: Invoice for parking reservations** 9](#_Toc196781556)

[**3.12 User Story 12: Record of all parking spaces** 9](#_Toc196781557)

[**3.13 User Story 13: Reminder before reservation expires** 9](#_Toc196781558)

[**3.14 User Story 14: User Feedback** 10](#_Toc196781559)

[**3.15 User Story 15: List of all registered users** 10](#_Toc196781560)

[**3.16 User Story 16: Reset Vehicle owner’s password** 10](#_Toc196781561)

[***4. Product Backlog*** 11](#_Toc196781562)

[***5. Sprint 1 and 2 Backlog*** 11](#_Toc196781563)

[13](#_Toc196781564)

[***6. Project Plan*** 14](#_Toc196781565)

[**1. Project Management** 14](#_Toc196781566)

[**2. Requirements Analysis** 14](#_Toc196781567)

[**3. System Design** 14](#_Toc196781568)

[**4. Frontend Development** 14](#_Toc196781569)

[**5. Backend Development** 14](#_Toc196781570)

[**6. Database Setup** 15](#_Toc196781571)

[**7. Integration** 15](#_Toc196781572)

[**8. Testing & QA** 15](#_Toc196781573)

[**9. Maintenance & Support** 15](#_Toc196781574)

[***7. Architecture Diagrams*** 16](#_Toc196781575)

[**1. Layered Architecture (N-Tier Architecture)** 18](#_Toc196781576)

[**2. Client-Server Architecture** 18](#_Toc196781577)

[***8. Actual Project Screenshots*** 19](#_Toc196781578)

[***9. Burn down Chart*** 20](#_Toc196781579)

[***10. Trello Board*** 21](#_Toc196781580)

[***11. Black box Testing*** 23](#_Toc196781581)

[***12. Work Division*** 27](#_Toc196781582)

[***12. Lessons Learnt*** 28](#_Toc196781583)

# ***1.******Project Introduction***

## **1.1 Purpose**

The **Vehicle Parking Management System (VPMS)** is a comprehensive software solution designed to enhance the efficiency of managing parking spaces in public and private facilities. This document outlines the software requirements for **VPMS Version 1.0**.

The system enables **vehicle owners** to register, add their vehicle information, search for available parking spots, make reservations, and oversee their parking activities. For **administrators**, the VPMS offers functionalities to configure parking slots, monitor space usage, handle bookings, and generate performance reports.

The primary objectives of VPMS include:

* Minimizing human errors in parking operations.
* Maximizing parking space utilization through smart allocation.
* Enhancing the user experience with quick and seamless reservations.
* Delivering real-time updates on slot availability and booking status.

This Software Requirements Specification (SRS) defines both **functional** and **non-functional** aspects of the system.

## **1.2 Intended Audience**

This document is intended for the following stakeholders:

* **Software Developers** – to implement system features.
* **Project Managers** – to oversee progress and deadlines.
* **System Administrators** – to configure and maintain system components.
* **QA Testers** – to verify conformance to requirements.
* **End Users** (Vehicle Owners & Admins) – to understand system features and limitations.

# ***2. Functional and Non Functional Requirements***

## **2.1 Functional Requirements**

* **User Registration**: Allows users to register and manage their profiles.
* **Vehicle Registration**: Enables users to register vehicles with license plate and type.
* **Parking Space Search**: Users can search for available parking spaces using filters and time slots.
* **Booking Parking Spots**: Users can reserve specific parking spots in advance.
* **Booking Confirmation**: System sends confirmation notification post booking.
* **Reservation Management**: Users can view, update, or cancel existing bookings.
* **Admin Parking Setup**: Admins can configure spots, pricing, and parking rules.
* **Admin Dashboard**: Enables real-time monitoring, system settings, and analytics.
* **Notification System**: Automatic alerts for bookings, cancellations, and expirations.
* **Invoice Generation**: Generates digital receipts for completed transactions.
* **Parking Space Records**: Maintains logs and live availability of parking spots.
* **User Feedback**: Users can submit feedback and service ratings.
* **User Authentication**: Secure login system with password reset features.
* **Reporting & Analytics**: Generates various reports including usage statistics.
* **Profile Management**: Allows updating personal or vehicle information.

## **2.2 Non Functional Requirements**

**Performance**

* **NFR-1**: Support at least 100 concurrent users.
* **NFR-2**: Search results should return within 2 seconds.
* **NFR-3**: Bookings should be processed within 3 seconds.
* **NFR-4**: Notifications must be sent within 5 seconds.

**Usability**

* **NFR-5**: Intuitive user interface that is easy to learn.
* **NFR-6**: Interface must meet WCAG 2.1 AA accessibility guidelines.

**Security**

* **NFR-7**: All user data must be encrypted (AES-256).
* **NFR-8**: Session auto-logout after 15 minutes of inactivity.

**Scalability**

* **NFR-9**: System should scale to support up to 500 users.
* **NFR-10**: Database must accommodate 1 million+ records.

**Availability**

* **NFR-11**: Minimum 99.5% uptime.
* **NFR-12**: Disaster recovery within 10 minutes.

**Organizational**

* **NFR-13**: GDPR compliance is mandatory.
* **NFR-14**: Follow ISO 27001 standards.
* **NFR-15**: Agile-based development with 2-week sprints.
* **NFR-16**: Git-based version control.
* **NFR-17**: Security updates must be applied quarterly.
* **NFR-18**: Admin support available 24/7.

**External Integration**

* **NFR-19**: Use MongoDB for data storage.
* **NFR-20**: Integration with SMS and email APIs.
* **NFR-21**: (Future) Enable digital payment gateways.
* **NFR-22**: Refunds must be processed within 48 hours.
* **NFR-23**: All communications must use TLS 1.2+.
* **NFR-24**: System must support both IPv4 and IPv6.

# ***3. User Stories***

## **3.1 User Story 1: Vehicle Owner Registration**

**Pre-conditions:**

* The user has access to the registration page.
* The user has valid personal details.
* The system is available.

**Post-conditions:**

* The user account is created.
* The user receives a confirmation of registration.

## **3.2 User Story 2: Vehicle Registration**

**Pre-conditions:**

* The user has a registered account.
* The user has vehicle details available.

**Post-conditions:**

* The vehicle is registered in the system.
* The user receives a confirmation message.

## **3.3 User Story 3: Admin Parking Space Setup**

**Pre-conditions:**

* The admin is logged in.
* The system allows configuration of parking spaces.

**Post-conditions:**

* Parking spaces are defined and stored.
* Parking rules are set.

## **3.4 User Story 4: Book a Parking Space**

**Pre-conditions:**

* The user is logged in.
* The user has a registered vehicle.

**Post-conditions:**

* The parking space is reserved.
* The user receives booking confirmation.

## **3.5 User Story 5: Manage Reservations**

**Pre-conditions:**

* The user is logged in.
* The user has active reservations

**Post-conditions:**

* Reservations are updated or canceled.

## **3.6 User Story 6: Admin Parking Space Monitoring**

**Pre-conditions:**

* The admin is logged into the system with appropriate permissions.
* Parking occupancy data is being updated in real-time.
* The system is operational and able to track parking usage.

**Post-conditions:**

* The admin can view the real-time status of all parking spaces (available, occupied, or reserved).
* The system updates space availability dynamically.

## **3.7 User Story 7: Change Password**

**Pre-conditions:**

* The user has an active parking reservation.
* The vehicle owner is logged in.
* The change password page is accessible with a field for the current password.

**Post-conditions:**

* The new password replaces the old one in the system.
* A confirmation message is displayed.

## **3.8 User Story 8: Logging out of Account**

**Pre-conditions:**

* The vehicle owner is logged into their account.
* The logout option is visible and accessible on the interface.

**Post-conditions:**

* The active session is terminated, removing all session data.
* The user is redirected to the home page.

## **3.9 User Story 9: Setup parking rules**

**Pre-conditions:**

* The parking administrator has accessed the rule-setting interface.
* The system verifies that the user has the privileges to set or modify parking rules.

