VEHICLE PRE-BOOKING SYSTEM

A PROJECT REPORT

Submitted by

SRIRAM S	**2303811710421158**
VIJAYA PRAKASH G	**2303811710421176**
VIJAYESWARAN V S	**2303811710421177**
VIJESH S S	**2303811710421178**
YUVARAJ P	**2303811710421188**

in partial fulfillment of the requirements for the award degree of

Bachelor in Engineering

CSB1303 – OBJECT ORIENTED ANALYSIS AND DESIGN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

DECEMBER - 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM - 621112

BONAFIDE CERTIFICATE

The work embodied in the present project report entitled "**VEHICLE PRE-BOOKING SYSTEM**" has been carried out by the students **SRIRAM S, VIJAYA PRAKASH G, VIJAYESWARAN V S, VIJESH S S, YUVARAJ P**. The work reported here in is original and we declare that the project is their own work, except where specifically acknowledged, and has not been copied from other sources or been previously submitted for assessment.

Date of Viva Voce: ……

Mrs.V. KALPANA M.E., (Ph.D.,)
SUPERVISOR
Assistant Professor
Department of CSE
K. Ramakrishnan College of Technology (Autonomous)
Samayapuram – 621 112.

Mr. R. RAJAVARMAN M.E., (Ph.D.,)
HEAD OF THE DEPARTMENT
Assistant Professor (Sr. Grade)
Department of CSE
K. Ramakrishnan College of Technology (Autonomous)
Samayapuram – 621 112.

INTERNAL EXAMINER **EXTERNAL EXAMNIER**

ABSTRACT

The Vehicle Pre-Booking System is a comprehensive web-based application developed to revolutionize the process of reserving vehicles for rental, pre-booking new vehicles from showrooms, and booking curated travel packages. Built with React 19 and TypeScript, the system provides users with an intuitive interface to browse available vehicles, filter by type and category, and make secure bookings with real-time price calculation. The platform incorporates an AI-powered travel assistant leveraging Google Gemini API to offer personalized trip planning and vehicle recommendations based on natural language queries. For administrators, the system includes a sophisticated dashboard with analytics, revenue tracking, and booking management capabilities. The application features a responsive design with dark/light mode support, secure user authentication with role-based access control, and a seamless booking workflow. By integrating multiple vehicle-related services into a single platform, the system enhances user convenience while providing businesses with efficient management tools, ultimately modernizing the vehicle booking experience through technology and artificial intelligence.

Keywords: Vehicle Pre-Booking System, web-based application, vehicle rental automation, digital booking, real-time pricing, admin dashboard, React frontend, TypeScript, Gemini AI integration, secure authentication, showroom pre-booking, travel packages, AI trip planner, dynamic pricing.

ACKNOWLEDGEMENT

We thank our **Dr. N. Vasudevan**, Principal, for his valuable suggestions and support during the course of my research work.

We thank our **Mr. R. Rajavarman**, Head of the Department, Assistant Professor (Sr. Grade), Department of Computer Science and Engineering, for his valuable suggestions and support during the course of my research work.

We wish to record my deep sense of gratitude and profound thanks to my Guide **Mrs. V. KALPANA,** Assistant Professor , Department of Computer Science and Engineering**,** for her keen interest, inspiring guidance, constant encouragement with my work during all stages, to bring this thesis into fruition.

We are extremely indebted to our project coordinator **Mrs. V. KALPANA,** Assistant Professor, Department of Computer Science and Engineering, for her valuable suggestions and support during the course of my research work.

We also thank the faculty and non-teaching staff members of the Department of Computer Science And Engineering, K.Ramakrishnan College Of Technology, Samayapuram, for their valuable support throughout the course of my research work.

Finally, we thank our parents, friends and our well wishes for their kind support.

SIGNATURE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	-	Application Programming Interface
CDN	-	Content Delivery Network
JSON	-	JavaScript Object Notation
DOM	-	Document Object Model
HMR	-	Hot Module Replacement
CLI	-	Command Line Interface
UI	-	User Interface
UX	-	User Experience
CSS	-	Cascading Style Sheets
JS	-	JavaScript
TS	-	TypeScript
AI	-	Artificial Intelligence

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION ABOUT DOMAIN

The domain of vehicle pre-booking and travel management encompasses services that allow users to rent various types of vehicles for short-term or long-term use, pre-book new vehicles directly from showrooms, and purchase curated travel and tour packages. This industry is characterized by dynamic pricing, real-time availability checks, customer preference matching, and transaction management. With the rise of digital platforms, users expect seamless online experiences that integrate search, booking, payment, and post-booking management. Additionally, the integration of artificial intelligence for personalized recommendations and itinerary planning represents a significant advancement in enhancing user engagement and satisfaction. The Vehicle Pre-Booking System operates within this domain, aiming to consolidate these fragmented services into a unified, intelligent platform.

1.2 PROBLEM DESCRIPTION

The current landscape for vehicle rentals and travel bookings is often siloed. Customers typically use separate platforms for car rentals, showroom inquiries, and holiday packages, leading to a fragmented and time-consuming experience. Key challenges include: The current system faces several challenges, including the lack of an integrated platform that combines vehicle rentals, showroom pre-booking, and tour packages in one place. Most processes are still manual and inefficient, as users must search through limited options without advanced filtering or intelligent suggestions. There is also no AI-powered assistance to help users with complex trip planning queries, such as recommending the best vehicle for specific travel conditions. The Vehicle Pre-Booking System allows users to reserve vehicles in advance, ensuring availability when needed. It manages vehicle inventory, booking schedules, and user information efficiently.

1.3 OBJECTIVE OF THE PROJECT

The primary objective is to develop a unified, secure, and intelligent vehicle pre-booking system that: Provides a seamless interface for renting vehicles, pre-booking showroom models, and booking travel packages. Implements a robust user authentication and authorization system with distinct roles for customers and administrators. Integrates an AI-powered assistant (using Google Gemini API) to provide intelligent travel advice and vehicle recommendations. Features a dynamic booking engine that calculates prices in real-time based on rental duration, vehicle type, and booking category. Includes a comprehensive admin dashboard for monitoring bookings, revenue analytics, and system performance.

1.4 SCOPE OF THE PROJECT

The proposed system includes a modern single-page application (SPA) developed using React 19, TypeScript, and Tailwind CSS for a fast and responsive user experience. It supports complete user management, enabling registration, login, and profile handling for both customers and administrators. The platform provides robust inventory management, showcasing rental vehicles, showroom models, and travel packages with advanced filtering and search capabilities. A comprehensive booking workflow allows users to select items, choose rental dates, view dynamic price calculations, and complete payments. AI integration through the Google Gemini API enhances the system by offering intelligent travel advice and vehicle recommendations based on user queries. Administrators benefit from a dedicated dashboard featuring charts, statistics, and a transaction log for monitoring platform performance. The entire interface is fully responsive across desktop, tablet, and mobile devices, while client-side state management is efficiently handled using React hooks along with in-memory structures such as Maps and Arrays. The system enables users to book vehicles in advance, ensuring convenience and efficient resource utilization. It tracks vehicle availability, booking history, and user details in real time. Admins can manage the fleet, approve or cancel bookings, and generate reports. The system enhances operational efficiency, reduces waiting time, and provides secure and reliable booking services

CHAPTER 2
SYSTEM REQUIREMENT SPECIFICATIONS (SRS)

2.1 FUNCTIONAL REQUIREMENTS

2.1.1 User Authentication:

The system shall allow new users to register with email, password, name, and role (User/Admin).The system shall allow registered users to log in using email and password.The system shall enforce unique email addresses during registration.The system shall maintain user session after successful login.

2.1.2 Vehicle and Package Browsing:

The system shall display a list of available rental vehicles with details (name, type, price per day, image).The system shall display a list of showroom vehicles available for pre-booking with details (name, type, booking fee, full price).The system shall display a list of travel packages with details (title, location, duration, price, highlights).The system shall allow users to filter vehicles by type (Car, SUV, Bike, Luxury, etc.) and category (Rental, Showroom).

2.1.3 Booking Management:

The system shall allow logged-in users to book a rental vehicle by selecting start and end dates.The system shall calculate the total price for rentals dynamically (price per day × number of days).The system shall allow logged-in users to pre-book showroom vehicles by paying a fixed booking fee.The system shall allow logged-in users to book travel packages at a fixed price.The system shall prompt unauthenticated users to log in when attempting to book. It ensures real-time tracking of available vehicles, prevents double bookings, and maintains a record of all transactions. Admins can approve, reject, or reschedule bookings and generate reports for better fleet utilization.

2.1.4 AI Trip Planner:

The system shall provide an AI-powered trip planning feature that allows users to enter natural language queries for personalized travel assistance. User queries will be processed and forwarded to the Google Gemini API, which will generate detailed trip plans including route suggestions, must-visit places, and helpful travel tips. Along with trip planning, the AI will recommend the most suitable vehicles based on factors such as terrain, distance, passenger count, and travel purpose. This enables users to easily identify the ideal rental or showroom vehicle for their journey. The system will also support complex queries like “best car for a mountain trip with family” or “cheapest package for a weekend tour,” ensuring accurate and relevant suggestions. All AI-generated insights will be displayed seamlessly within the interface and integrated directly into the booking workflow, allowing users to proceed with vehicle or package bookings instantly based on the recommendations.

2.1.5 Admin Dashboard:

The system shall provide a dedicated dashboard exclusively for users with the 'Admin' role, offering a centralized view of all operational insights. The dashboard shall display key performance metrics including total revenue, total bookings, active users, and average customer ratings. It shall also feature interactive charts to visualize revenue analytics and the performance of different vehicle categories. Additionally, the dashboard shall allow administrators to monitor real-time booking trends and system activity, ensuring efficient decision-making and smooth overall management. The dashboard shall also include a transaction log to track recent payments and booking updates. Furthermore, administrators shall have access to quick-action controls for managing vehicles, packages, users, and system settings directly from the dashboard. The Admin Dashboard provides a centralized interface for managing the entire system. Admins can monitor vehicle availability, approve or cancel bookings, manage users, and generate reports. It offers real-time insights to ensure smooth operations and efficient fleet management.

2.2 NON-FUNCTIONAL REQUIREMENTS

The system shall meet several non-functional requirements to ensure a smooth and dependable user experience. The interface shall be highly usable, intuitive, and aligned with modern UI/UX standards so that users require minimal training. Performance shall be optimized such that initial views load within 3 seconds, and AI API responses are processed asynchronously with clear loading indicators. Reliability is essential; therefore, core booking and authentication processes must operate without crashes during any user session. Security shall be maintained by storing passwords in memory for this prototype and enforcing authentication for all booking-related actions. The system shall be maintainable through modular, cleanly structured, and well-documented TypeScript code. It shall also be portable, running efficiently across all modern web browsers including Chrome, Firefox, Safari, and Edge.

2.3 HARDWARE REQUIREMENTS

The hardware requirements for the system are minimal, ensuring easy deployment and accessibility. On the server side, the application can be hosted on any standard web hosting service such as Vercel, Netlify, or AWS S3 that is capable of serving static files. On the client side, users only need a device such as a PC, laptop, tablet, or smartphone equipped with a modern web browser and an active internet connection. Additionally, the system shall function smoothly even on low-specification devices, ensuring broad accessibility for all users. The system shall also utilize minimal processing power, allowing it to run efficiently without requiring specialized hardware. Furthermore, future scalability options shall support deployment on cloud-based servers for handling increased user traffic. The Vehicle Pre-Booking System requires a computer with at least an Intel i3 or equivalent AMD processor, 4 GB of RAM (8 GB recommended), and 500 GB HDD or 256 GB SSD for smooth operation. A 14-inch HD display or higher is preferred, along with a stable internet connection to ensure real-time booking updates and seamless system performance.

