Final Report: EasyParking Management System

**1. Project Introduction**

The Vehicle Parking Management System is a web-based application designed to simplify and automate the process of finding, reserving, and paying for parking spaces. It caters to both users and administrators, offering functionality such as reservation, slot management, pricing adjustments, and real-time availability.

**2. Functional and Non-Functional Requirements**

Functional Requirements

- User registration and login  
- Parking reservation  
- Slot selection based on vehicle type  
- Online payments and invoice generation  
- Real-time slot availability  
- Review and complaint system  
- Admin management of users, slots, bookings, pricing  
- Reports and promotional discounts

Non-Functional Requirements

- Concurrent user handling  
- Fast response times (<3 seconds)  
- Encrypted user data  
- Role-based access control (RBAC)  
- Responsive UI  
- Maintainable and well-documented code  
- User manual and 24/7 support

**3. User Stories**

* **Parking Reservation User Role**: A registered user

**Goal:** Reserve a parking spot online.

**Reason:** Ensures a parking space before arriving.

**Pre-conditions:** The user must be logged into the system.

Parking slots must be available.

**Post-conditions:** The parking slot is reserved for the selected time.

The user receives a booking confirmation.

* **Checking Parking Availability User Role:** A general user

**Goal:** Check available parking spaces in different locations.

**Reason:** Helps users find a suitable spot before making a reservation.

**Pre-conditions:** The system must have updated parking slot availability.

**Post-conditions:** The user is shown real-time parking slot availability.

The user can proceed with a reservation.

* **Payment for Parking User Role:** A registered user **Goal**: Pay for parking online.

**Reason:** Enables a cashless and convenient payment process.

**Pre-conditions:** The user must have a reserved parking slot.

A valid payment method must be available.

**Post-conditions:** Payment is successfully processed.

A payment receipt is generated and sent to the user.

* **Selecting Parking Spot Type User Role**: A general user

**Goal:** Choose a parking spot based on vehicle type.

**Reason:** Ensures parking in an appropriate space.

**Pre-conditions:** The system must provide different parking spot categories.

The user must select their vehicle type.

**Post-conditions:** The user is allocated a suitable parking spot, especially for people with specific parking requirements including disabled person's car parking slots.

The parking spot is reserved if the user proceeds with a booking.

* **Setting Parking Duration User Role:** A registered user

**Goal:** Specify how long to park.

**Reason:** Ensures correct billing and availability for others.

**Pre-conditions:** The user must select a parking spot.

**Post-conditions:** The system calculates the total parking cost.

The parking spot is reserved for the chosen duration.

* **Receiving Parking Expiry Notifications User Role:** A registered user

**Goal**: Get reminders when parking time is about to expire.

**Reason:** Allows users to extend or vacate on time.

**Pre-conditions:** The user must have an active parking session.

**Post-conditions:** The user receives timely notifications before expiry.

The user can choose to extend the parking time or vacate.

* **Cancelling a Reservation User Role:** A registered user

**Goal:** Cancel a parking reservation.

**Reason:** Allows flexibility in case plans change.

**Pre-conditions:** The user must have an active parking reservation.

The ticket must have been canceled at least 1 hour ago. Otherwise, the amount won’t be refunded to the user's wallet.

**Post-conditions:** The reservation is canceled, and the slot is made available for others. If applicable, the system processes a refund.

* **Viewing My Parking History User Role:** A registered user

**Goal:** View past parking bookings.

**Reason:** Helps track expenses and parking usage.

**Pre-conditions:** The user must have completed at least one parking session.

**Post-conditions:** The system displays a history of previous bookings and payments.

* **Checking Reviews and Ratings User Role:** A general user

**Goal:** Read user reviews of parking facilities.

**Reason:** Helps in selecting a reliable and safe parking spot.

**Pre-conditions:** Reviews must be available in the system.

**Post-conditions:** The user can make an informed decision based on reviews.

* **Managing Parking Slots User Role**: Admin

**Goal:** Add, update, and remove parking slots.

**Reason:** Keeps parking availability accurate.

**Pre-conditions:** The system must have a database of parking slots.

**Post-conditions:** The updated slot availability is reflected in the system.

* **Managing Bookings User Role:** Admin

**Goal:** View and manage all parking reservations.

**Reason:** Ensures smooth system functionality.

**Pre-conditions:** The system must have booking records.

**Post-conditions:** The admin can modify or cancel reservations as needed.

* **Handling User Complaints User Role:** Admin

**Goal:** Review and respond to user complaints.

**Reason:** Ensures customer satisfaction by addressing issues.

**Pre-conditions:** Users must be able to submit complaints through the system.

The system must record complaints with relevant details.