**Post-conditions:**

* The new parking duration rules are successfully saved in the system.

## **3.10 User Story 10: See Profile details**

**Pre-conditions:**

* The admin is logged in.
* The system has stored the user's profile details.
* The user navigates to the profile section.

**Post-conditions:**

* The system retrieves and displays the user's profile details.
* The vehicle owner can view their account information, including name, contact details, and registered vehicles.

## **3.11 User Story 11: Invoice for parking reservations**

**Pre-conditions:**

* The system has recorded the reservation details, including payment information if applicable.
* The vehicle owner has successfully booked a parking reservation.
* The admin is logged in.

**Post-conditions:**

* The system generates a receipt or invoice containing the reservation details (date, time, parking space, cost, etc.).

## **3.12 User Story 12: Record of all parking spaces**

**Pre-conditions:**

* The system is actively tracking parking space reservations.
* Each reservation includes relevant details such as vehicle owner, parking space ID, start time, end time, and payment (if applicable).

**Post-conditions:**

* All parking space usage data is recorded and stored securely.

## **3.13 User Story 13: Reminder before reservation expires**

**Pre-conditions:**

* The user has an active account and an associated booking ID.
* The system is configured to send expiration reminders before the reservation ends.

**Post-conditions:**

* The vehicle owner receives a reminder notification before the reservation expires.
* The system logs the reminder delivery for tracking purposes

## **3.14 User Story 14: User Feedback**

**Pre-conditions:**

* The user has completed a parking session and checked out. The user has a registered vehicle with type details.
* The user has an active account and an associated booking ID.

**Post-conditions:**

* The feedback is recorded and stored in the database.
* The parking spaces **average rating** is updated based on new reviews.

## **3.15 User Story 15: List of all registered users**

**Pre-conditions:**

* The admin is authenticated and has the necessary permissions to view user records.
* The system has a database of registered users.
* The admin accesses the user management section of the system.

**Post-conditions:**

* The admin can view, filter, or search for specific users.
* The system displays a list of all registered users, including relevant details (e.g., name, email, account status).

## **3.16 User Story 16: Reset Vehicle owner’s password**

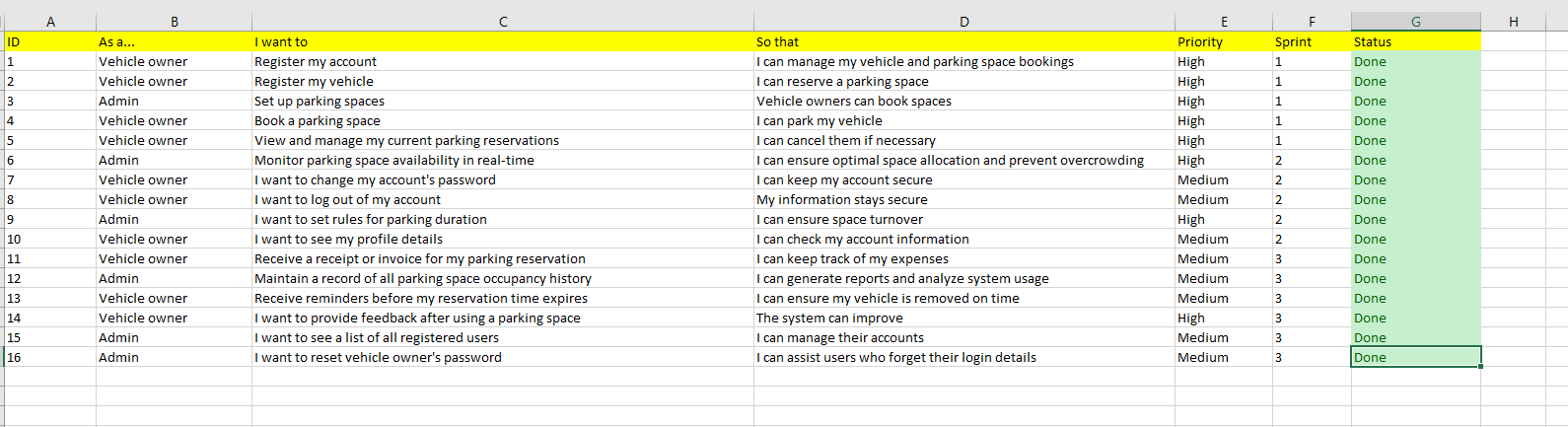
**Pre-conditions:**

* The admin is authenticated and has the necessary permissions to reset user passwords.
* The user requesting the reset exists in the system.

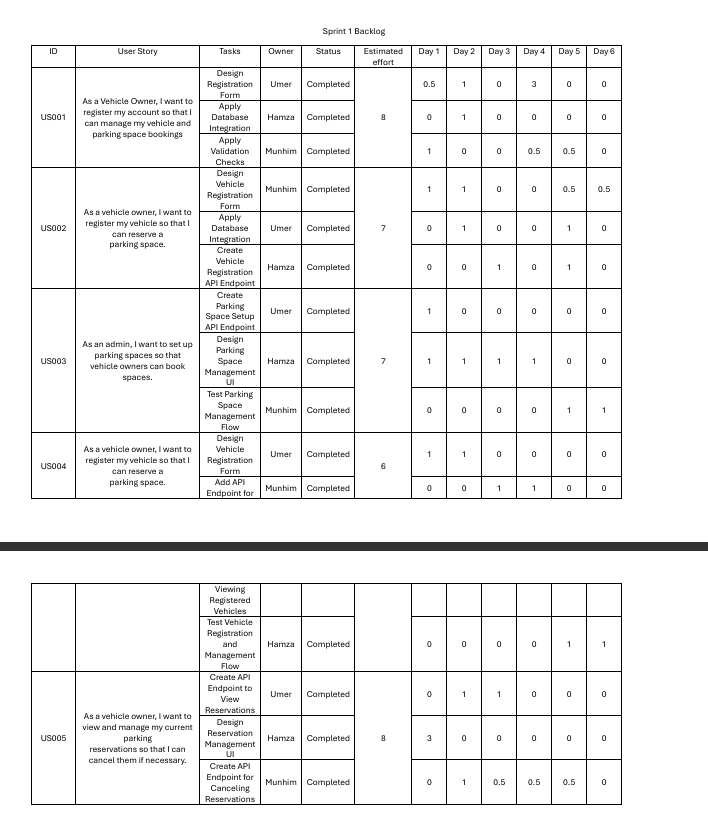
**Post-conditions:**

* The user's password is successfully reset to a temporary password or a new password set by the admin.
* The user is notified of the password reset via email or another communication method.

# ***4. Product Backlog***



# ***5. Sprint 1 and 2 Backlog***



# 

# ***6. Project Plan***

**Project Breakdown**

## **1. Project Management**

1.1 Project Planning  
1.2 Resource Allocation  
1.3 Timeline Estimation  
1.4 Risk Assessment  
1.5 Progress Monitoring  
1.6 Documentation & Reporting

## **2. Requirements Analysis**

2.1 Requirement Gathering  
2.2 Stakeholder Meetings  
2.3 SRS Documentation  
2.4 Approval & Sign-off

## **3. System Design**

3.1 System Architecture Design  
3.2 Database Schema (ER Diagram)  
3.3 Use Case & Sequence Diagrams  
3.4 UI/UX Design

## **4. Frontend Development**

4.1 Login/Registration Pages  
4.2 Dashboard for Vehicle Owners  
4.3 Admin Dashboard  
4.4 Parking Space Search & Booking UI  
4.5 Reservation Management Interface  
4.6 Notification Popups & History  
4.7 Profile & Feedback Pages

## **5. Backend Development**

5.1 Authentication & Authorization  
5.2 User & Vehicle Management APIs  
5.3 Parking Space Management Logic  
5.4 Booking & Reservation Module  
5.5 Invoice Generation Module