2.4 SOFTWARE REQUIREMENTS

The system requires a modern and efficient software stack to ensure smooth development and high-quality performance. The frontend is built using React 19.2.0, providing a fast and modular component-based architecture. TypeScript (~5.8.2) is used as the primary programming language to ensure type safety, cleaner code, and easier debugging. Vite 6.2.0 serves as the build tool, enabling lightning-fast development and optimized production builds. For UI elements, Lucide React is used to provide lightweight and customizable icons. Recharts 3.5.0 is integrated to generate interactive and visually appealing charts within the admin dashboard. AI features are powered by the Google Gemini API, accessed through the @google/genai 1.30.0 package. Development relies on the latest LTS version of Node.js to ensure compatibility and stability. Git is used for version control, enabling safe collaboration and code management. Additionally, the system supports integration with modern code editors like VS Code for an enhanced development experience. The software stack is designed to be scalable, allowing easy upgrades and feature expansion in future versions.

2.5 USER CHARACTERISTICS

The system is designed for two primary user groups with distinct characteristics. End users or customers are individuals looking to rent vehicles, pre-book new cars, or book holiday packages; they possess basic computer literacy and typically access the system through mobile devices or desktop browsers. Administrators, on the other hand, are staff members or managers responsible for monitoring bookings, reviewing analytics, and overseeing overall system performance. They require access to more detailed data, advanced functionalities, and real-time insights to manage operations effectively. The system is designed for users of varying technical expertise, including customers, fleet managers, and administrators. Customers can easily search, book, and manage vehicle reservations with minimal training. Fleet managers and admins require basic computer skills to monitor bookings, manage vehicles, and generate reports. The interface is intuitive, ensuring accessibility and ease of use for all user types.

2.6 CONSTRAINTS

The system operates under several important constraints that shape its design and functionality. Technically, it functions as a frontend-only single-page application, with all data stored in memory, meaning information is lost upon page refresh since this is only a prototype. The AI-based features rely on the external Google Gemini API, which requires a valid API key and a stable internet connection for proper operation. Browser compatibility is also a limiting factor, as the application uses modern JavaScript (ES2022) and may not work correctly on outdated browsers. Additionally, the entire project is developed within a restricted academic or MVP timeframe, which limits the scope for implementing advanced backend features or long-term storage solutions.

CHAPTER 3

ANALYSIS AND DESIGN

3.1 USE CASE MODEL

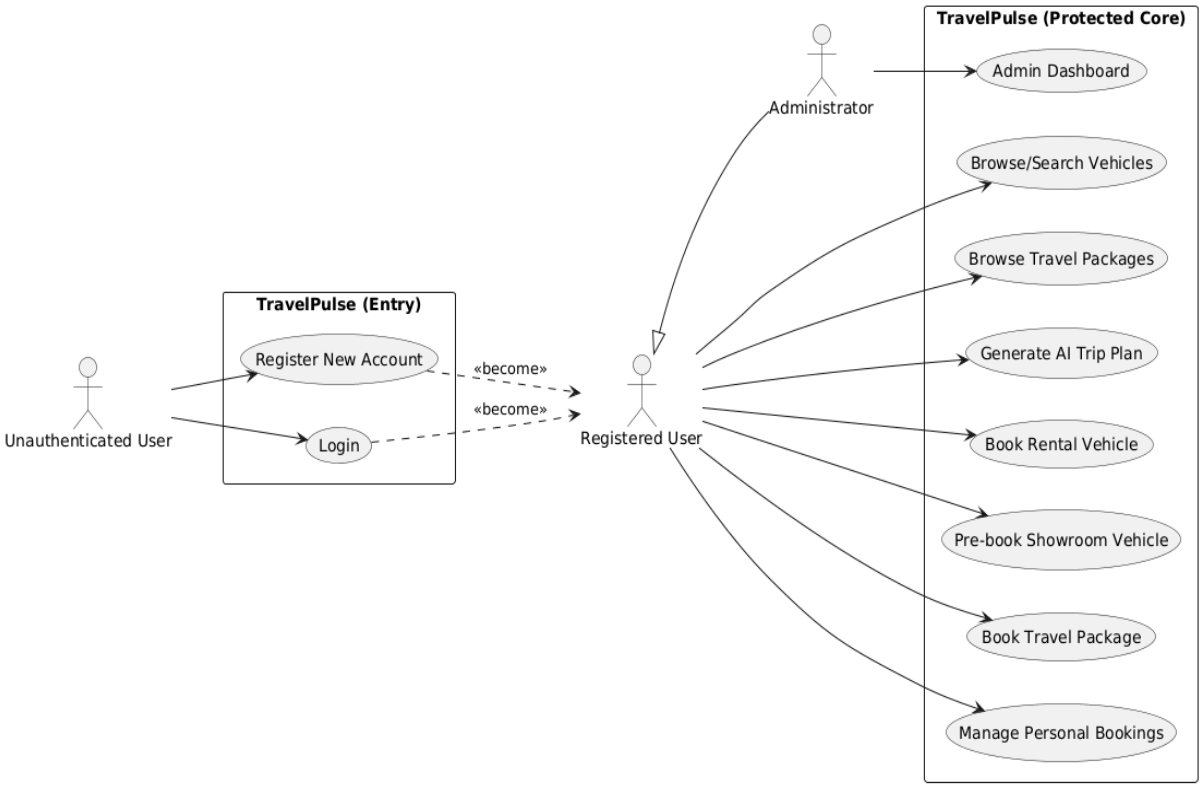

Figure 3.1 Use Case Model

3.1.1 Use Case Description

In the “Book Rental Vehicle” use case, the actor is an authenticated customer who must be logged in and viewing the Rentals or Home page. The process begins when the user selects a vehicle from the available list and clicks the “Rent Now” button, prompting the system to open a booking modal. The system automatically pre-fills the start date as the current day and the end date as the following day, after which the user may adjust either date as needed. As the dates are modified, the system dynamically recalculates the total rental cost based on the vehicle’s daily rate multiplied by the selected number of days.

3.2 CLASS DIAGRAM

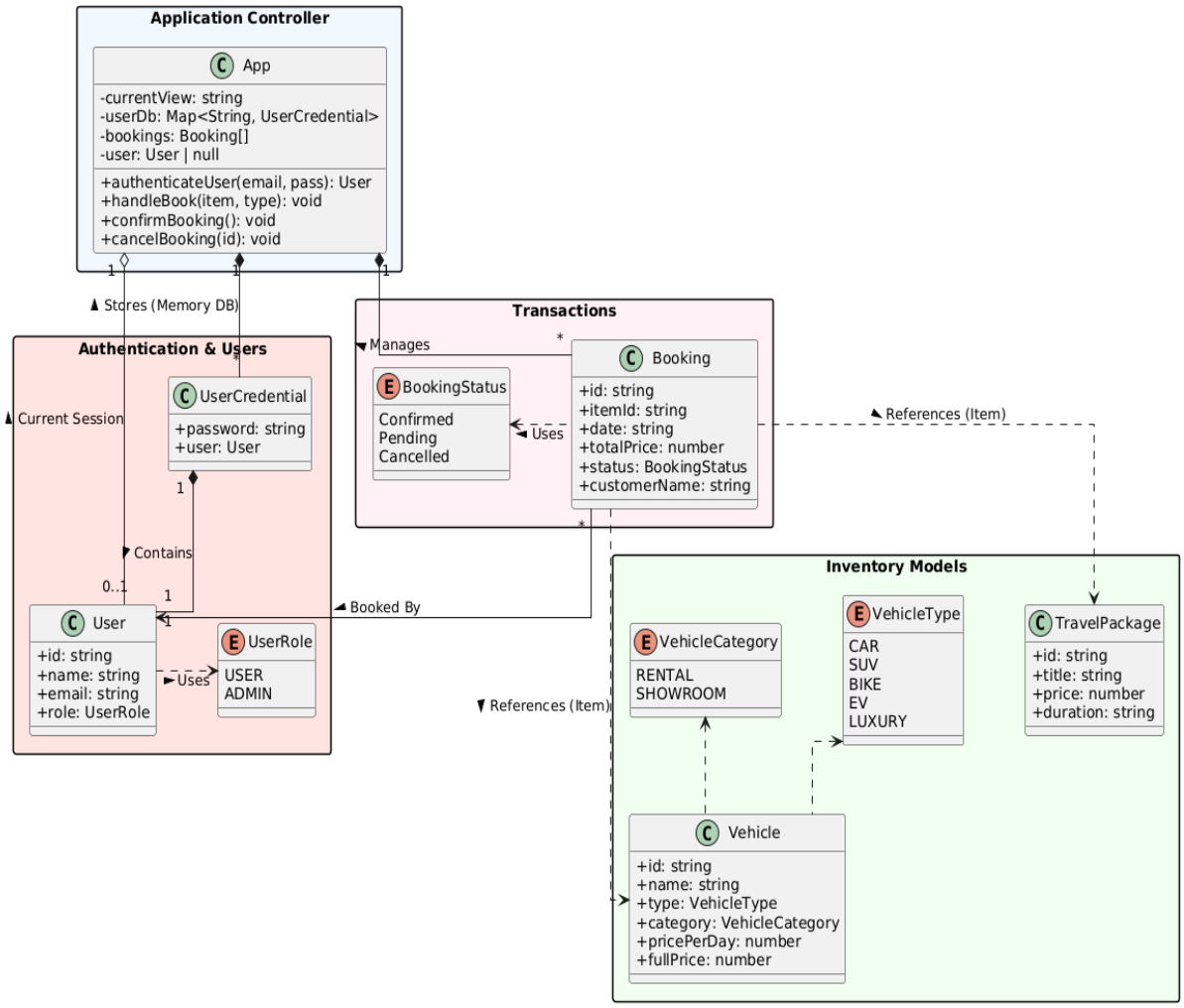

Figure 3.3 Class Diagram

3.2.1 Class Diagram Description

The system architecture is centered around the App Controller, which manages the overall application state through attributes such as currentView, a userDb map storing UserCredentials, a bookings array, and the currently authenticated User. It provides core methods including authenticateUser(), handleBook(), confirmBooking(), and cancelBooking() to control user actions and booking workflows. A UserCredential object links a user's password to their corresponding User profile, while the User model includes details such as id, name, email, and role. Booking objects store essential transaction data, id, itemId, itemType, date, totalPrice, status (Confirmed, Pending, or Cancelled), and customerName, along with optional startDate and endDate for rental bookings.

