1. INTRODUCTION

1.1. Purpose of the Document

This System Design Document (SDD) provides a comprehensive and detailed blueprint for the design, architecture, and implementation of the CarEase platform. The document serves as a bridge between requirements analysis and system development, outlining how the system will be constructed, integrated, and maintained. It specifies the intended functionalities, technical architecture, design principles, and operational considerations necessary to deliver a robust, scalable, and secure web-based platform for car servicing, detailing, tinting, and maintenance in Kenya.

The SDD includes the following:

- Detailed system architecture and design philosophy.
- Descriptions of all major modules and their interactions.
- Database design, data flow, and control flow diagrams.
- Specifications for user interfaces, APIs, and third-party integrations.
- Non-functional requirements such as security, scalability, and performance.
- Deployment and infrastructure plans.
- Testing strategies and risk mitigation measures.
- Glossary of technical terms and acronyms.

This document is a mandatory deliverable and must be submitted and approved before the commencement of development. All subsequent development, testing, and deployment activities must adhere strictly to the specifications and guidelines set forth herein.

1.2. Scope of the System

CarEase is a web-based platform designed to revolutionize the car servicing industry in Kenya by providing a centralized, technology-driven solution for car washing, detailing, tinting, and general maintenance. The platform will offer on-demand services at the customer's chosen location, leveraging digital tools to enhance transparency, accessibility, efficiency, and environmental sustainability.

Key features and functionalities include:

- **Online Booking:** Customers can schedule car washing, detailing, tinting, and maintenance services online, selecting preferred times and locations.
- **Real-Time GPS Tracking:** Users can monitor the live location and estimated arrival time of service providers.
- **Secure Digital Payments:** Support for cashless transactions via mobile wallets (e.g., M-Pesa), credit/debit cards, and bank transfers, with end-to-end encryption and compliance with PCI-DSS standards.
- **Customer Review and Rating System:** Enables users to provide feedback and rate service providers, promoting transparency and accountability.
- **Automated Notifications and Reminders:** Real-time updates for appointment confirmations, reminders, and service status.
- **AI-Powered Diagnostics and Predictive Maintenance:** Personalized service recommendations and automated maintenance reminders based on user history and vehicle data.
- **Eco-Friendly Service Options:** Selection of water-efficient car washes and biodegradable cleaning products to support sustainability.
- **Service Provider Portal:** Tools for service providers to manage appointments, track earnings, and optimize service delivery.
- **Admin Dashboard:** For platform administrators to manage users, providers, analytics, compliance, and dispute resolution.
- **Cross-Platform Accessibility:** Optimized for both desktop and mobile devices, ensuring wide accessibility.
- **Data Security and Privacy:** Implementation of advanced security protocols, two-factor authentication, and compliance with Kenya Data Protection Act.

Excluded features:

- Major mechanical repairs (e.g., engine overhauls).
- Vehicle towing and roadside assistance.
- Car sales, leasing, or rentals.
- Offline functionality.
- Insurance claims processing.

- Advanced OBD diagnostics (beyond basic IoT).
- Multi-currency support (initial release: KES only).
- Voice command integration.

The system is designed to be scalable and modular, supporting future expansion into additional services, regions, and advanced technologies such as IoT diagnostics and blockchain-based service records.

1.3. Intended Audience

This System Design Document is intended for the following audiences:

- **Developers:** To guide the implementation of all system components according to the specified architecture, modules, and interfaces.
- **Testers/QA Engineers:** To develop and execute test plans that validate functional and non-functional requirements.
- **Project Managers:** To monitor project progress, ensure adherence to design specifications, and coordinate between teams.
- **System Architects:** To review and validate the overall system architecture and integration points.
- **Stakeholders:** Including business owners, investors, and regulatory bodies, to understand the system's capabilities, compliance measures, and value proposition.
- **Platform Administrators:** To manage user onboarding, service provider verification, compliance, and dispute resolution.
- **Regulatory Auditors:** To verify compliance with industry standards, data protection laws, and environmental regulations.
- **Service Providers:** To understand the technical requirements and workflows for integration with the platform.

1.4. Glossary

Term/Acronym	Definition
SDD	System Design Document – a formal document outlining the architecture and design of a system.