## **6. Database Setup**

6.1 Schema Creation in MongoDB  
6.2 Indexing & Query Optimization  
6.3 Data Validation & Constraints  
6.4 Reservation History Storage

## **7. Integration**

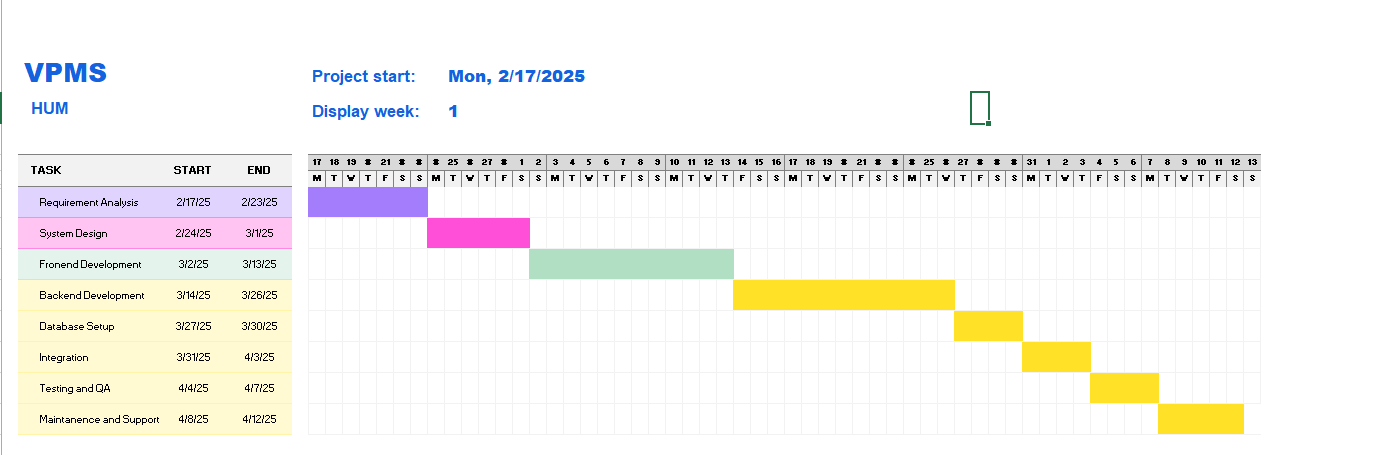
7.1 Frontend-Backend API Integration  
7.2 Notification System Setup  
7.4 Error Handling & Logging

## **8. Testing & QA**

8.1 Unit Testing  
8.2 Integration Testing  
8.3 System Testing  
8.4 Security Testing  
8.5 Usability Testing

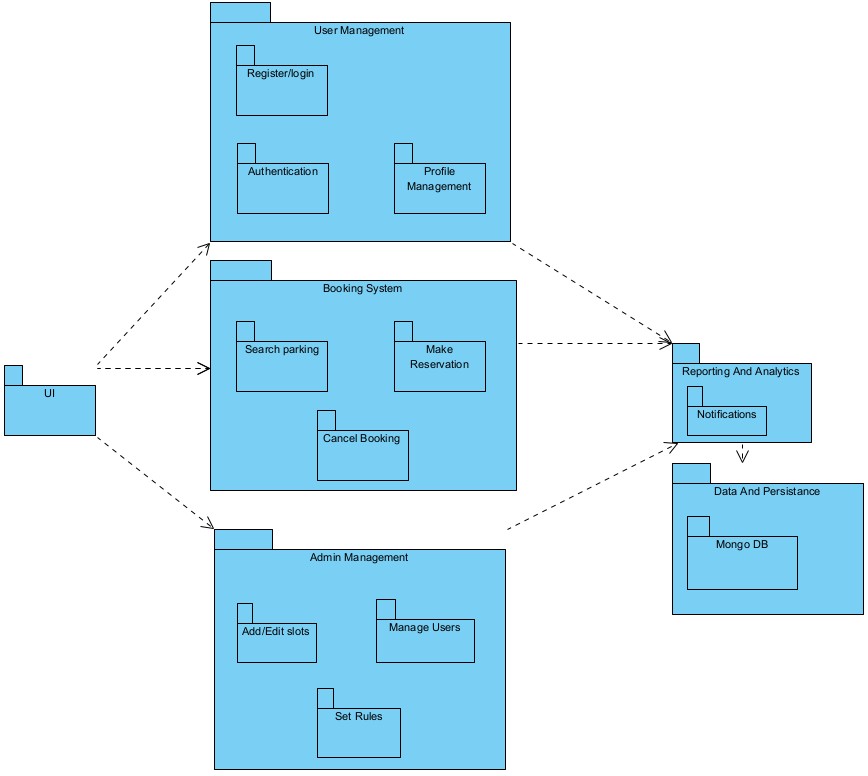
## **9. Maintenance & Support**

9.1 Bug Fixes & Patches  
9.2 Security Updates (Quarterly)  
9.3 24/7 Admin Support Setup  
9.4 System Monitoring & Logs

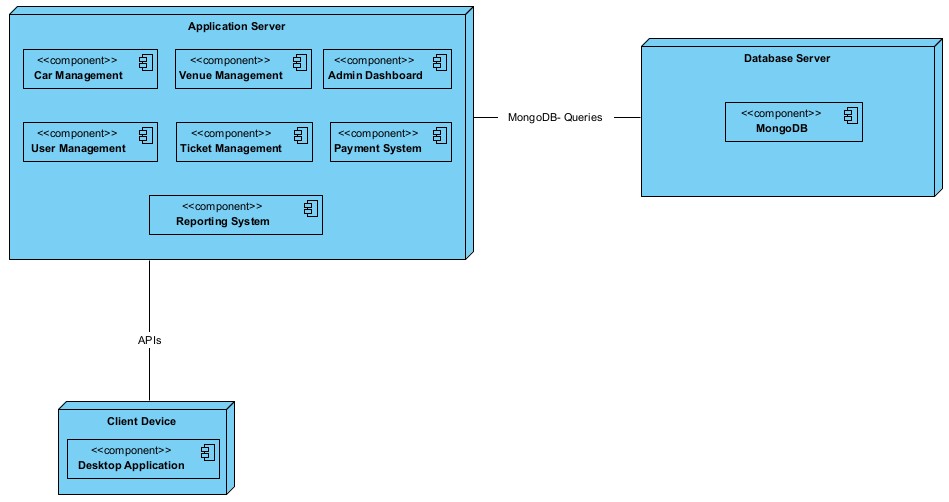


# ***7. Architecture Diagrams***

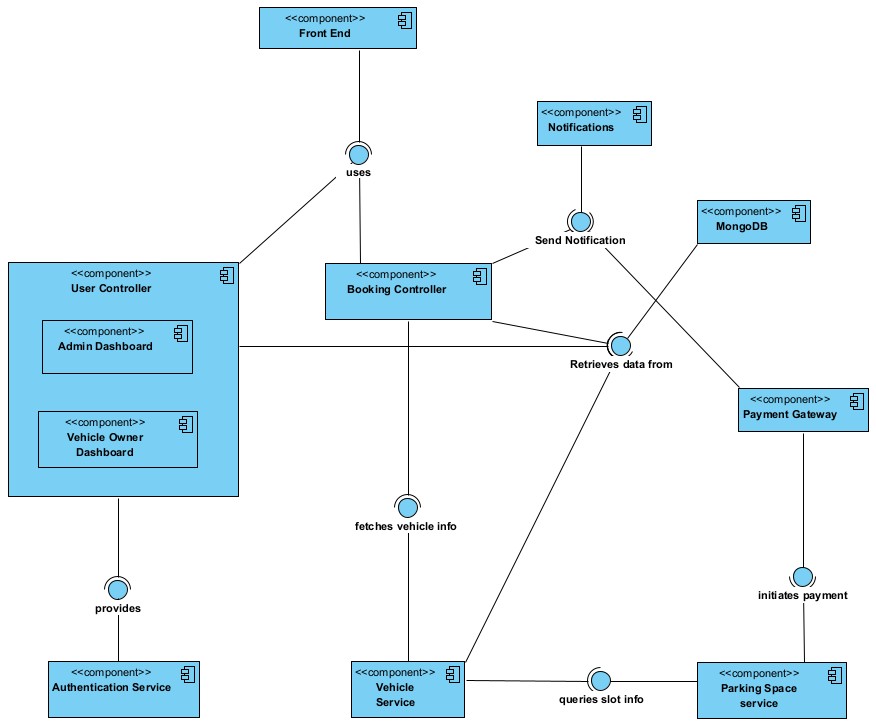
**Package Diagram**



**Deployment Diagram**



**Component Diagram**



**Architecture Styles**

## **1. Layered Architecture (N-Tier Architecture)**

The VPMS follows a layered architecture where the system is divided into logical layers:

* **Presentation Layer (Client Side):**
  + Built using HTML5, CSS3, and JavaScript.
  + Provides interfaces for Vehicle Owners and Admins to interact with the system.
* **Business Logic Layer (Application Server):**
  + Node.js backend handles application rules like booking logic, user management, admin operations, etc.
* **Data Access Layer:**
  + Communicates with MongoDB through Mongoose ORM.
  + Encapsulates database operations (CRUD on users, vehicles, reservations).
* **External Services Layer:**
  + Integrates with external APIs (e.g., Email/SMS Notification services).

**Reason:**   
Separation of concerns makes the system modular, easier to test, and maintainable.

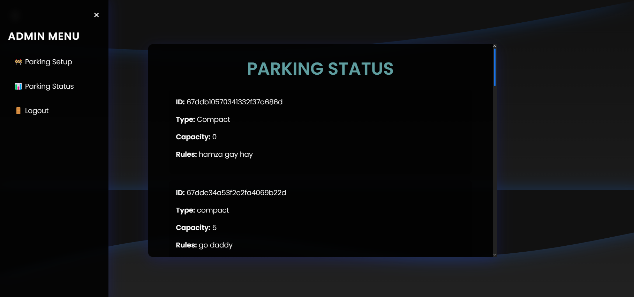
## **2. Client-Server Architecture**

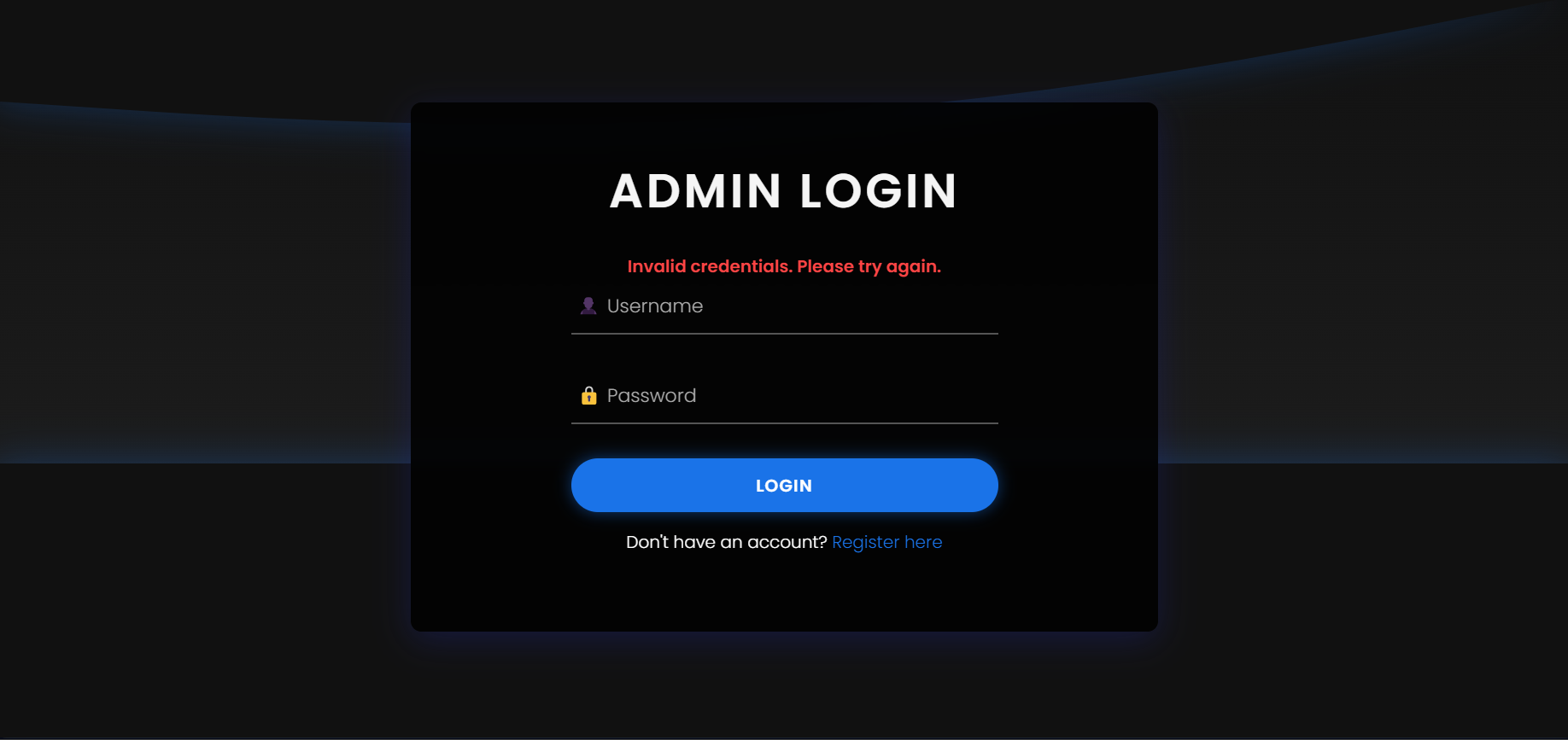
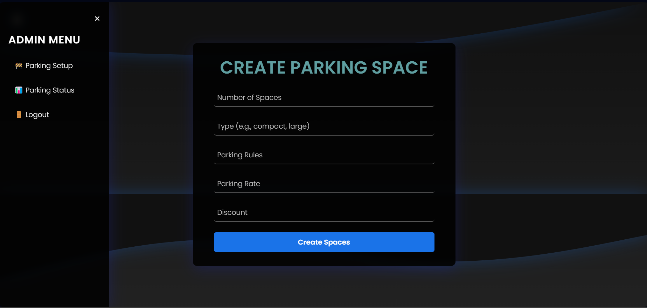
The system uses a classic client-server model:

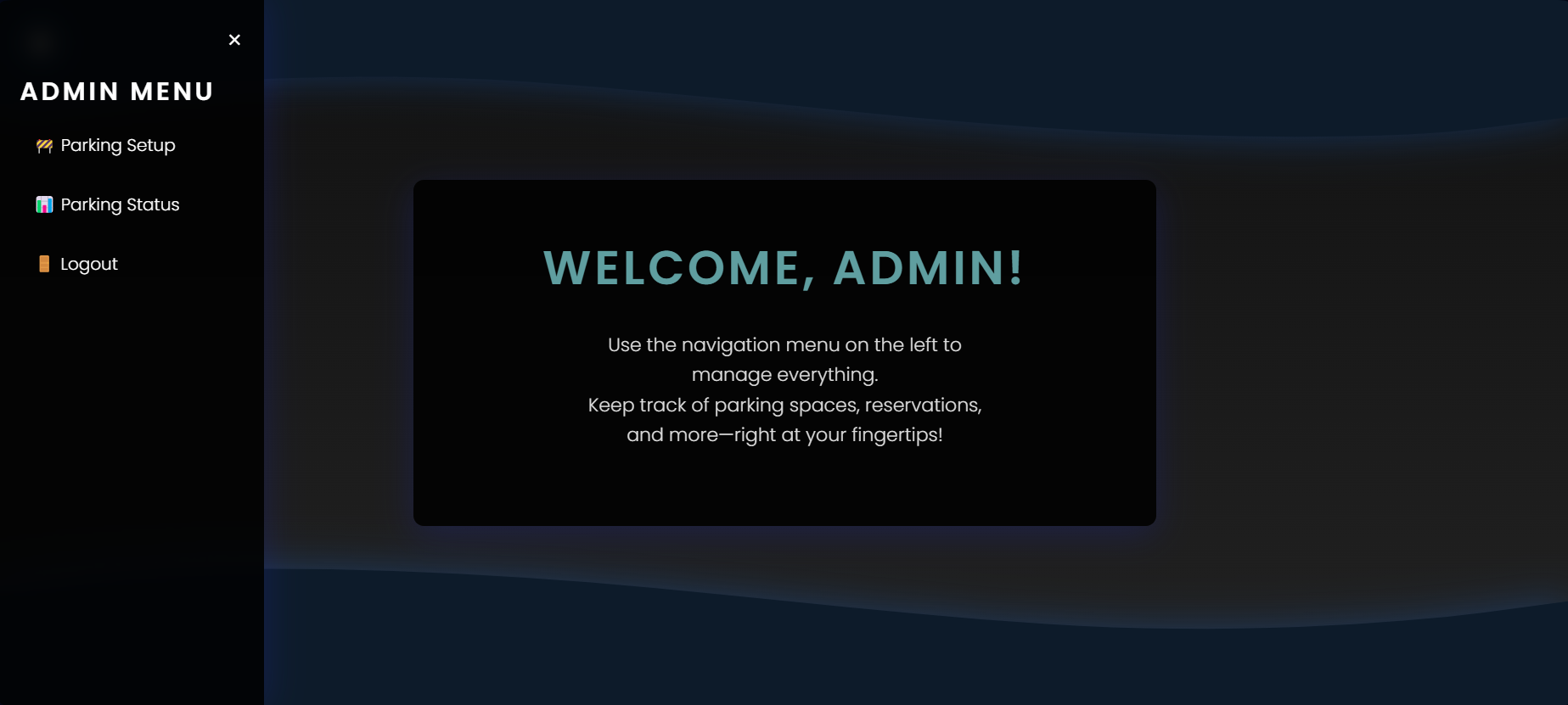
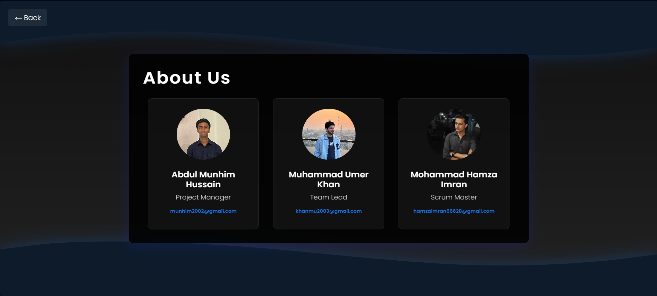
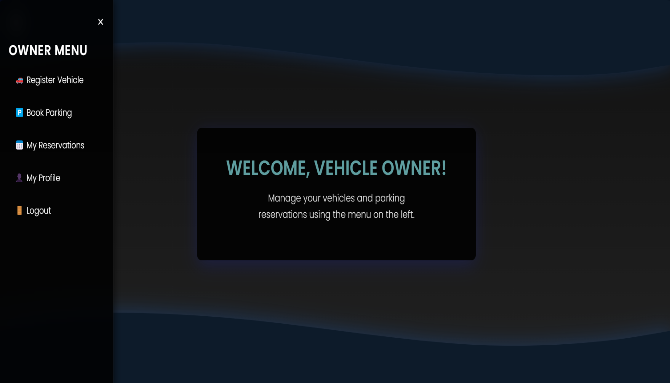
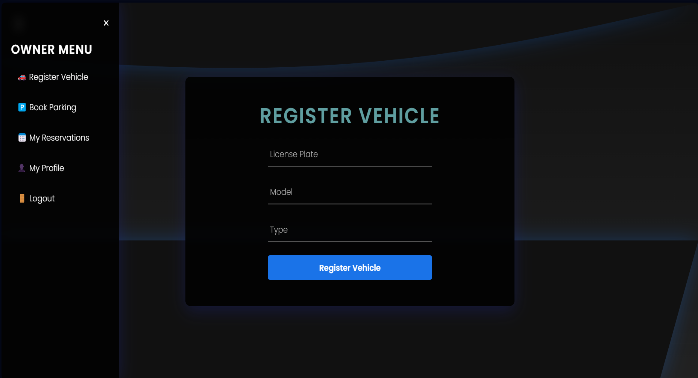
* **Client Devices (Web):**
  + Act as clients making HTTP requests.
* **Application Server (Node.js):**
  + Acts as the server, processing requests, executing business logic, and returning results.
* **Database (MongoDB):**
  + Acts as the persistent data store, accessed only by the server.

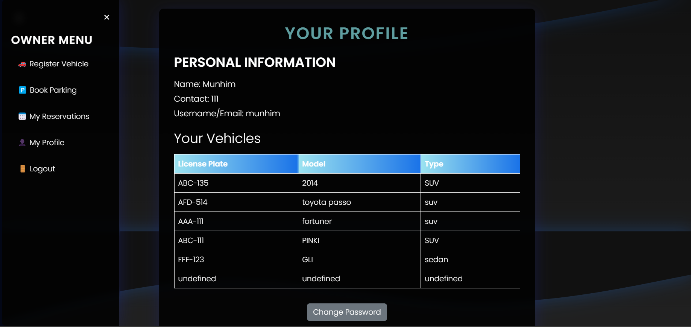
**Reason:**   
Ensures centralized control over data and logic.  
Allows multiple different clients (web app, mobile app) to use the same server backend.