3.3 ACTIVITY DIAGRAM

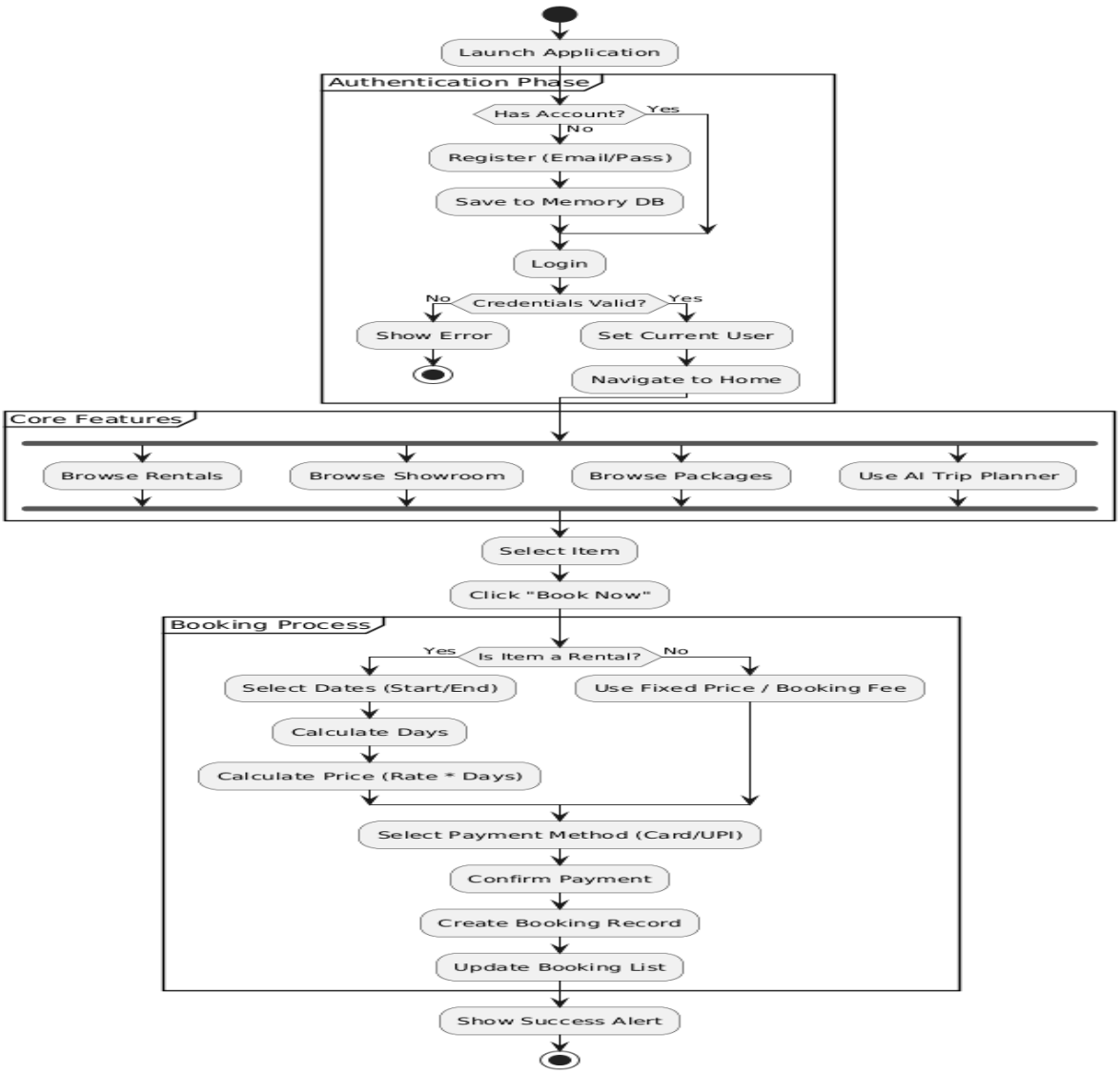

Figure 3.2 Activity Diagram

3.3.1 Activity Diagram Description

The system workflow begins with launching the application, followed by the authentication phase, where users either log in with valid credentials or register a new account, which is then saved to the in-memory database and used to log in; any authentication failures trigger an error message. Once authenticated, users can access core features, including browsing rental vehicles, showroom cars, travel packages, or generating personalized plans through the AI Trip Planner.

3.4 SEQUENCE DIAGRAM

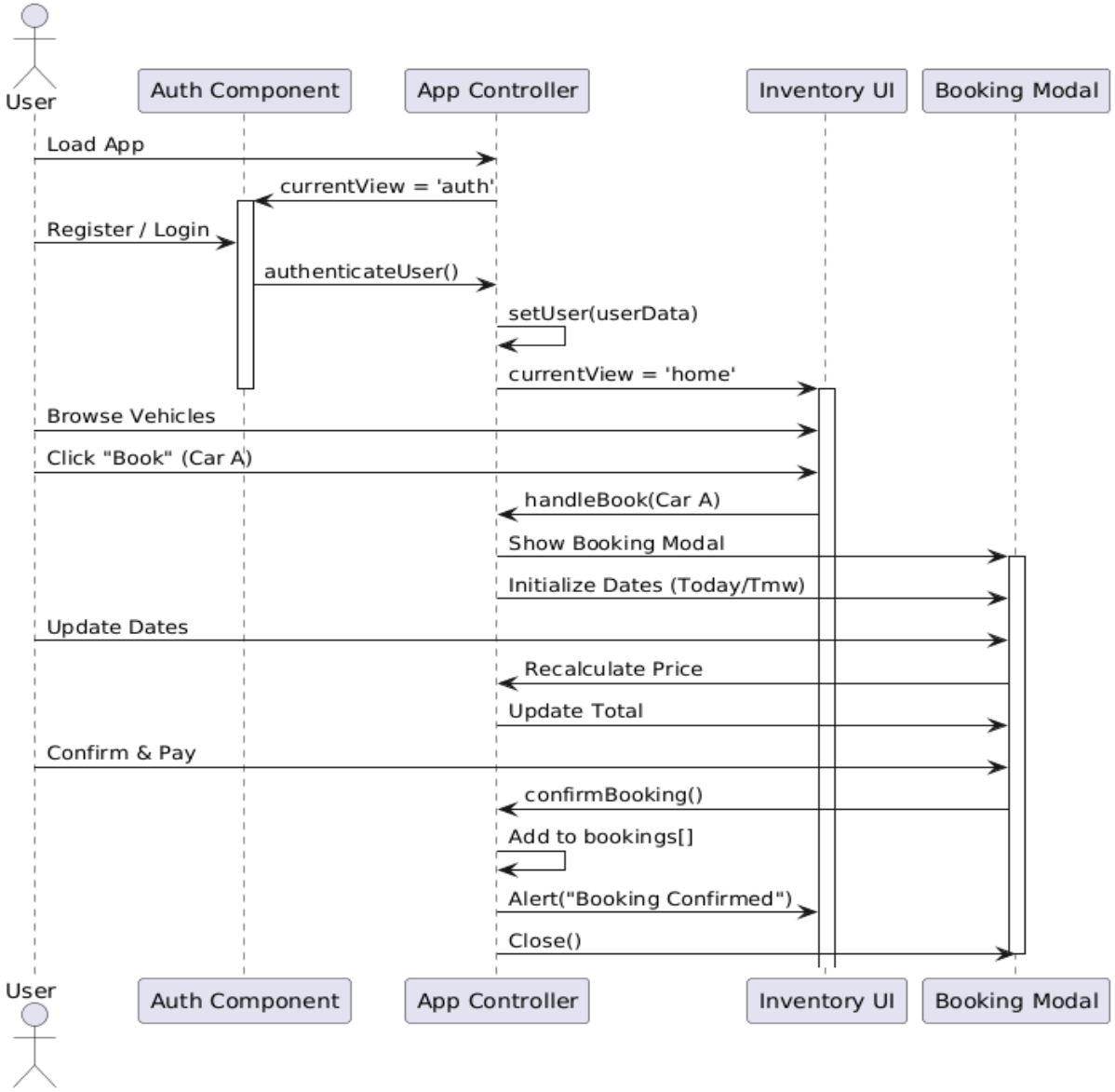

Figure 3.4 Sequence Diagram

3.4.1 Sequence Diagram Description

User clicks "Book" on a vehicle in the Inventory UI.Inventory UI calls handleBook(item) on the App Controller.App Controller triggers the Booking Modal to open.Booking Modal initializes dates (today/tomorrow) and displays.User updates dates in the modal.Booking Modal recalculates price and updates the total display.User clicks "Confirm & Pay".Booking Modal calls confirmBooking() on the App Controller.App Controller shows an "Alert" (Booking Confirmed).

3.5 STATE MACHINE DIAGRAM

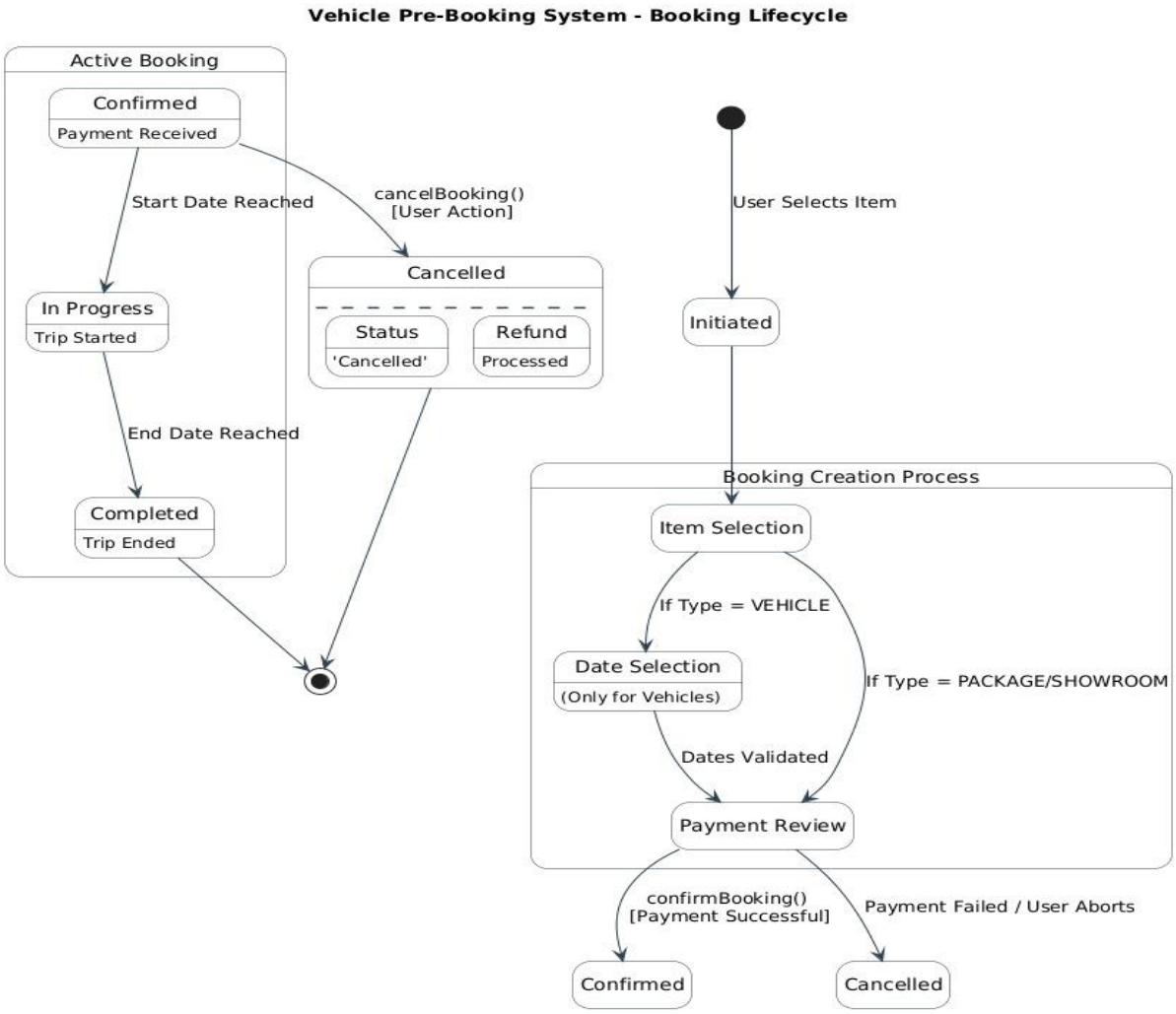

Figure 3.5 State Machine Diagram

3.5.1 State Machine Diagram Description

The Vehicle Pre-Booking System implements several state machines that govern the behavior of key entities throughout the application lifecycle. These state machines ensure proper workflow management and data consistency. User Session State Machine: This machine manages the authentication status of a user interacting with the system. Booking Process State Machine: This machine defines the lifecycle of a booking transaction from initiation to completion. Booking Entity State Machine: This machine represents the status of an individual booking record in the system. Application View State Machine: This machine controls the navigation and rendering of different views in the application.