Term/Acronym	**Definition**
CarEase	The web-based platform for on-demand car servicing, detailing, tinting, and maintenance.
API	Application Programming Interface – a set of protocols for building and integrating applications.
M-Pesa	A mobile money transfer and payment service widely used in Kenya.
PCI-DSS	Payment Card Industry Data Security Standard – a set of security standards for handling card payments.
IoT	Internet of Things – networked devices that collect and exchange data.
AI	Artificial Intelligence – technology that enables systems to make decisions and predictions.
2FA	Two-Factor Authentication – an extra layer of security for user authentication.
GDPR	General Data Protection Regulation – a regulation on data protection and privacy (EU standard).
NEMA	National Environment Management Authority (Kenya) – regulatory body for environmental standards.
RBAC	Role-Based Access Control – a method of restricting system access to authorized users.
STK Push	SIM ToolKit Push – a mobile payment method used in M-Pesa transactions.
PWA	Progressive Web App – a type of application delivered through the web, built using common web technologies.
UAT	User Acceptance Testing – the process of verifying that a solution works for the user.
JWT	JSON Web Token – a compact, URL-safe means of representing claims to be transferred between two parties.
WCAG	Web Content Accessibility Guidelines – standards for making web content accessible to people with disabilities.
KDPA	Kenya Data Protection Act – legislation governing data privacy and protection in Kenya.

2. System Overview and Design Philosophy

2.1.High-Level Description: Overall Architecture and Tech Stack

The CarEase platform is architected as a modern, cloud-based, web application designed to deliver seamless, real-time car servicing, detailing, tinting, and maintenance experiences to users across Kenya. The architecture is structured to support high availability, modularity, and scalability, ensuring the system can efficiently handle thousands of concurrent users, service providers, and transactions.

Core Architectural Layers

Presentation Layer (Frontend):

- Built using React.js, providing a dynamic, responsive, and interactive user interface optimized for both desktop and mobile devices.
- Employs component-based design, enabling the reuse of UI elements and facilitating rapid development and maintenance.

Application Layer (Backend):

- Developed with Node.js and Express.js, supporting asynchronous, event-driven operations for efficient handling of service bookings, notifications, and payment processing.
- Implements RESTful APIs to facilitate communication between the frontend, backend, and third-party integrations.

Database Layer:

- Utilizes MongoDB as the primary NoSQL database, allowing flexible storage of user profiles, service requests, provider data, and transaction histories.
- Supports horizontal scaling to accommodate growing data volumes and user bases.

AI & Analytics Layer:

- Integrates TensorFlow for AI-driven diagnostics, predictive maintenance, and personalized service recommendations.
- Leverages real-time data analytics to generate insights for both users and service providers.

Cloud Infrastructure:

- Hosted on AWS EC2/S3 for reliable, auto-scaling compute and storage.
- Employs Docker for containerization, ensuring consistent deployment across development, staging, and production environments.

Third-Party Integrations:

- Google Maps API: Real-time GPS tracking and route optimization for service providers.
- M-Pesa API (Daraja) and Stripe: Secure, multi-channel payment processing (mobile money, cards).
- Firebase Authentication: Multi-factor authentication and secure session management.

Admin and Provider Portals:

- Web-based dashboards for administrators (user and provider management, analytics, compliance) and service providers (appointment scheduling, earnings tracking, customer feedback).

System Workflow Overview

1. User Registration and Authentication: Users register via email, phone OTP, or social login. Authentication is handled securely via Firebase.
2. Service Booking: Customers select service type, time, and location via the React.js frontend. Requests are processed by the Node.js backend and stored in MongoDB.
3. Real-Time Tracking: Google Maps API provides live updates on service provider location and estimated arrival.
4. Payment Processing: Users complete transactions through M-Pesa or Stripe, with all payment data encrypted and compliant with PCI-DSS standards.
5. Service Execution and Feedback: Service providers update job status in real-time; users receive notifications and can submit ratings/reviews post-service.
6. AI-Powered Recommendations: TensorFlow analyzes user/service data to generate personalized maintenance reminders and suggestions.