**Post-conditions:** The admin can view and track complaints.

The user receives a response or resolution notification.

* **Adjusting Pricing User Role:** Admin

**Goal:** Set different parking rates based on time, location, and demand.

**Reason:** Helps manage peak and off-peak pricing effectively.

**Pre-conditions:** The system must allow dynamic pricing updates.

There must be predefined rules for pricing changes.

**Post-conditions:** The updated pricing is reflected in the booking system.

Users see accurate pricing before making a reservation.

* **Generating Reports User Role:** Admin

**Goal:** Generate reports on parking usage, revenue, and occupancy trends.

**Reason:** Provides insights for better decision-making.

**Pre-conditions:** The system must collect and store transaction and usage data.

**Post-conditions:** The admin receives detailed analytical reports.

Reports help optimize parking space management and pricing strategies.

* **Creating Discounts and Promotions User Role:** Admin

**Goal:** Offer promotional discounts and special deals.

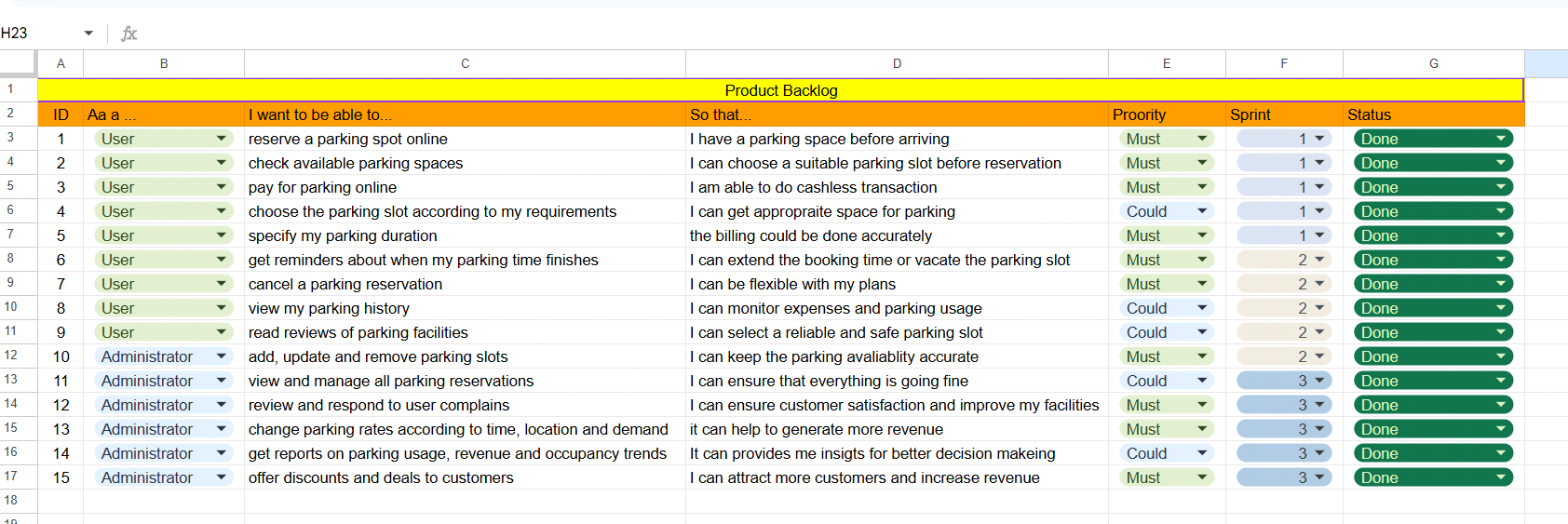
**Reason:** Attracts more users and increases bookings.