3.6 COMPONENT DIAGRAM

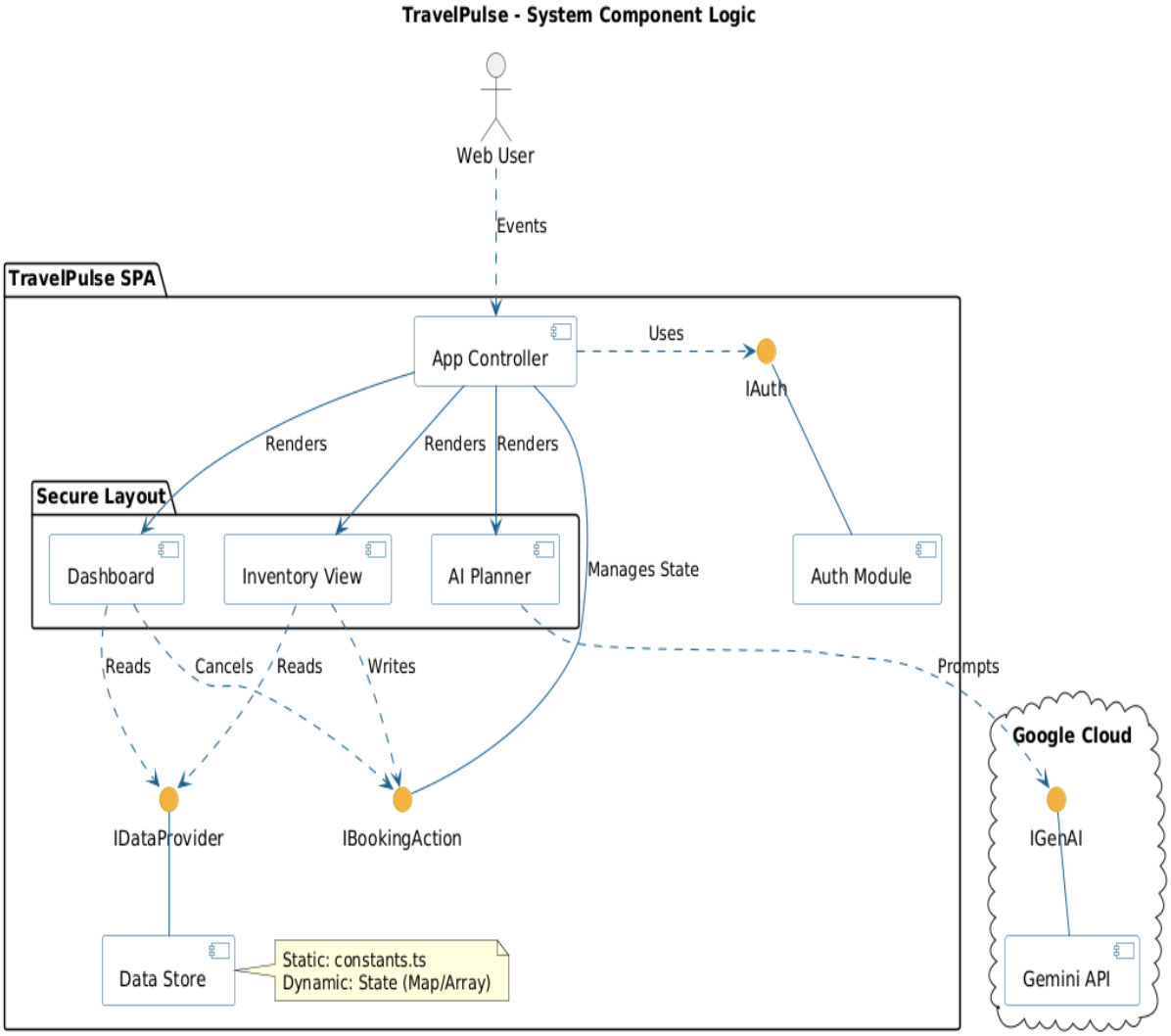

Figure 3.6 Component Diagram

3.6.1 Component Diagram Description

The system architecture is organized into four main layers to ensure modularity and clarity. The Presentation Layer, implemented as a single-page application (SPA), is managed by the App Controller and includes View Controllers for Inventory, Dashboard, and Planner, along with a Layout Manager to handle UI rendering and navigation. The Business Logic Layer contains core functions such as validateSession(), handleBook(), and generatePlan(), controlling access with “Access Granted” checks and managing transaction saving. The Persistence Layer is responsible for loading inventory data and performing read/write operations on the Booking Registry, ensuring that all user actions are accurately recorded.

3.7 DEPLOYMENT DIAGRAM

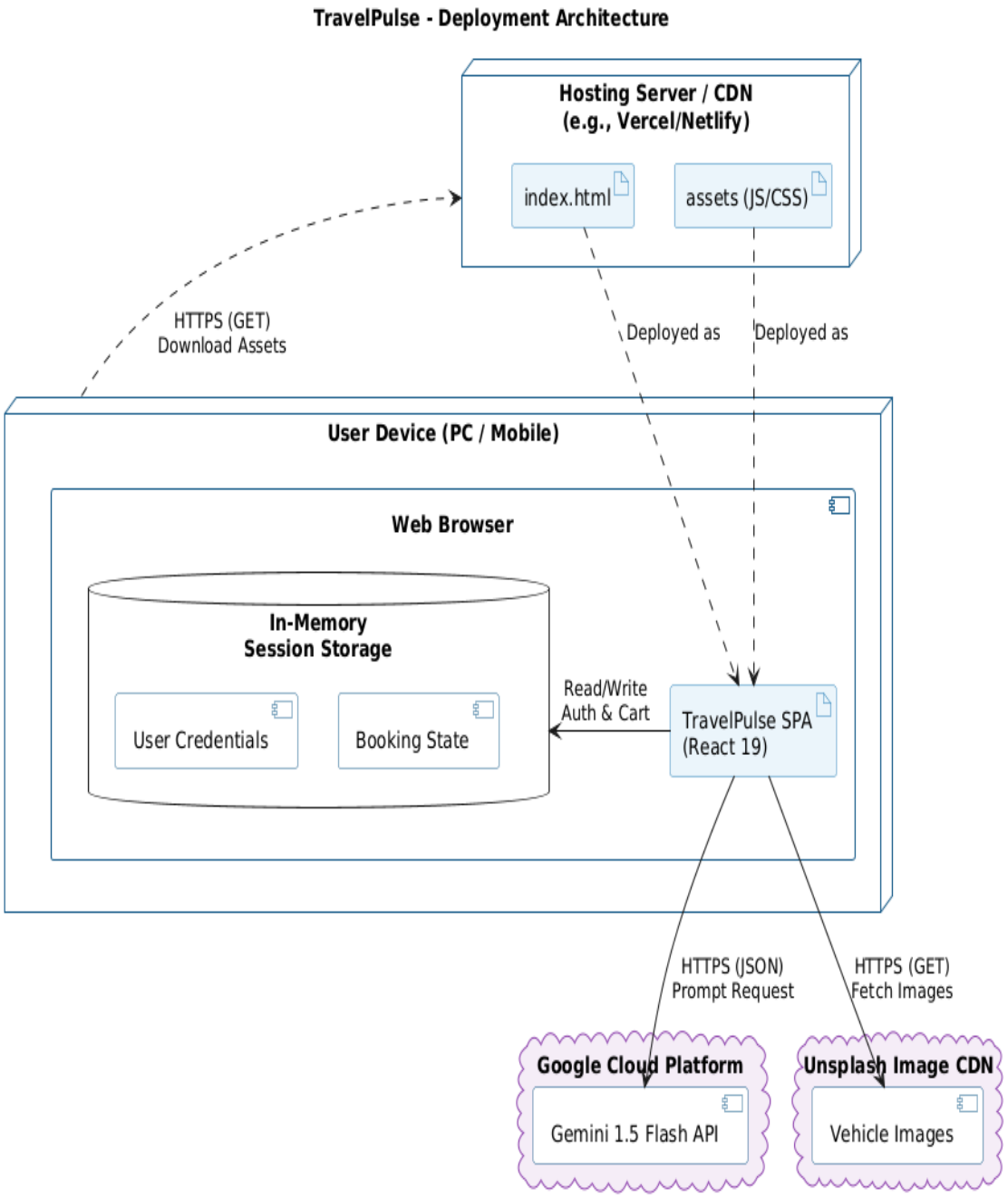

Figure 3.7 Deployment Diagram

3.7.1 Deployment Diagram Description

The User Device (browser) downloads these assets via HTTPS.The SPA runs in the browser, maintaining state in In-Memory Session Storage.The SPA makes HTTPS requests to:Google Cloud Platform (Gemini API) for AI features.Unsplash Image CDN for vehicle images.

3.8 PACKAGE DIAGRAM

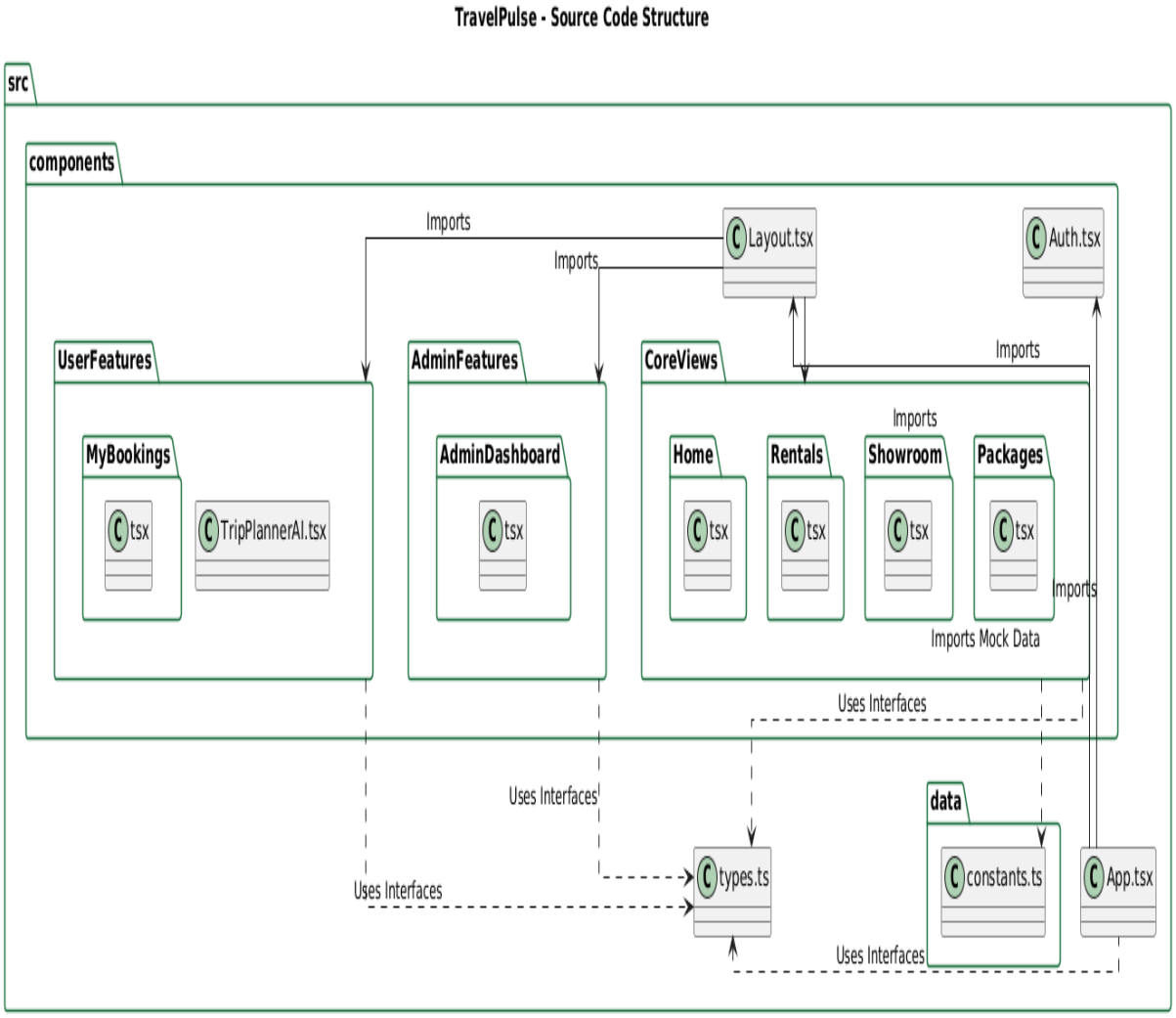

Figure 3.8 Package Diagram

3.8.1 Package Diagram Description

The TravelPulse source code is organized in a modular structure within the src directory. The components folder contains reusable UI elements. The UserFeatures folder includes files such as MyBookings.tsx and TripPlannerAI.tsx for customer-specific functionality, while the AdminFeatures folder contains AdminDashboard.tsx for administrative tasks. The CoreViews folder holds the main application views, including Layout.tsx, Auth.tsx, Home.tsx, Rentals.tsx, Showroom.tsx, and Packages.tsx, which make up the core pages of the single-page application. The services folder contains geminiService.ts, which interacts with the Google Gemini API and uses interfaces from types.ts to ensure modularity and maintainable code.