# ***8. Actual Project Screenshots***

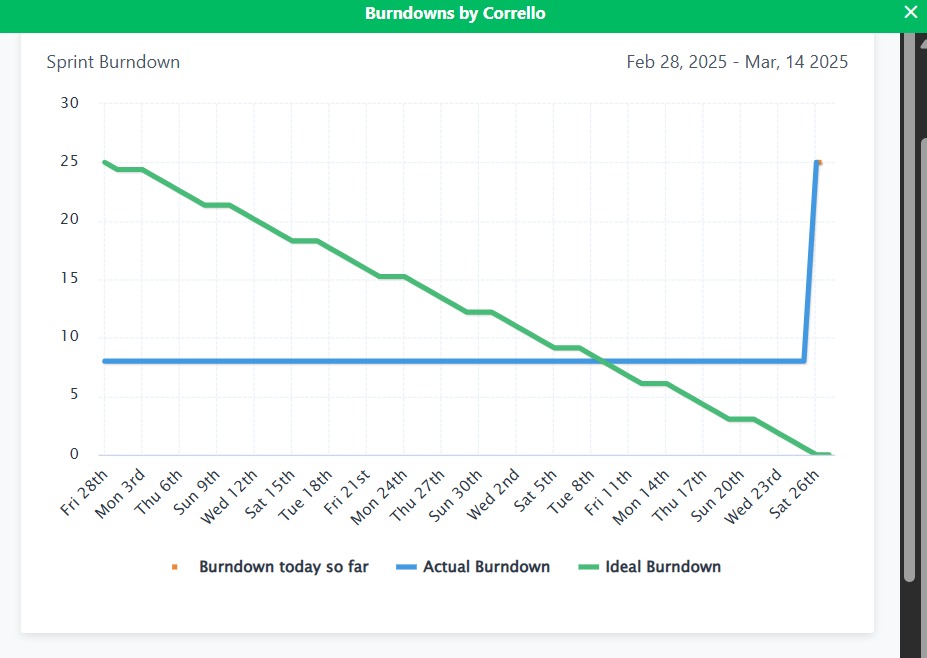




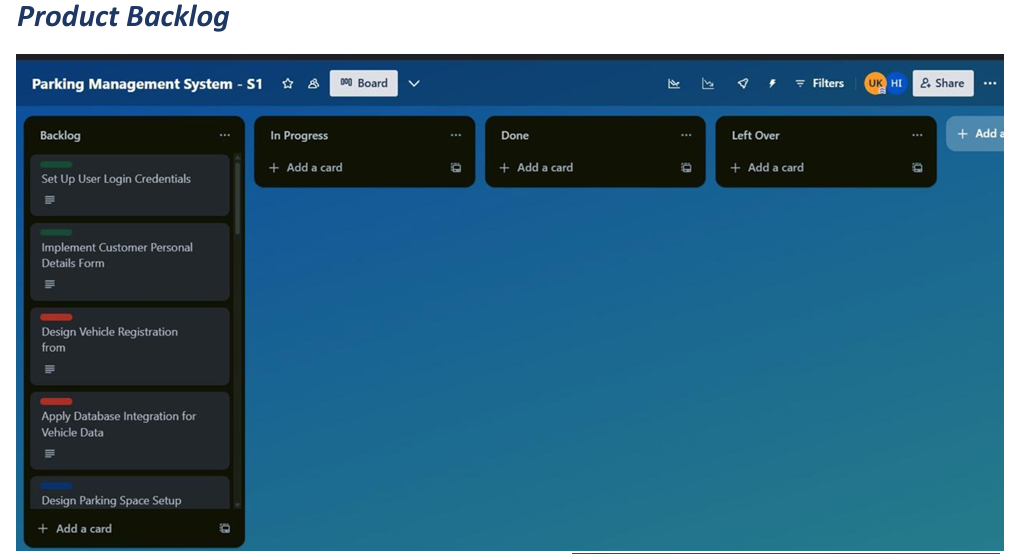


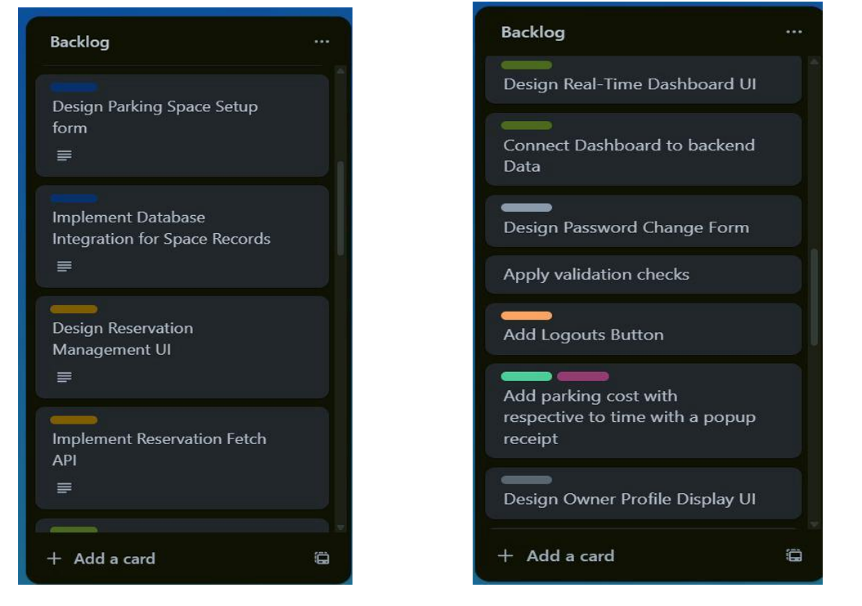


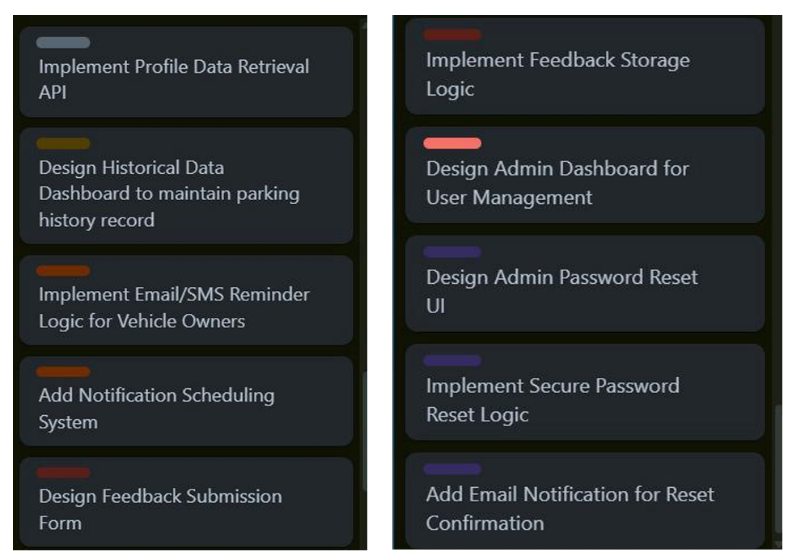
# ***9. Burn down Chart***

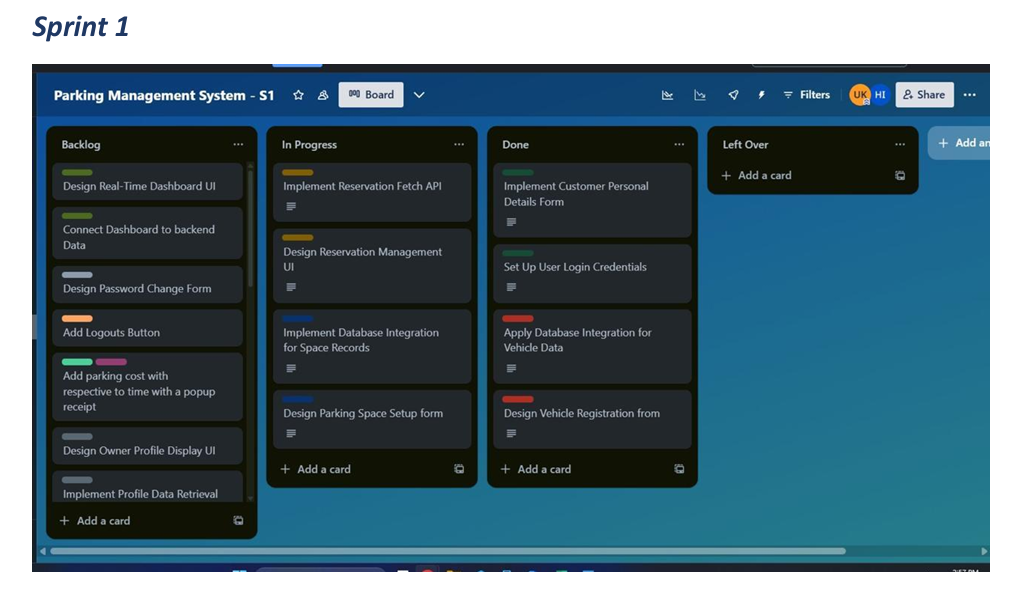


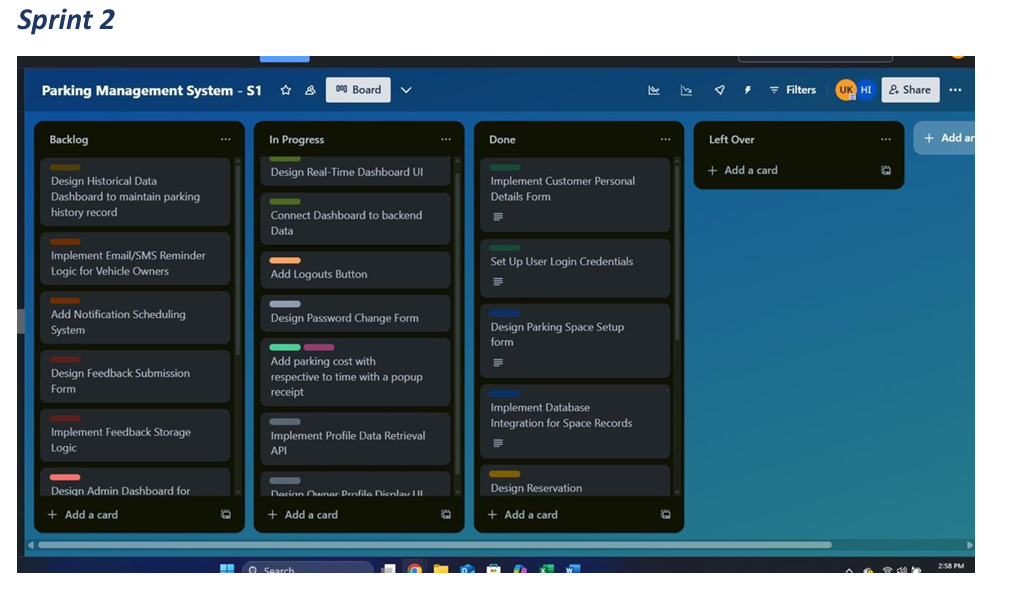
# ***10. Trello Board***











# ***11. Black box Testing***

**1. User Registration**

• UR1 — Name:

* { 1 ≤ length(name) AND matches [A-Z]+ | [a-z]+ }

• UR2 — Username:

* { 1 ≤ length(name) }

• UR3 — Password:

* { 5 ≤ length(password) }

• UR3 — Contact:

* { length(Contact) = 11 }

**2. Vehicle Registration**

• VR1 — License Plate:

* { matches /^[A-Z]{3}\d{4}$/ (e.g. “ABC1234”) }

**3. Booking a spot**

• PS1 — Search Date:

* { date ∈ [today, today + 365 days] }

• PS2 — Start Time:

* { time ∈ {00:00, 00:15, …, 23:45} }

• PS3 — End Time:

* { endTime > startTime }

**4. Admin Parking‐Lot Setup**

• AS1 — Spot Count:

* { 1 ≤ numberOfSpots }

• AS2 — Type:

* { 1 ≤ length(Type) AND matches [A-Z]+ | [a-z]+ }

• AS3 — Hourly Rate:

* { matches [0-9]+ | [0-9]+ }

**5. Feedback**

• FB1 — Rating:

* { integer ∈ {1,2,3,4,5} }

• FB2 — Comment:

* { length(comment) ≤ 256 characters }

1. **User Registration (6 TCs)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TC | Name | Username | Password | Contact | Expected Result |
| UR-1 | "A" | "user1" | "abcde" | "03123456789" | Pass: Account created |
| UR-2 | "" | "user1" | "abcde" | "03123456789" | Fail: "Name must be at least 1 character" |
| UR-3 | "John123" | "johnD" | "abcde" | "03123456789" | Fail: "Name must contain only letters" |
| UR-4 | "Alice" | "" | "abcde" | "03123456789" | Fail: "Username must be at least 1 character" |
| UR-5 | "Bob" | "bob42" | "1234" | "03123456789" | Fail: "Password must be at least 5 characters" |
| UR-6 | "Carol" | "carolX" | "abcde" | "0312345678" | Fail: "Contact must be exactly 11 digits" |

1. **Vehicle Registration (3 TCs)**

|  |  |  |
| --- | --- | --- |
| TC | License Plate | Expected Result |
| VR-1 | "ABC1234" | Pass: Vehicle added |
| VR-2 | "abc1234" | Fail: "License must be 3 uppercase letters + 4 digits" |
| VR-3 | "ABCD123" | Fail: "License must match format ABC1234" |

1. **Booking a Spot (6 TCs)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TC | Search Date | Start Time | End Time | Expected Result |
| PS-1 | Apr 26 2025 | 09:00 | 10:00 | Pass: Returns available spots |
| PS-2 | Apr 27 2026 | 10:00 | 11:00 | Fail: "Date must be within 365 days from today" |
| PS-3 | Apr 25 2025 | 09:00 | 10:00 | Fail: "Date cannot be before today" |
| PS-4 | May 1 2025 | 09:10 | 10:10 | Fail: "Start time must be on 15-minute increments" |
| PS-5 | May 1 2025 | 10:00 | 09:45 | Fail: "End time must be after start time" |
| PS-6 | Apr 26 2026 | 23:30 | 23:45 | Pass: Edge of range (365 days ahead) |

1. **Feedback (5 TCs)**

|  |  |  |  |
| --- | --- | --- | --- |
| TC | Rating | Comment (Length) | Expected Result |
| FB-1 | 1 | "Good." (5) | Pass: Feedback saved, average rating updated |
| FB-2 | 5 | 256 chars | Pass: Feedback saved |
| FB-3 | 0 | "Bad" (3) | Fail: "Rating must be between 1 and 5" |
| FB-4 | 6 | "OK" (2) | Fail: "Rating must be between 1 and 5" |