2.2.Design Principles

Modularity

- The system is decomposed into independent modules: user management, booking engine, payment gateway, provider management, AI analytics, and notification services.
- Each module is developed, tested, and deployed independently, promoting separation of concerns and parallel development.

Reusability

- Component-based frontend (React.js) and service-oriented backend (Node.js/Express.js) enable reuse of code across different features and workflows.
- Common utilities (e.g., authentication, notification, data validation) are abstracted into shared libraries, reducing duplication and maintenance overhead.

Scalability

- Horizontal scaling is achieved through cloud deployment (AWS EC2/S3) and MongoDB's sharding capabilities, supporting growth in user base and data volume.
- Stateless backend APIs and containerized microservices (Docker) allow for rapid scaling and deployment of new features without downtime.

Performance

- Asynchronous processing in Node.js ensures non-blocking operations, supporting high concurrency for booking, payment, and notification workflows.
- Caching strategies (e.g., Redis, in-memory caches) are employed to reduce response times for frequently accessed data.
- Database indexing and optimized queries enhance data retrieval speed, meeting the platform's sub-2-second response time requirement**.**

Security

- End-to-end encryption (TLS/SSL) is enforced for all data in transit; sensitive data at rest is encrypted in MongoDB.
- Role-Based Access Control (RBAC) restricts system access based on user roles (customer, provider, admin), ensuring data privacy and integrity.
- Multi-factor authentication (Firebase), secure payment gateways (PCI-DSS compliance), and regular security audits protect against unauthorized access and data breaches.
- Compliance with Kenya Data Protection Act (KDPA) and global standards (GDPR, PCI-DSS) is maintained throughout the platform.

Maintainability and Extensibility

- Modular architecture and well-documented APIs facilitate easy updates, debugging, and integration of new features or third-party services.
- Automated testing (unit, integration, system) and CI/CD pipelines (GitHub Actions, Jenkins) ensure code quality and rapid deployment of patches and enhancements.

Observability and Monitoring

- Centralized logging, real-time monitoring, and alerting (using ELK Stack, Grafana, or Prometheus) enable proactive detection of issues and performance bottlenecks.
- System health dashboards support continuous improvement and operational transparency.

Usability and Accessibility

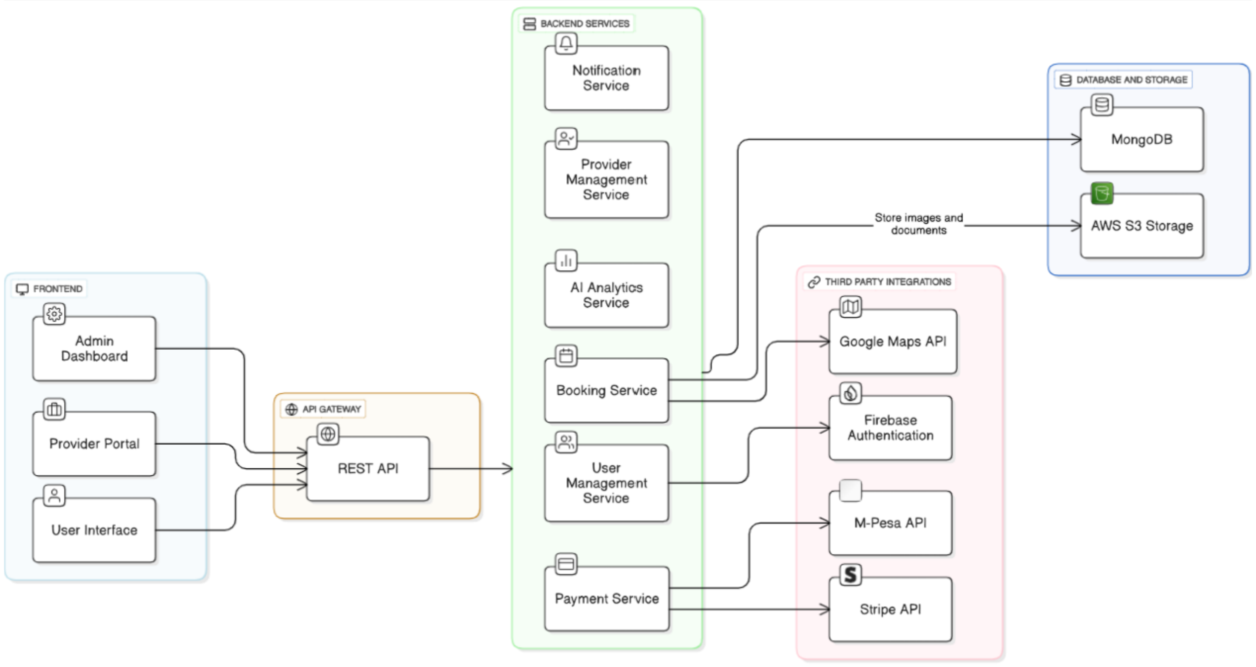
- User interfaces are designed to be intuitive and accessible, conforming to WCAG 2.1 AA standards.
- Responsive design ensures seamless experience across devices and platforms.