**Pre-conditions:** The system must support discount codes and promotional offers.

**Post-conditions:** Users see available discounts during booking.

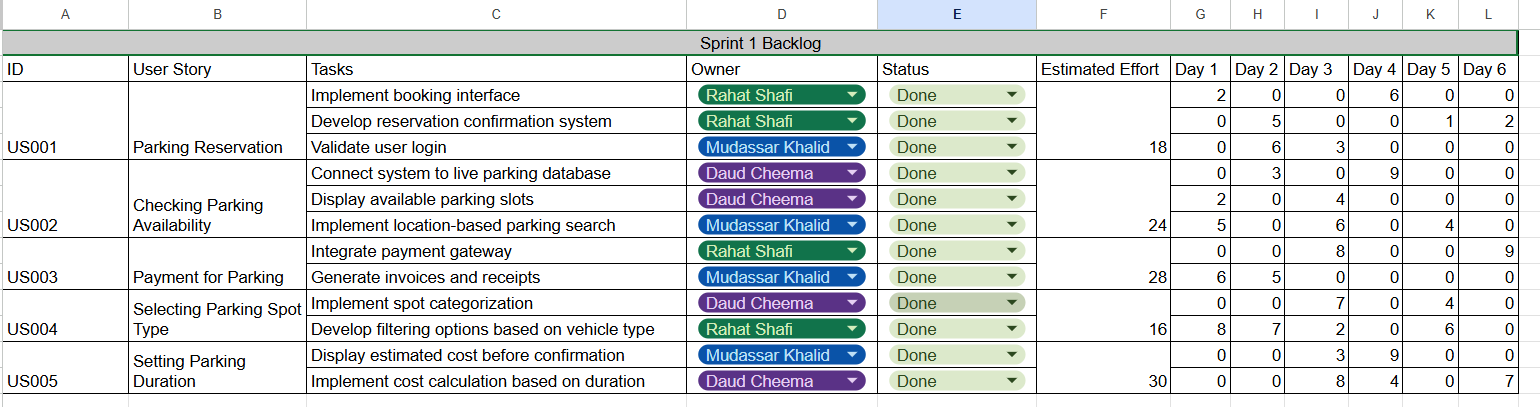
Discounted rates are applied automatically based on eligibility.

**4. Product Backlog**



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

Sprint 1

****

Sprint 2

A screenshot of a computer

AI-generated content may be incorrect.

**6. Project Plan**

**Phase 1: Requirements and Design**

**1.1 Requirement Gathering**

* Stakeholder meetings to collect functional and non-functional requirements
* Documenting user stories, pre- and post-conditions (Refer: Report, User Stories)

**1.2 System Design**

* Define system architecture and database schema
* Create UML diagrams:
  + Use Case Diagram
  + Sequence Diagrams (Reservation, Payment, Cancellation)
  + Class Diagram

**1.3 Sprint Planning & Retrospectives**

* Conduct regular sprint planning meetings
* End each sprint with a retrospective to improve team efficiency

**1.4 Documentation**

* Create and maintain:
  + Software Requirements Specification (SRS)
  + Architecture design documentation
  + Diagrams and report deliverables

**1.5 Progress Tracking**

* Tools Used: Trello (for task management), GitHub Projects (for code progress)
* Regular updates to track task completion and bug resolution

**Phase 1: Development and Testing**

**2.1 Frontend Development (React + Vite)**

* **UI/UX Design**
  + Design user-friendly interfaces
  + Ensure responsive design for mobile and desktop
* **Component Development**
  + Implement pages for login, reservation, payment, reviews, etc.
  + Integrate Google Maps API for location-based parking slots

**2.2 Backend Development (ASP.NET Web API)**

* Implement RESTful APIs using generic repository pattern
* Handle authentication, role management, parking logic, payments, and complaints

**2.3 Integration**

* Connect frontend and backend via API endpoints

**2.4 Testing**

* Perform Black Box and White Box testing
* Ensure at least 70%+ unit test coverage
* Include test cases for all core modules