| FB-5 | 3 | 300 chars | Fail: "Comment must be ≤256 characters" |

1. **Admin Parking Setup (5 TCs)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TC | Spot Count | Type | Hourly Rate | Expected Result |
| AS-1 | 1 | "Covered" | "50" | Pass: Configuration saved |
| AS-2 | 0 | "Covered" | "50" | Fail: "Spot count must be ≥1" |
| AS-3 | 10 | "A1" | "50" | Fail: "Type must contain only letters" |
| AS-4 | 10 | "Open" | "5.5" | Fail: "Rate must be integer digits only" |
| AS-5 | 500 | "Handicap" | "100" | Pass: Configuration saved |

**Boundary Value Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Area | Input/Test Case | Expected Result |
| 1 | **UR1 - Name** | **Length 0** | **Fail (must be ≥1 character)** |
| 2 | **UR1 - Name** | **Length 1** | **Pass** |
| 3 | **UR1 - Name** | **Length 2** | **Pass** |
| 4 | **UR2 - Username** | **Length 0** | **Fail (must be ≥1 character)** |
| 5 | **UR2 - Username** | **Length 1** | **Pass** |
| 6 | **UR2 - Username** | **Length 2** | **Pass** |
| 7 | **UR3 - Password** | **Length 4** | **Fail (must be ≥5 characters)** |
| 8 | **UR3 - Password** | **Length 5** | **Pass** |
| 9 | **UR3 - Password** | **Length 6** | **Pass** |
| 10 | **UR4 - Contact** | **Length 10** | **Fail (must be exactly 11 digits)** |
| 11 | **UR4 - Contact** | **Length 11** | **Pass** |
| 12 | **UR4 - Contact** | **Length 12** | **Fail (must be exactly 11 digits)** |
| 13 | **VR1 - License Plate** | **Length 6** | **Fail (must be 7 chars: 3 letters + 4 digits)** |
| 14 | **VR1 - License Plate** | **Length 7** | **Pass** |
| 15 | **VR1 - License Plate** | **Length 8** | **Fail** |
| 16 | **PS1 - Search Date** | **Today -1 day** | **Fail** |
| 17 | **PS1 - Search Date** | **Today** | **Pass** |
| 18 | **PS1 - Search Date** | **Today +1 day** | **Pass** |
| 19 | **PS1 - Search Date** | **Today +365 days** | **Pass** |
| 20 | **PS1 - Search Date** | **Today +366 days** | **Fail** |
| 21 | **PS2 - Start Time** | **00:00** | **Pass** |
| 22 | **PS2 - Start Time** | **00:01** | **Fail (must be 15-min increment)** |
| 23 | **PS2 - Start Time** | **00:15** | **Pass** |
| 24 | **PS2 - Start Time** | **23:45** | **Pass** |
| 25 | **PS2 - Start Time** | **23:59** | **Fail** |
| 26 | **PS3 - End Time** | **Start 10:00, End 10:00** | **Fail (end must be > start)** |
| 27 | **PS3 - End Time** | **Start 10:00, End 10:15** | **Pass** |
| 28 | **AS1 - Spot Count** | **0** | **Fail** |
| 29 | **AS1 - Spot Count** | **1** | **Pass** |
| 30 | **AS1 - Spot Count** | **2** | **Pass** |
| 31 | **AS2 - Type** | **Length 0** | **Fail** |
| 32 | **AS2 - Type** | **Length 1** | **Pass** |
| 33 | **AS2 - Type** | **Length 2** | **Pass** |
| 34 | **AS3 - Hourly Rate** | **Empty ""** | **Fail** |
| 35 | **AS3 - Hourly Rate** | **"5"** | **Pass** |
| 36 | **AS3 - Hourly Rate** | **"50"** | **Pass** |
| 37 | **FB1 - Rating** | **0** | **Fail (must be between 1–5)** |
| 38 | **FB1 - Rating** | **1** | **Pass** |
| 39 | **FB1 - Rating** | **5** | **Pass** |
| 40 | **FB1 - Rating** | **6** | **Fail** |
| 41 | **FB2 - Comment** | **Length 0** | **Pass** |
| 42 | **FB2 - Comment** | **Length 255** | **Pass** |
| 43 | **FB2 - Comment** | **Length 256** | **Pass** |
| 44 | **FB2 - Comment** | **Length 257** | **Fail** |