3.9 DESIGN PATTERNS USED

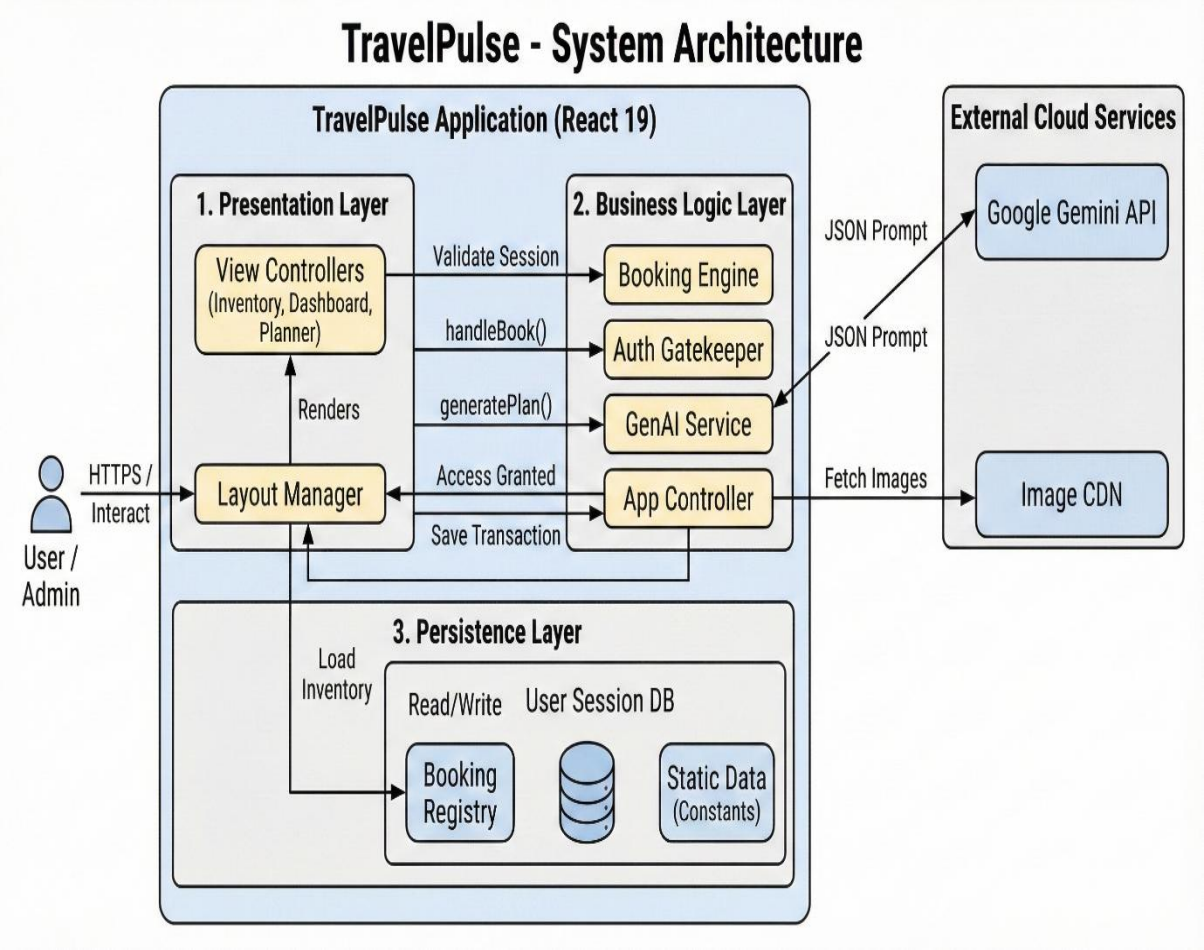

Figure 3.9 System Architecture

3.9.1 Design pattern Description

The system applies several design patterns to ensure clean architecture and maintainable code. In terms of GRASP patterns, the App.tsx component serves as the main controller, handling user events and coordinating interactions between views and data. The Creator pattern is reflected in components like VehicleCard and Packages, which generate booking intent objects that are then passed to the App controller for processing. Regarding GoF patterns, the application effectively implements a Singleton pattern, as the main application state—including userDb, bookings, and the current user—is managed by the root App component and shared throughout the application via props. Additionally, the Strategy pattern is used in the booking confirmation logic, where the calculation of the final price varies based on the item type, such as VEHICLE,SHOWROOM_PREBOOK,orPACKAGE.

CHAPTER 4

IMPLEMENTATION

4.1 MODULE DESCRIPTION

4.1.1 State Management and Controller Module

This module acts as the core controller of the application and is implemented in App.tsx. It manages global state such as the current view, authenticated user, in-memory user database, bookings list, UI preferences, and booking modals. It handles key business logic functions including user authentication, booking creation, confirmation, and cancellation. The module coordinates communication between components through props and controls conditional rendering based on user roles and authentication status to ensure smooth single-page navigation.

4.1.2 User Authentication and Registration Module

The authentication module manages user registration and login using the Auth.tsx component. It validates credentials against an in-memory user database and prevents duplicate registrations. Upon successful login, user session details and role-based access are established. The module includes form validation, feedback messages, and enforces authentication before allowing booking actions.

4.1.3 Vehicle Inventory and Display Module

The Vehicle Inventory and Display Module handles the presentation of rental and showroom vehicles using components such as Rentals.tsx, Showroom.tsx, and VehicleCard.tsx. It organizes vehicles by category and supports search and filtering based on vehicle type and name. Each vehicle is displayed in a standardized card layout showing images, specifications, pricing, and booking options. The module dynamically adapts its UI based on whether the vehicle is for rental or showroom pre-booking. This module serves as the primary interface for browsing and selecting vehicles within the system.

4.1.4 Booking Management and Payment Module

The booking module manages the complete booking process for rentals, showroom vehicles, and travel packages. It supports date selection for rentals, dynamic price calculation, payment method selection, and booking confirmation. Booking records are created with status tracking, and users are allowed to cancel bookings by updating their status in the system.

4.1.5 AI Trip Planning and Recommendation Module

This module integrates the Google Gemini API to provide AI-based trip planning and vehicle recommendations. Implemented in TripPlannerAI.tsx, it allows users to enter natural language queries and receive intelligent travel suggestions. It also supports AI-powered search fallback when local search results are unavailable.

4.1.6 Admin Dashboard and Analytics Module

The Admin Dashboard module provides administrators with system analytics and booking insights. It displays key metrics such as revenue, bookings, and user activity, along with charts and a transaction table. This module enables administrators to monitor performance and manage bookings efficiently.

4.2 TECHNOLOGY DESCRIPTION

The system uses several modern technologies to build a responsive and maintainable application. React 19 is employed as the core JavaScript library for creating reusable UI components and managing the view layer. TypeScript, a strongly-typed superset of JavaScript, is used throughout the codebase to define interfaces, catch errors during development, and improve maintainability. Vite serves as a fast build tool and development server, offering features like Hot Module Replacement (HMR) for an enhanced developer experience. Tailwind CSS, a utility-first framework, is applied via inline classes to style the application quickly and responsively. Lucide React provides customizable SVG icons used across buttons, cards, and navigation elements.

CHAPTER 5

TESTING

5.1 TESTING STRATEGY

The system follows a structured testing strategy to ensure reliability and correctness. Unit testing was performed manually, where individual functions such as authenticateUser, getDynamicPrice, and handleBook were tested by developers with various inputs, and outputs were verified through the browser console or UI behavior. Integration testing focused on the interactions between components, for instance, confirming that clicking "Book" on a VehicleCard in Rentals.tsx correctly triggered the handleBook method in App.tsx and opened the booking modal.

5.2 SAMPLE TEST CASES:

Test Case 1:

Name: TC01 – Home Page Load

Description: Verifies that the home page loads correctly with navigation and main sections.

Input: Open application URL

Expected Result: Home page displays hero section, service cards, and popular rentals.

Status: Pass

Test Case 2:

Name: TC02 – User Registration

Description: Verifies successful registration using valid user details.

Input: Enter valid name, email, password, role → Click Sign Up

Expected Result: User stored in userDb and success message displayed.

Status: Pass

Test Case 3:

Name: TC03 – User Login

Description: Verifies successful login with valid credentials.

Input: Enter registered email and password → Click Sign In

Expected Result: User authenticated and redirected to Home page.

Status: Pass

Test Case 4:

Name: TC04 – Invalid Login Attempt

Description: Checks system behavior for incorrect login credentials.

Input: Enter invalid email or password → Click Sign In

Expected Result: Error message displayed and login denied.

Status: Pass

Test Case 5:

Name: TC05 – Vehicle Browsing and Filtering

Description: Verifies vehicle browsing and filter functionality.

Input: Navigate to Rentals → Apply vehicle type filter

Expected Result: Only selected vehicle type is displayed.

Status: Pass

Test Case 6:

Name: TC06 – Rental Vehicle Booking

Description: Verifies rental booking with date selection and price calculation.

Input: Select rental vehicle → Choose dates → Confirm payment

Expected Result: Booking created with correct total price and status confirmed.

Status: Pass

Test Case 7:

Name: TC07 – Booking Cancellation

Description: Verifies cancellation of an existing booking.

Input: Go to My Bookings → Click Cancel

Expected Result: Booking status changes to Cancelled.

Status: Pass

Test Case 8:

Name: TC08 – AI Trip Planner

Description: Verifies AI trip planning functionality.

Input: Enter travel query → Click Send

Expected Result: AI-generated response displayed after loading indicator.

Status: Pass

Test Case 9:

Name: TC09 – Admin Dashboard Access

Description: Verifies dashboard access for admin users only.

Input: Login as Admin → Click Dashboard

Expected Result: Dashboard loads with analytics and transaction data.

Status: Pass

Test Case 10:

Name: TC10 – Responsive Design

Description: Validates UI responsiveness across devices.

Input: Resize browser window to mobile size

Expected Result: Layout adapts correctly with mobile navigation.

Status: Pass

5.3 TEST RESULTS:

All major test cases were executed successfully and produced the expected results. User authentication, vehicle search and filtering, booking workflows, payment simulation, and booking cancellation functioned correctly. The AI Trip Planner responded accurately to valid inputs, and admin access control worked as intended. Navigation between modules was smooth without page reloads. The system remained stable during testing with no crashes or data inconsistencies. Overall, the application met all functional and usability requirements. Additionally, responsive design testing confirmed proper behavior across different screen sizes.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The Vehicle Pre-Booking System project successfully demonstrates the development of a feature-rich, AI-integrated web application for managing vehicle and travel bookings. The system meets its core objectives by providing a unified platform for rentals, showroom pre-booking, and tour packages, enhanced with an intelligent trip planner. The implementation of a secure client-side authentication flow, a dynamic booking engine, a comprehensive admin dashboard, and a polished, responsive UI validates the chosen technology stack and architectural approach. The use of React, TypeScript, and modern UI libraries resulted in a maintainable and scalable codebase. While functioning as a complete prototype, the project clearly outlines the workflow and user experience for a real-world booking system.

6.2 FUTURE ENHANCEMENT

Future enhancements for the system include improving both backend functionality and overall user experience. A key upgrade is integrating a full backend using Node.js or Python Django with a PostgreSQL database to replace the current in-memory storage, ensuring secure and persistent data management for users, bookings, and inventoryEmail and SMS notification services can be implemented to automatically inform users about booking confirmations, reminders, updates, and cancellations. An inventory management CMS can also be introduced, allowing administrators or staff to easily add, edit, or manage vehicles and travel packages without manually modifying the source code.Additional enhancements include enabling user reviews and ratings for vehicles and packages to improve transparency and trust. AI-based recommendation features can further personalize the user experience by suggesting vehicles, packages, or routes based on the user’s preferences, search history, and past bookings. These future improvements will make the system more scalable, professional, and ready for real-world deployment.