**Phase 3: Deployment and Review**

**3.1 Deployment**

* Deploy on a cloud platform (e.g., Azure, AWS)
* Ensure HTTPS and database hosting is secure and scalable

**3.2 User Acceptance Testing**

* Test the system with actual users
* Gather feedback from testers

**3.3 Final Bug Fixes**

* Address feedback and resolve all open issues

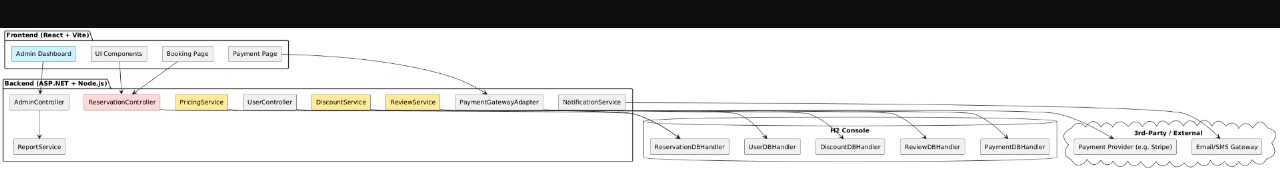
**3.4 Final Deliverables**

* Submit:
  + Final Report
  + User & Admin Manuals
  + System Documentation
  + Trello & GitHub snapshots

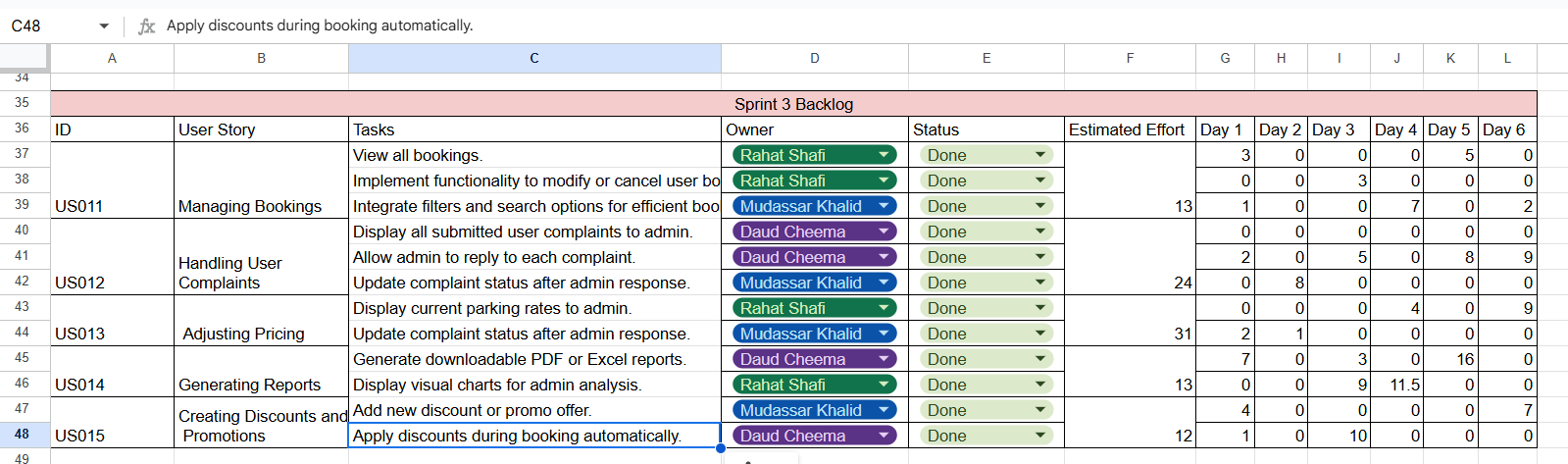
**7. Architecture Diagram**

A diagram of a server

AI-generated content may be incorrect.