# 

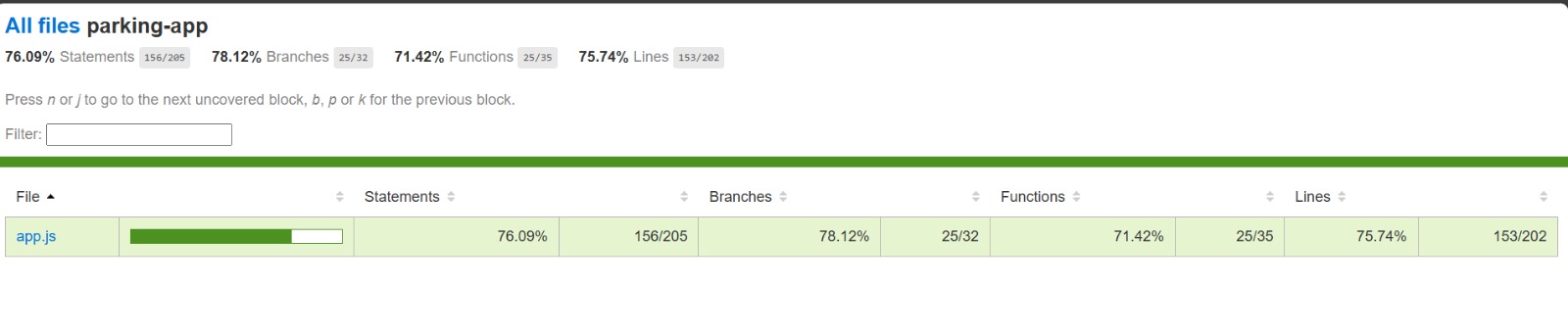
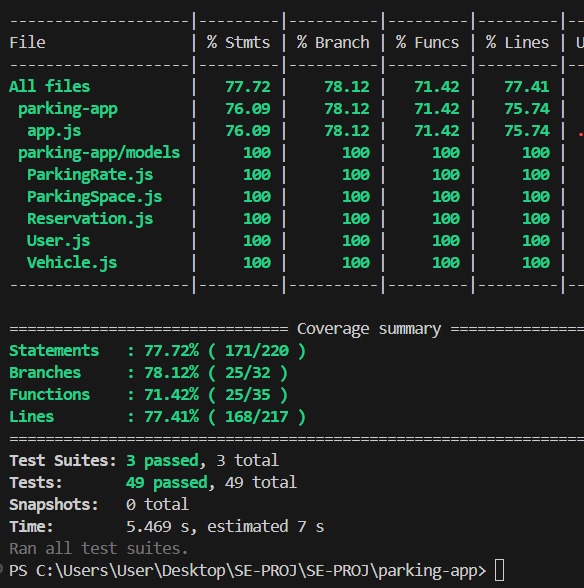
# ***12. White Box Testing***

**Covered well**

* **Core route handlers:** all major POST/GET endpoints (admin & owner login, registration, vehicle & parking‐space creation, booking flow, password change, profile & reservations data, reports, reminder, logout) have both success and error branches exercised.
* **Business logic branches:** invalid credentials, missing fields, not‐found cases, and internal‐error branches via mocked throws are all hit.
* **Session flows:** use of Supertest agents ensures session persists across requests for owner/admin scenarios.

**Not Covered**

* **Static-only middleware and asset delivery (e.g. serving CSS/JS from public**/): we treat those as infrastructure—no need to unit-test every static file.
* **Third-party modules & deep Mongoose schema logic**: we mock Mongoose models to isolate the app code.
* Any dead/error‐logging code paths that log to console but don’t affect response—these are generally untestable without significant instrumentation and are lower priority.



# ***13. Work Division***

Our group consisted of **three members**, and we decided to **divide the work evenly**. Instead of assigning fixed modules to each member, **everyone contributed collaboratively** across all stages of the project. The contributions were as follows:

* **Requirement Gathering and Analysis**: All members collectively gathered requirements, created user stories, and prepared the Software Requirements Specification (SRS).
* **Sprint Coding**: Sprints 1 and 2 were coded Collectively while Sprint 3 was coded by Hamza and Munhim.
* **Project Plan and System Architecture**: Designed by Umer.
* **System Design**: Umer worked on designing system architecture diagrams (ER diagram, Use Case, Sequence, Package, Deployment, Component diagrams).
* **Frontend and Backend Development**: All group members participated in both frontend (Mostly by Munhim) (HTML/CSS/JS) and backend (Node.js, Express.js) development, ensuring that each person understood the full application stack.
* **Database Setup**: Setting up MongoDB schema, data validations, and optimizations were done jointly.
* **Testing and Debugging**: Blackbox testing was done collectively while Whitebox testing was done by Umer.
* **Documentation and Reporting**: The final project documentation was compiled by Munhim, including screenshots, sprint backlog (1 by Umer, 2 by Munhim), and burn-down charts, was prepared collectively. The PowerPoint Slides were made by Hamza.

We followed **agile practices** by conducting **google meet meetings** to discuss progress and reassign any pending or blocking tasks dynamically among ourselves.

# ***14. Lessons Learnt***

Throughout the project, our group gained significant insights and valuable lessons:

* **Importance of Proper Planning**: We realized that a well-defined plan (SRS and Project Plan) saved a lot of time during development.
* **Team Coordination Matters**: Regular communication and updates among team members helped prevent misalignment and task overlaps.
* **Version Control is Essential**: Using Git allowed us to manage code collaboratively without losing work, and merge conflicts taught us to be more careful while editing shared files.
* **Database Design Impact**: We learned that a good database schema (well-indexed, validated) improves performance, while poor design leads to bottlenecks.
* **Handling Unexpected Errors**: Practical development brought unforeseen issues (like integration errors, backend crashes), teaching us to think from both happy paths and edge cases.
* **Adaptability**: Sometimes tasks took longer than expected. Being flexible and quickly adapting to changes was crucial for project continuity.
* **Documentation Value**: Writing clear documentation alongside development helped in faster onboarding and easier debugging.
* **Agile Approach Benefits**: Sprint-based development kept our project organized, and short iterations helped in consistently delivering functionality and quickly adapting feedback.
* **Understanding Full Stack Development**: Since each member contributed to both frontend and backend, it enhanced our holistic understanding of the system.
* **Real-World Practices**: Security requirements, session handling, exposed us to practical industrial practices often missing from theoretical learning.

In conclusion, the project not only enhanced our technical skills but also strengthened our teamwork, problem-solving abilities, and adaptability to real-world software development challenges.