APPENDIX-A

SOURCE CODE

Index Page

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Home Page

```
import React from 'react';
import { MapPin, Calendar, Search, Car, CheckCircle } from 'lucide-react';
import { MOCK_VEHICLES } from '../constants';
import { VehicleCategory } from '../types';
import VehicleCard from './VehicleCard';

const Home = ({ user, setView, onBook }) => (
  <div>
    {/* Hero Section */}
    <div className="hero">
      <h1>Vehicle Pre-Booking System</h1>
      <p>Rent vehicles, pre-book showroom cars & explore packages.</p>

      <div className="search-box">
```

```
        <input placeholder="Where to?" />
        <input type="date" />
        <button onClick={() => setView('vehicles')}>
          <Search /> Search
        </button>
      </div>
    </div>
    {/* Services */}
    <div className="services">
      <div onClick={() => setView('vehicles')}> <Car /> Vehicle Rentals </div>
      <div onClick={() => setView('showroom')}> <CheckCircle /> Showroom
Booking </div>
      <div onClick={() => setView('packages')}> <MapPin /> Trips & Tours </div>
    </div>

    {/* Popular Rentals */}
    <div className="popular">
      {MOCK_VEHICLES.filter(v => v.category === VehicleCategory.RENTAL)
        .slice(0, 3)
        .map(v => (
          <VehicleCard key={v.id} vehicle={v} onBook={onBook}
isLoggedIn={!!user} />
        ))}
    </div>
  </div>
);
export default Home;
```

Vehicle Card

```
import React from 'react';
```

```
import { Users, Fuel, Settings, Star, Zap, Tag } from 'lucide-react';

const VehicleCard = ({ vehicle, onBook, isLoggedIn }) => (
  <div className="card">
    <img src={vehicle.image} alt={vehicle.name} />

    <div className="rating">
      <Star /> {vehicle.rating}
    </div>

    <h3>{vehicle.name}</h3>
    <p>{vehicle.description}</p>
    <div className="info">
      <Users /> {vehicle.capacity}
      <Settings /> {vehicle.transmission}
      {vehicle.fuel === 'Electric' ? <Zap /> : <Fuel />} {vehicle.fuel}
    </div>

    {vehicle.category === 'SHOWROOM' && (
      <div className="price-tag">
        <Tag /> Ex-Showroom: ${vehicle.fullPrice}
      </div>
    )}
    <button disabled={!vehicle.available} onClick={() => onBook(vehicle)}>
      {!vehicle.available ? 'Unavailable' :
       !isLoggedIn ? 'Login' :
       vehicle.category === 'SHOWROOM' ? 'Pre-Book' : 'Rent Now'}
    </button>
  </div>
);
export default VehicleCard;
```

Rentals Booking Page

```
import React, { useState, useEffect } from 'react';
import { MOCK_VEHICLES } from '../constants';
import { VehicleCategory, VehicleType } from '../types';
import VehicleCard from './VehicleCard';
import { getAIRecommendations } from '../services/geminiService';

const Rentals = ({ user, onBook }) => {
  const [search, setSearch] = useState('');
  const [type, setType] = useState('All');
  const [aiList, setAiList] = useState([]);
  const [loading, setLoading] = useState(false);

  const filtered = MOCK_VEHICLES.filter(
    v => v.category === VehicleCategory.RENTAL &&
    (type === 'All' || v.type === type) &&
    v.name.toLowerCase().includes(search.toLowerCase())
  );
  useEffect(() => {
    if (filtered.length === 0 && search.length > 2) {
      setLoading(true);
      getAIRecommendations(search).then(res => {
        setAiList(res);
        setLoading(false);
      });
    } else setAiList([]);
  }, [search]);

  return (
```

```
    <div>
      <h1>Vehicle Rentals</h1>
      <select value={type} onChange={e => setType(e.target.value)}>
        <option value="All">All Types</option>
        <option value={VehicleType.CAR}>Cars</option>
        <option value={VehicleType.BIKE}>Bikes</option>
        <option value={VehicleType.BUS}>Buses</option>
      </select>

      <input placeholder="Search..." onChange={e => setSearch(e.target.value)} />
      <div className="grid">
        {filtered.map(v => (
          <VehicleCard key={v.id} vehicle={v} onBook={onBook}
isLoggedIn={!!user} />
        ))}
      </div>
      {loading && <p>Searching...</p>}

      {!loading && aiList.length > 0 && (
        <div className="ai-section">
          {aiList.map(v => (
            <VehicleCard key={v.id} vehicle={v} onBook={onBook}
isLoggedIn={!!user} />
          ))}
        </div>
      )}
    </div>
  );
};
export default Rentals;
```

AI Trip Planner

```
import React, { useState } from 'react';
import { generateTripAdvice } from '../services/geminiService';

const TripPlannerAI = () => {
  const [query, setQuery] = useState('');
  const [reply, setReply] = useState('');
  const [loading, setLoading] = useState(false);
  const askAI = async e => {
    e.preventDefault();
    setLoading(true);
    setReply(await generateTripAdvice(query));
    setLoading(false);
  };
  return (
    <div>
      <h2>AI Smart Travel Assistant</h2>

      <form onSubmit={askAI}>
        <input value={query} onChange={e => setQuery(e.target.value)}
placeholder="Ask about trip..." />
        <button type="submit">Ask</button>
      </form>

      {loading && <p>Thinking...</p>}
      {reply && <div className="ai-reply">{reply}</div>}
    </div>
  );
};

export default TripPlannerAI;
```

APPENDIX-B

SCREENSHOTS

Login Page:

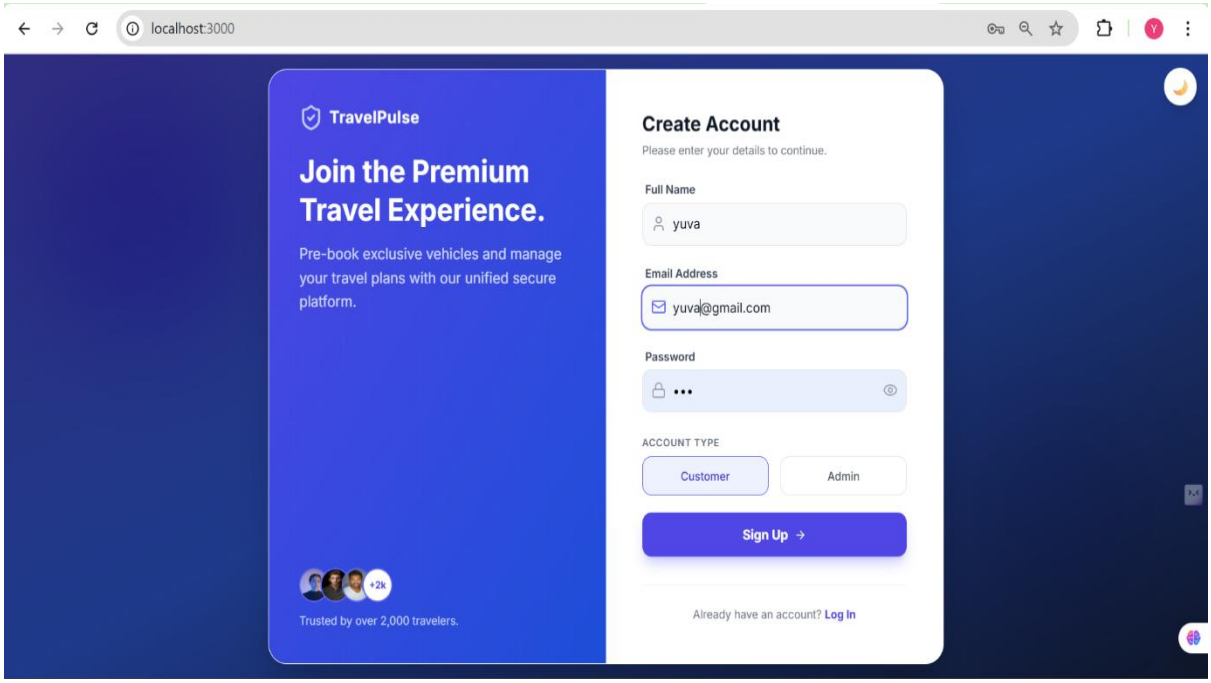

Figure B.1 Login Page

Home Page:

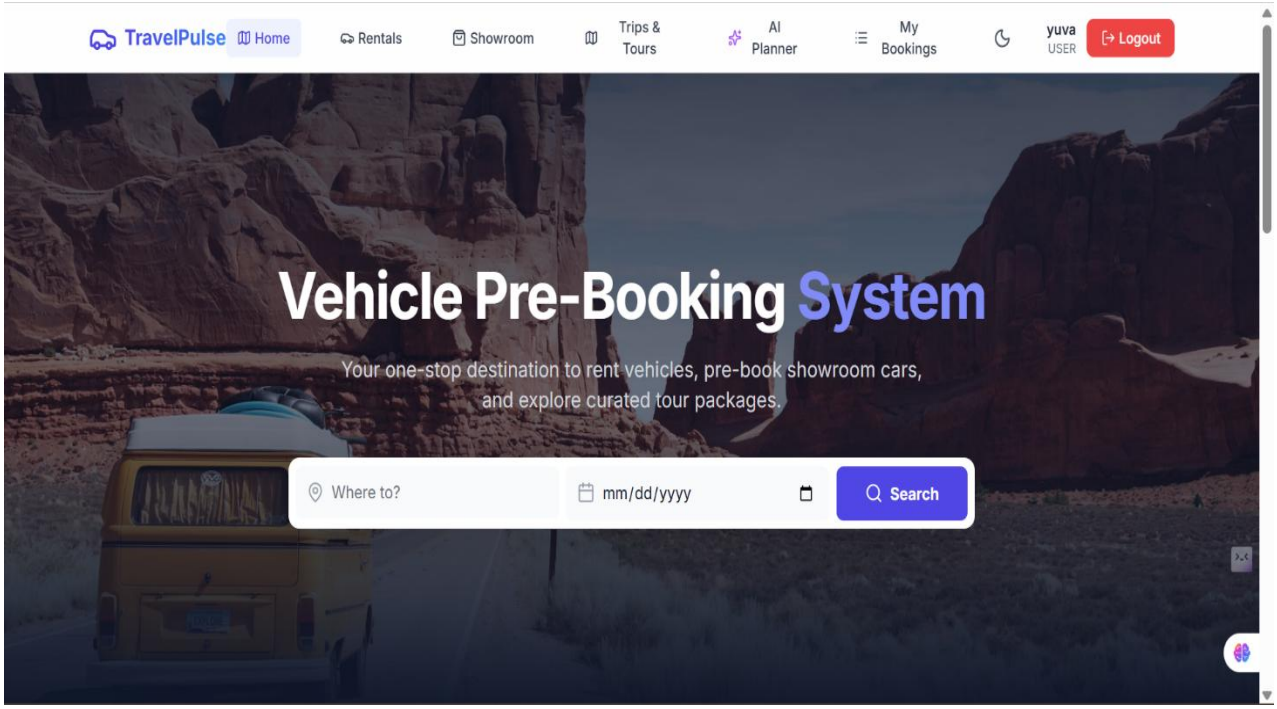

Figure B.2 Home Page

Vehicle Booking:

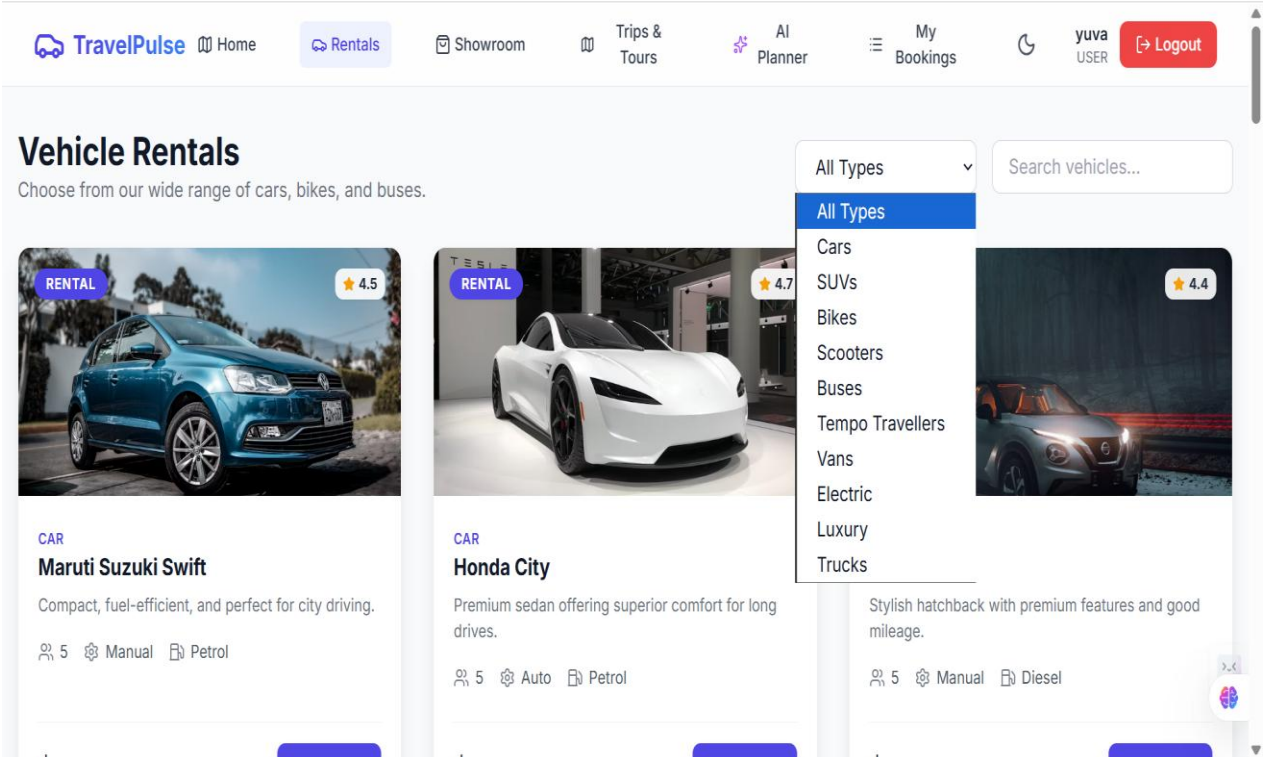

Figure B.3 Vehicle Booking

Travels And Trip Booking Page:

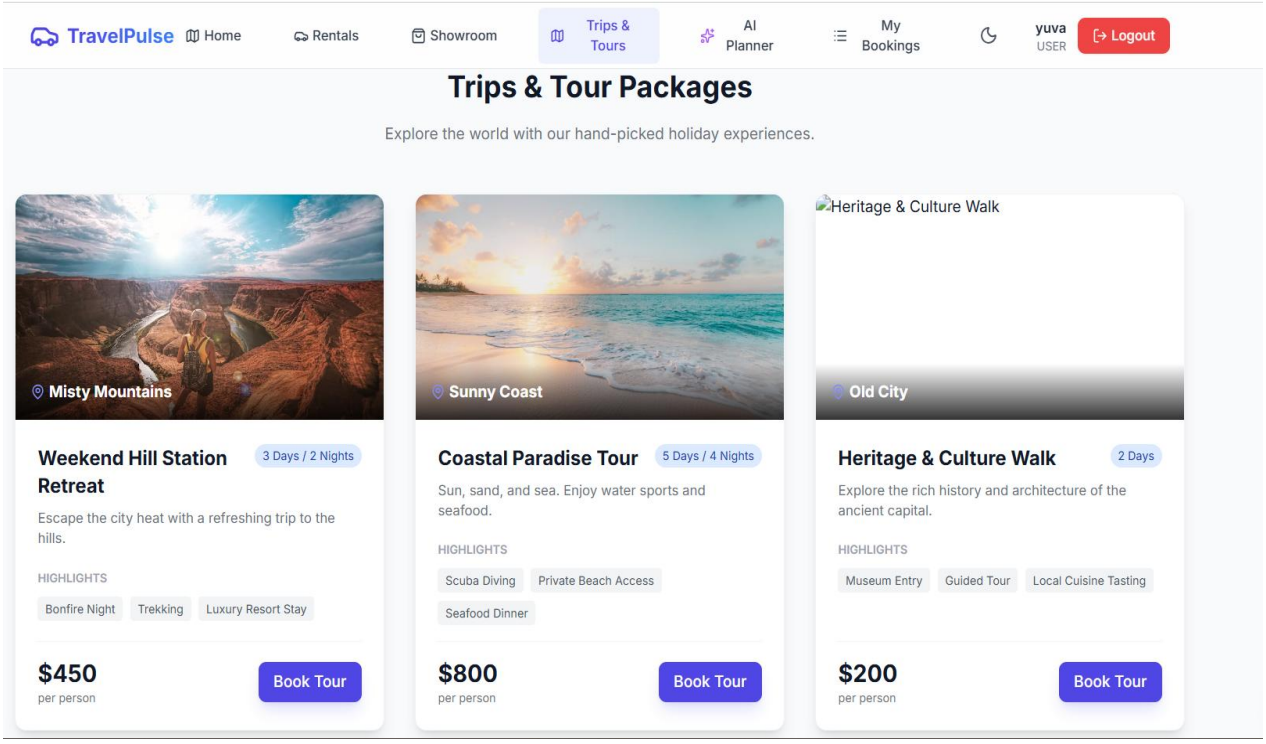

Figure B.4 Trip Booking

Booking Detail:

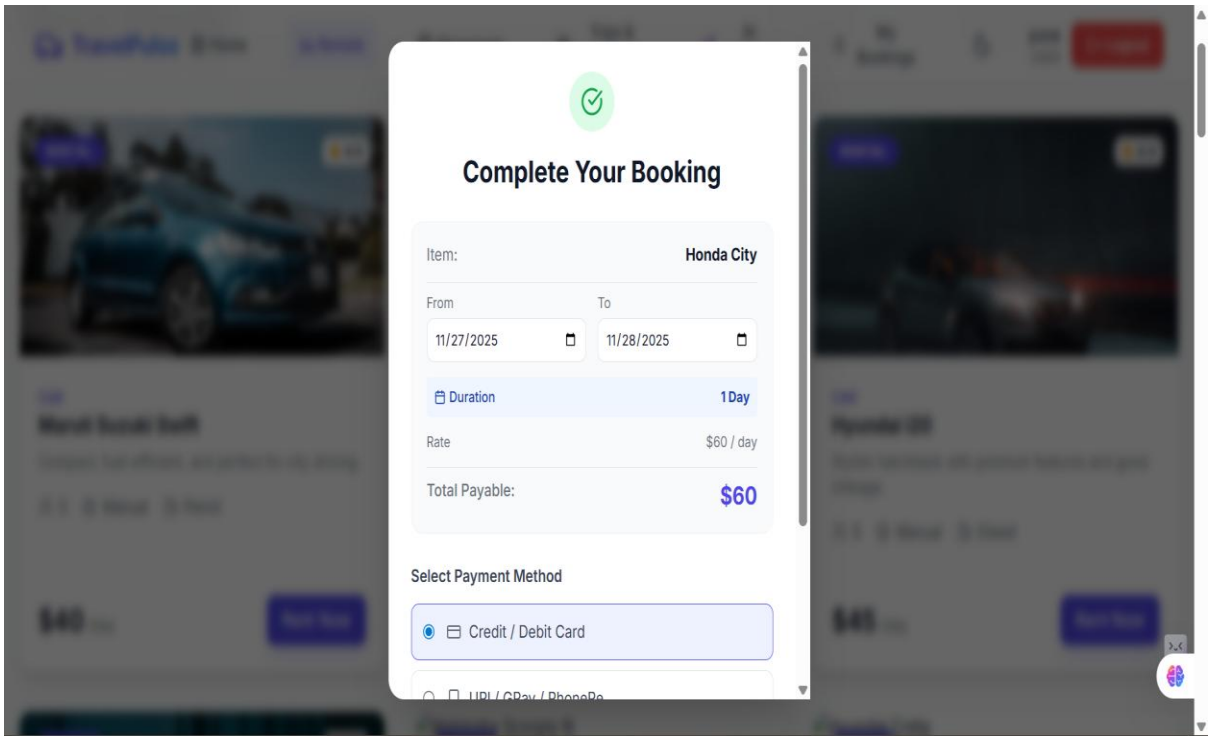

Figure B.5 Booking details

Notification On Booking:

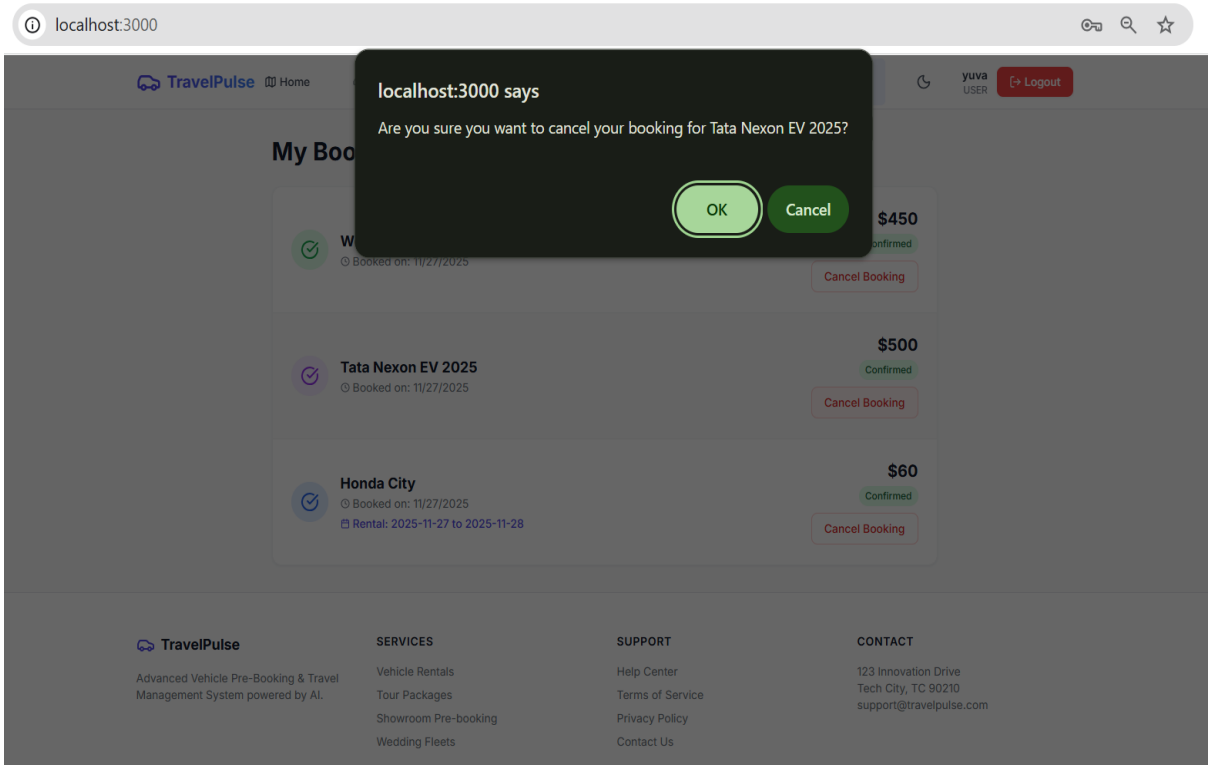

Figure B.6 Notification

Booking Dashboard:

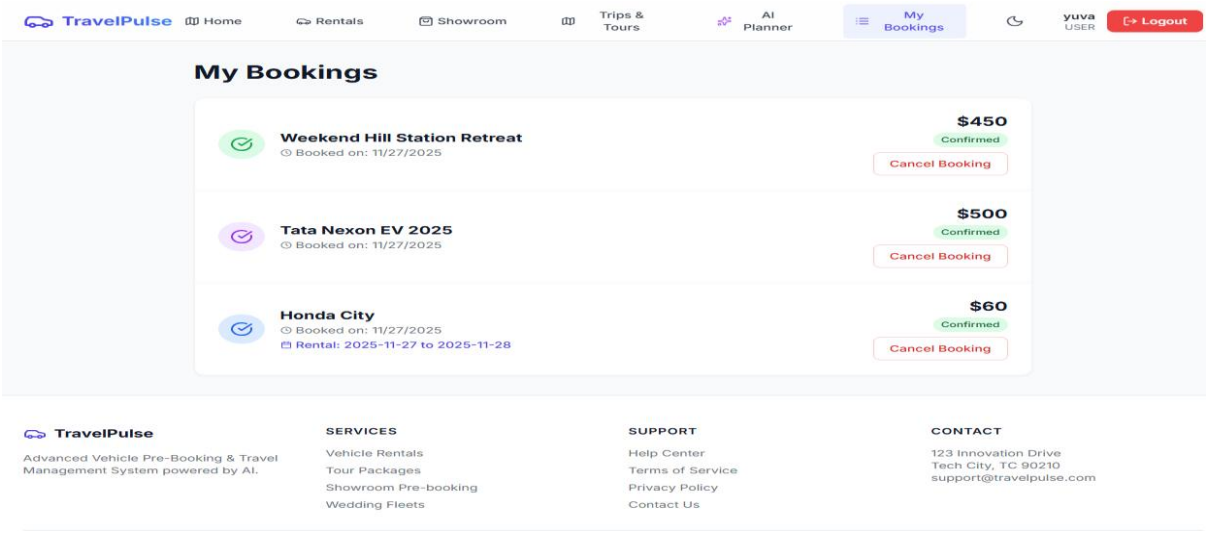

Figure B.7 Booking Dashboard

AI Suggesstion:

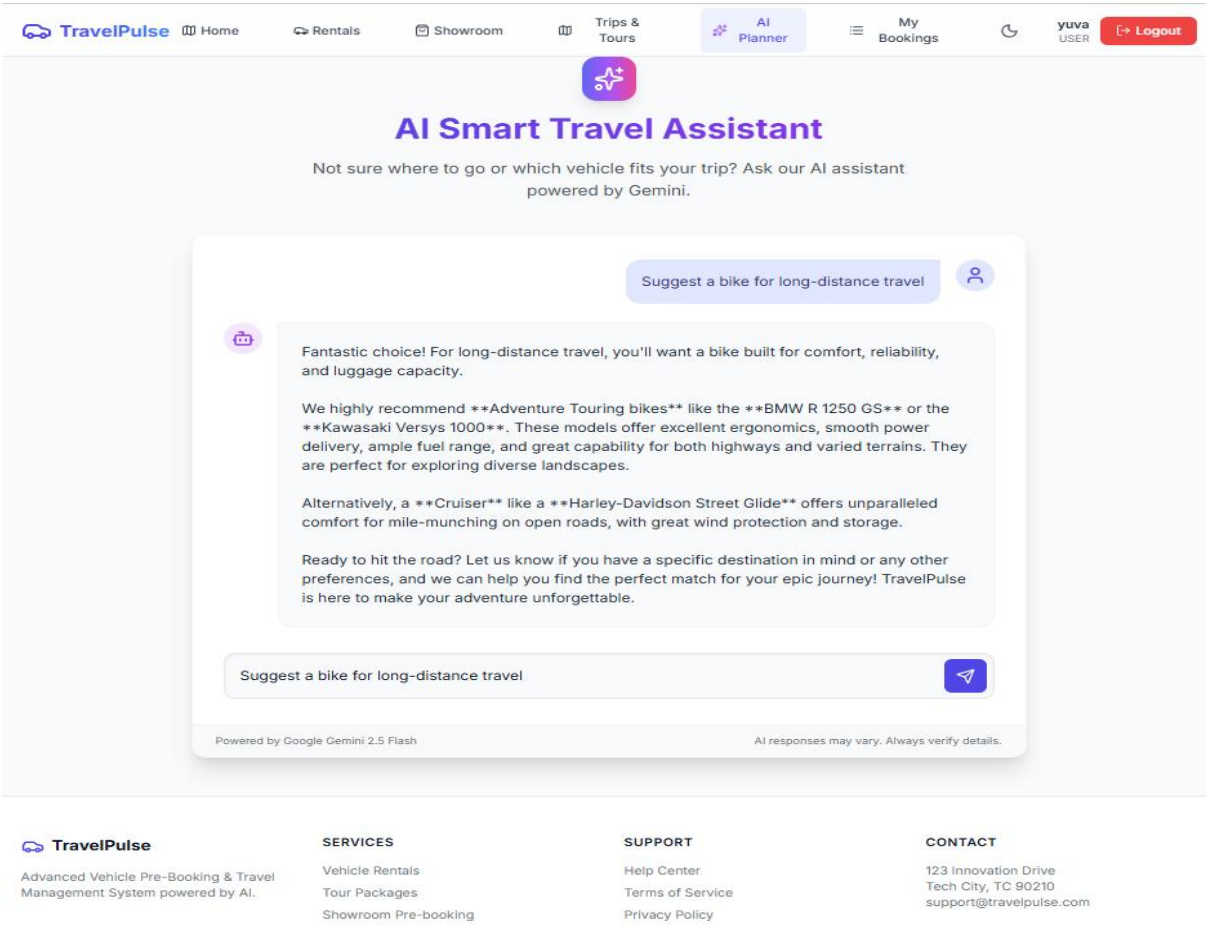

Figure B.8 AI Suggesstion

Admin Login:

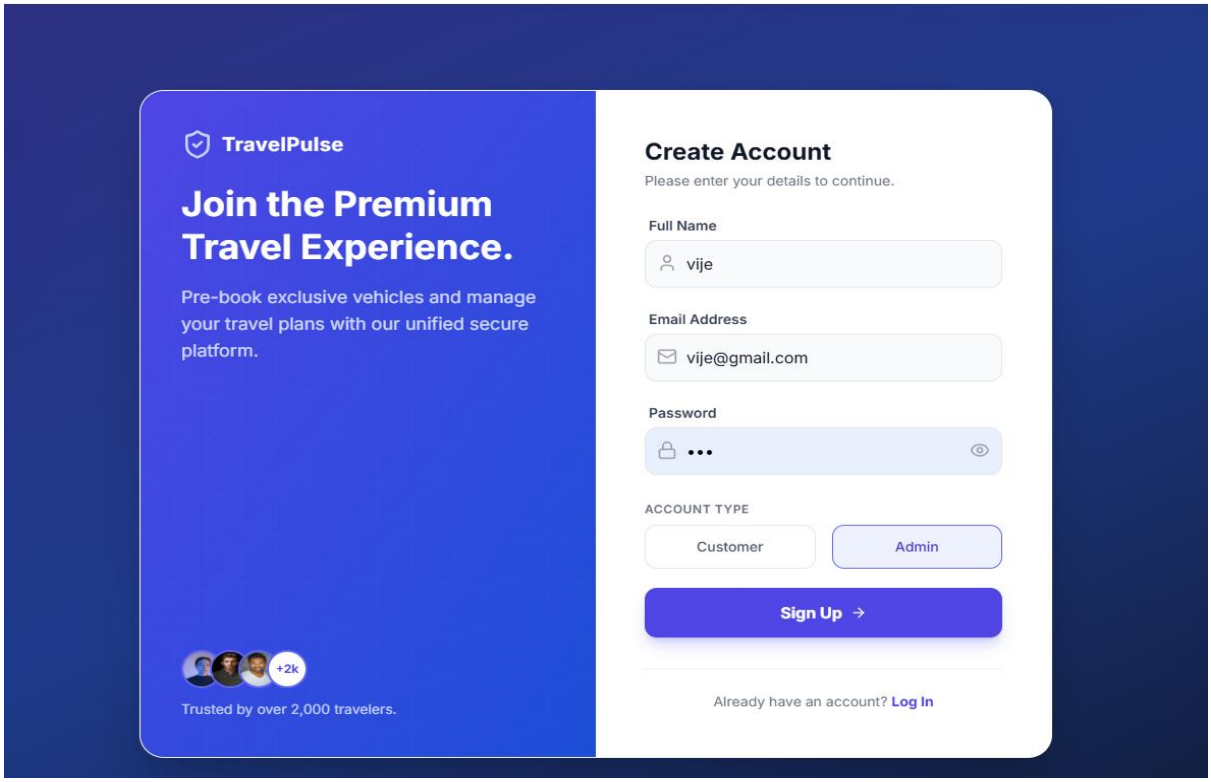

Figure B.9 Admin Login

Admin Dashboard:

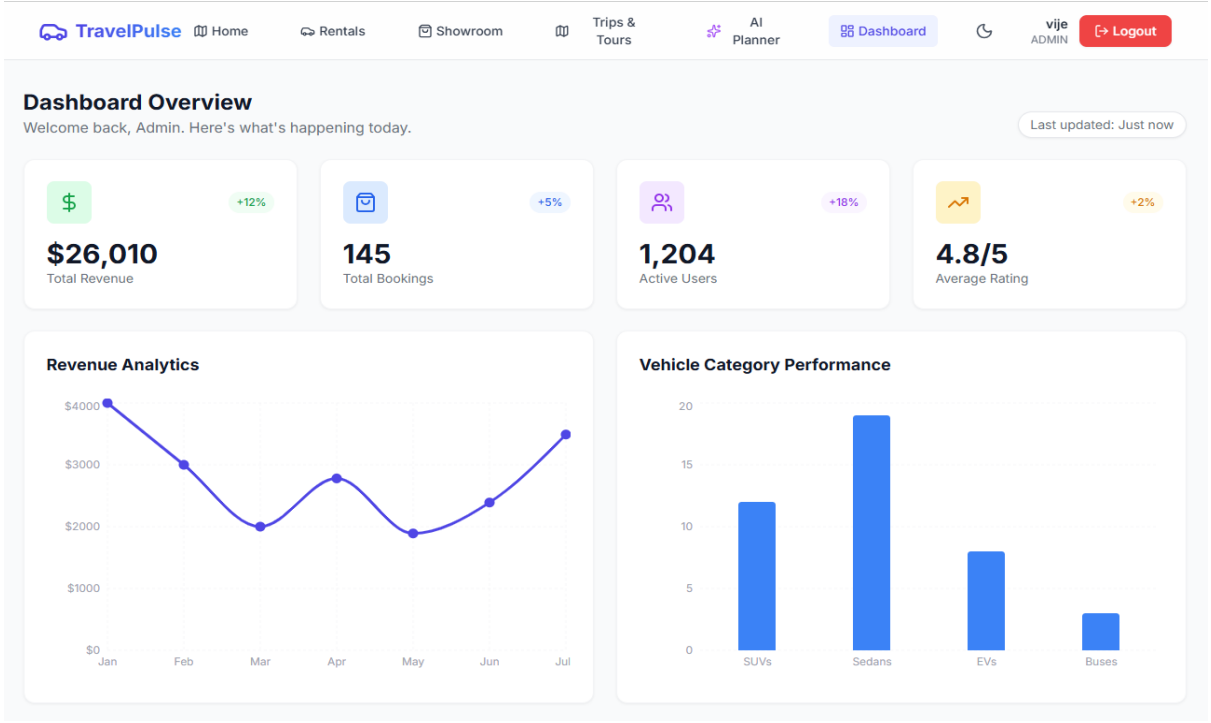

Figure B.10 Admin Dashboard

REFERENCES

Book References:

1. Bierman, G., Abadi, M., & Torgersen, M. Understanding TypeScript. Microsoft Press, 2021.
2. Elliott, J., & Thomas, R. Learning Web Development with React and TypeScript. Packt Publishing, 2023.
3. Flanagan, D. JavaScript: The Definitive Guide (7th Edition). O'Reilly Media, 2020.
4. Freeman, A., & Robson, E. Pro React 18. Apress, 2022.
5. Munzner, T. Visualization Analysis and Design. CRC Press, 2014.

Web References:

1. Google AI. Gemini API Documentation. (2024). Retrieved from https://ai.google.dev/gemini-api/docs
2. React Documentation. React – A JavaScript library for building user interfaces. (2024). Retrieved from https://react.dev
3. Recharts. A composable charting library built on React components. (2024). Retrieved from https://recharts.org
4. TypeScript. JavaScript With Syntax For Types. (2024). Retrieved from https://www.typescriptlang.org
5. Vite. Next Generation Frontend Tooling. (2024). Retrieved from https://vitejs.dev