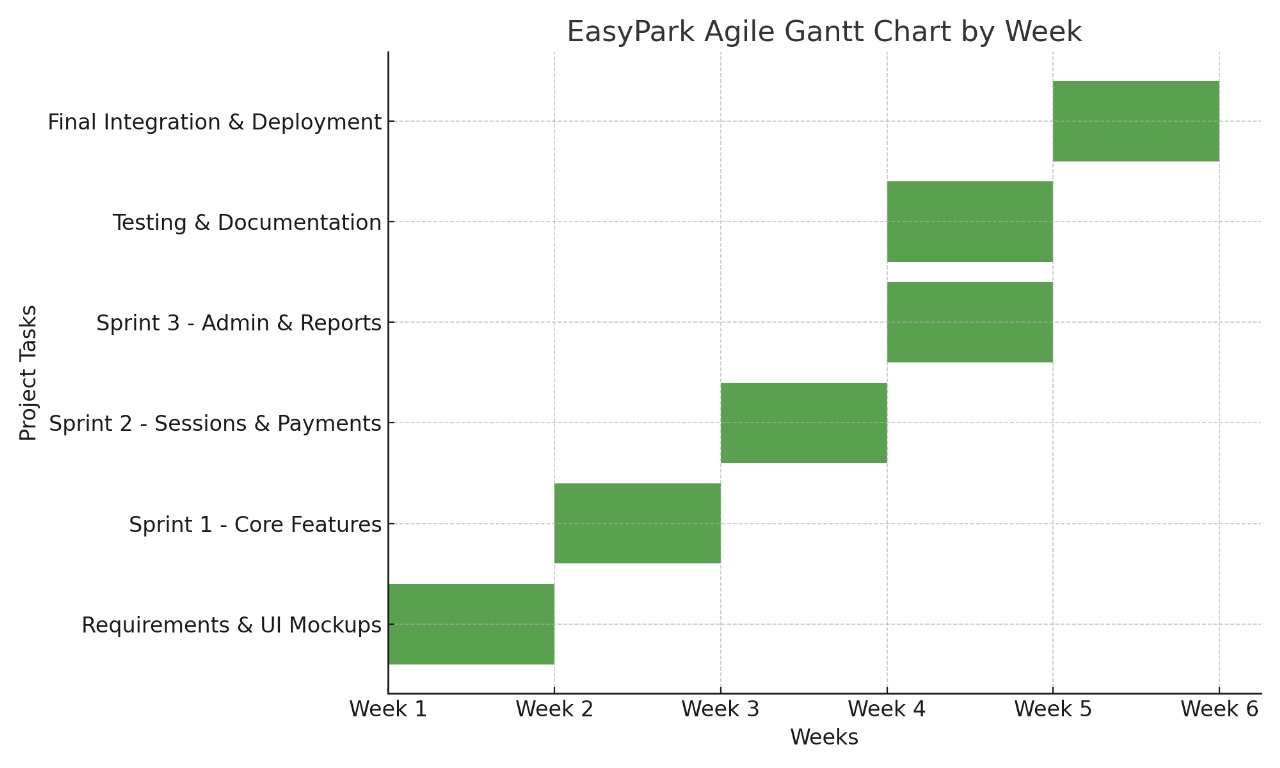


**8. Design(All sprint 3 diagrams)**

****

**9. Actual Implementation Screenshot**

**10. Product Burn down chart for the project**



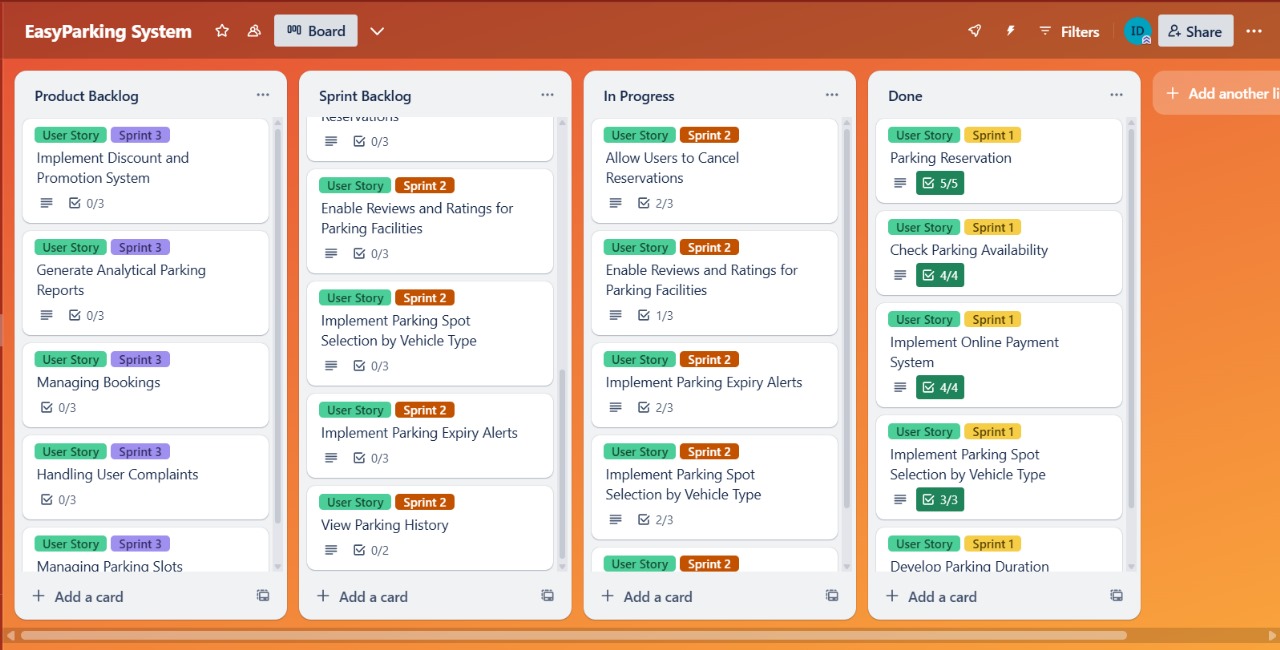
**11. Trello board screen shots**

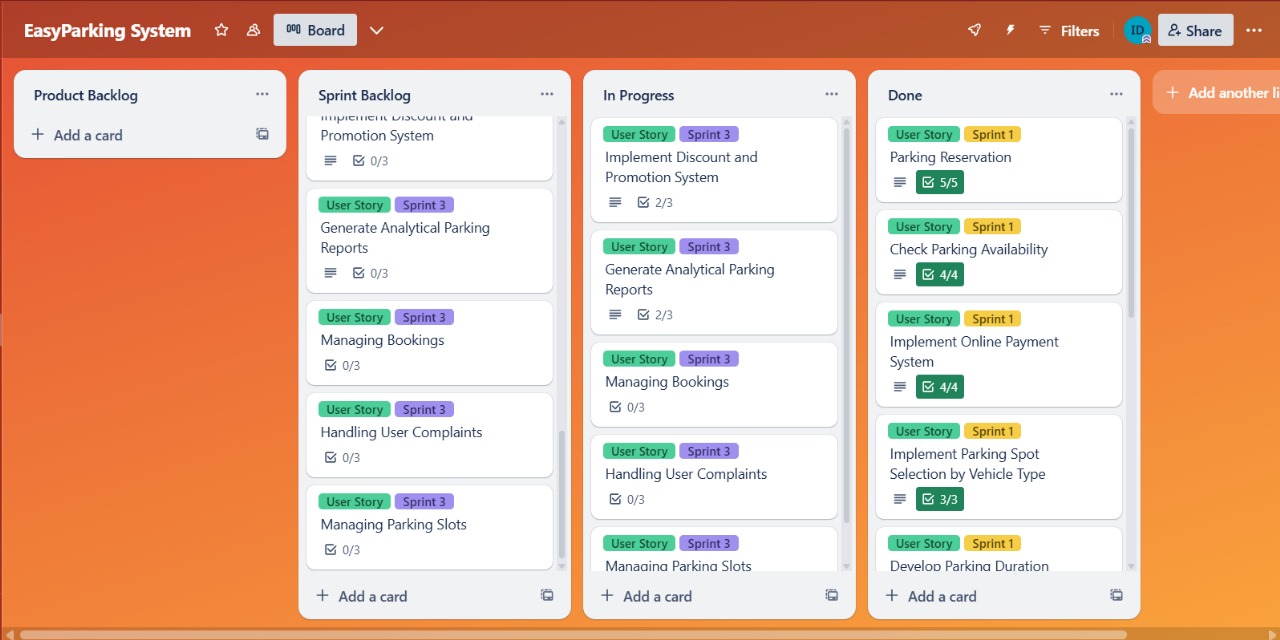
A screenshot of a computer

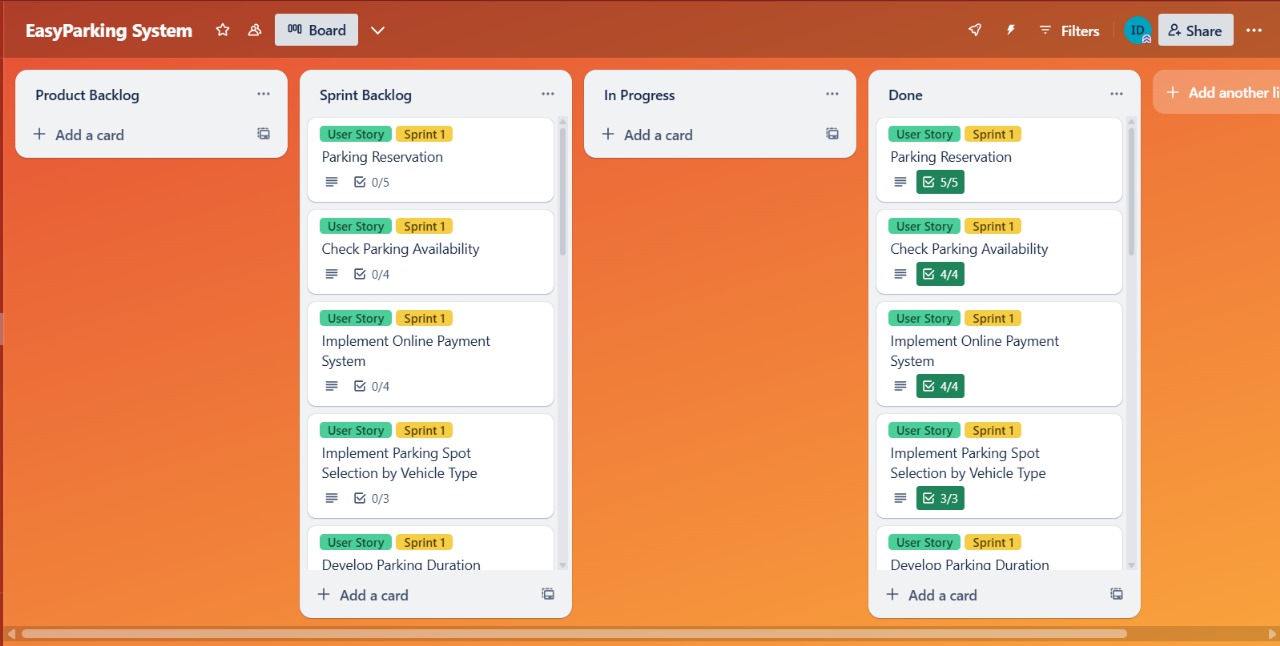
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.







**12. Testcases -Black box**

**Equivalence Class Partitioning**

| **Test Case ID** | **Description** | **Input** | **Expected Output** |
| --- | --- | --- | --- |
| TC1 | Register user with valid data | name, valid email, strong password, role=user | Success, redirected to login |
| TC2 | Register with invalid email | name, *invalid email*, password, role=user | Error: Invalid email |
| TC3 | Register with empty password | name, email, "", role=user | Error: Password required |
| TC4 | Login with incorrect password | email, *wrong password*, role=user | Error: Invalid credentials |
| TC5 | Book parking with valid data | user, slotId, timeDuration | Success, booking confirmed |
| TC6 | Book parking without selecting slot | user, *empty slotId*, timeDuration | Error: Slot selection required |
| TC7 | Cancel existing reservation | user, bookingId | Success, booking cancelled |
| TC8 | Cancel non-existing reservation | user, *invalid bookingId* | Error: Reservation not found |
| TC9 | Admin login with correct credentials | adminEmail, correct password | Success, redirected to admin dashboard |
| TC10 | Admin deletes parking slot with invalid ID | admin, *invalid slotId* | Error: Slot not found |

**Boundary Value Analysis**

| **Test Case ID** | **Description** | **Input** | **Expected Output** |
| --- | --- | --- | --- |
| TC11 | Register with 6-character password (valid) | name, email, "123456", role=user | Success |
| TC12 | Register with 5-character password (invalid) | name, email, "12345", role=user | Error: Password too short |
| TC13 | Select parking duration as 0 minutes | slotId, *0 minutes* | Error: Invalid parking duration |
| TC14 | Select parking duration as 1 minute (minimum valid) | slotId, *1 minute* | Success |
| TC15 | Book with max character length vehicle number | valid data, *max-length vehicle number string* | Success or Error (based on limit validation) |

**System Functionality Validation**

All of the above test cases were executed to validate the following workflows:

* User Registration and Login
* Admin Login
* Parking Slot Booking, Cancellation, Viewing
* Authentication & Input Validation
* Boundary scenarios (password, duration)

**13. Testcases -White box**

**14. Work Division between group members**

**1. Mudassar Khalid**

* **Role**: Frontend Developer (React or HTML/CSS/JS stack)
* **Responsibilities**:
  + User Interface (UI) design for user-facing modules
  + Implementing login, registration, and dashboard screens
  + Parking slot selection and booking interface
  + Review submission and parking history components
  + Compiled user stories, test results, and final report

**2. Muhammad Rahat Shafi**

* **Role**: Backend Developer (Node.js / ASP.NET / Django etc.)
* **Responsibilities**:
  + API development for booking, payment, and slot management
  + Integration of payment gateway
  + Role-based authentication and authorization
  + Parking availability logic and dynamic pricing
  + Frontend integration with backend APIs
  + Admin panel backend: reservation reports, discount control
  + Database schema design and optimization

**3. Muhammad Daud Cheema**

* **Role**: Testing & Project Coordination
* **Responsibilities**:
  + Developed and executed **black-box** and **white-box** test cases
  + Deployed system on local/online servers
  + Trello board management and sprint tracking
  + Verified functional/non-functional requirements implementation
  + Ensured overall project coordination and documentation formatting
  + Contributed to documentation and user manuals

**15. Lesson learnt by group**

* **Importance of Clear Requirements**  
  At the beginning of the project, ambiguities in requirements caused delays. We learned that clear, detailed requirements gathering is crucial to prevent miscommunication and scope creep.
* **Effective Task Distribution**  
  Dividing responsibilities based on individual strengths (frontend, backend, testing) significantly improved productivity and project quality.
* **Agile Methodology Works Well**  
  Adopting sprints and regular team meetings helped us track progress, prioritize features, and stay adaptable to changes.
* **Communication is Key**  
  Regular updates via group chats and task tracking (Trello) ensured everyone was aligned and bottlenecks were identified early.
* **Testing Should Start Early**  
  We learned that beginning both black-box and white-box testing during development (rather than after) helps catch bugs early and reduces last-minute stress.
* **User-Centric Design Matters**  
  Realizing that even the most functional systems can be frustrating without a good UI encouraged us to focus on usability and responsive design.
* **Integration Challenges Are Real**  
  Frontend and backend integration had issues due to mismatched expectations. We learned the importance of consistent API contracts and early testing across modules.
* **Documentation Saves Time**  
  Well-maintained documentation helped us onboard members quickly, ease handovers, and complete the final report with less friction.
* **Dealing with Real-Time Data is Complex**  
  Managing real-time parking availability and concurrency taught us the challenges of data synchronization and transaction management.
* **Team Collaboration Builds Soft Skills**  
  Beyond technical learning, we improved communication, teamwork, and time management—essential skills for real-world software development.