3. Architectural Design

3.1. Architecture Diagram

The CarEase system adopts a layered, client-server, REST-based architecture with clear separation between frontend, backend, database, and third-party integrations. The following UML Component Diagram illustrates the major modules and their interactions.

1.0 Component Diagram



3.2. Architecture Style

- ➢ Layered: The system is divided into presentation, application, data, and integration layers, ensuring separation of concerns and maintainability.
- ➢ Client-Server: The frontend (client) communicates with the backend (server) via RESTful APIs.
- ➢ REST-Based: All communication between frontend and backend, as well as third-party integrations, uses RESTful HTTP endpoints.
- ➢ Microservices-inspired (modular): While not pure microservices, the backend is structured into independent modules/services for scalability and ease of maintenance.

3.3 Component Description

Frontend vs Backend Separation

Frontend:

- ➢ Built with React.js.
- ➢ Includes three main applications: User Interface (for customers), Provider Portal (for service providers), and Admin Dashboard (for administrators).
- ➢ Handles user input, displays data, and communicates with backend services exclusively via the REST API.

Backend:

- ➢ Built with Node.js and Express.js.
- ➢ Exposes a unified REST API that acts as the API Gateway for all frontend clients.
- ➢ Implements business logic across modular services:
  - User Management Service: Handles registration, authentication, profile management, and RBAC.
  - Booking Service: Manages service scheduling, status tracking, and real-time updates.
  - Payment Service: Orchestrates payment initiation, confirmation, and reconciliation with M-Pesa and Stripe.
  - Notification Service: Sends email, SMS, and push notifications for confirmations, reminders, and status changes.
  - AI Analytics Service: Provides predictive maintenance, personalized recommendations, and service insights.
  - Provider Management Service: Supports onboarding, verification, and management of service providers

API Gateway

- ➢ The REST API (Express.js) serves as the single entry point for all client requests, routing them to the appropriate backend services.
- ➢ Handles authentication (via Firebase), request validation, and error handling.

Database and Storage Choices

MongoDB:

- ➢ Stores user profiles, booking records, service provider data, payment transactions, and analytics data.
- ➢ Chosen for its schema flexibility and scalability.

AWS S3 Storage:

- Used for storing images (before/after service, provider documents), receipts, and other binary assets.

Third-Party Integrations

M-Pesa API (Daraja):

- Enables mobile money payments, STK push, and payment reconciliation.

Stripe API:

- Supports credit/debit card payments for customers who prefer card transactions.

Google Maps API:

- Provides real-time GPS tracking, route optimization, and geolocation services for service providers.

Firebase Authentication:

- Manages secure user authentication (email/password, OTP, social login) and multi-factor authentication.

Component Interactions

User Journey Example:

- A customer logs in via the frontend (React.js), which authenticates through Firebase.
- The customer books a service; the booking request is sent to the REST API.
- The Booking Service creates a new booking record in MongoDB and requests real-time location data from Google Maps API.
- The Payment Service initiates an M-Pesa STK push or Stripe transaction.
- Upon payment confirmation, the Notification Service sends updates to the customer and provider.
- The Provider Portal allows service providers to accept jobs, update status, and upload service completion images to AWS S3.
- The AI Analytics Service analyzes booking and vehicle data to provide the customer with predictive maintenance suggestions.

4. Detailed Design

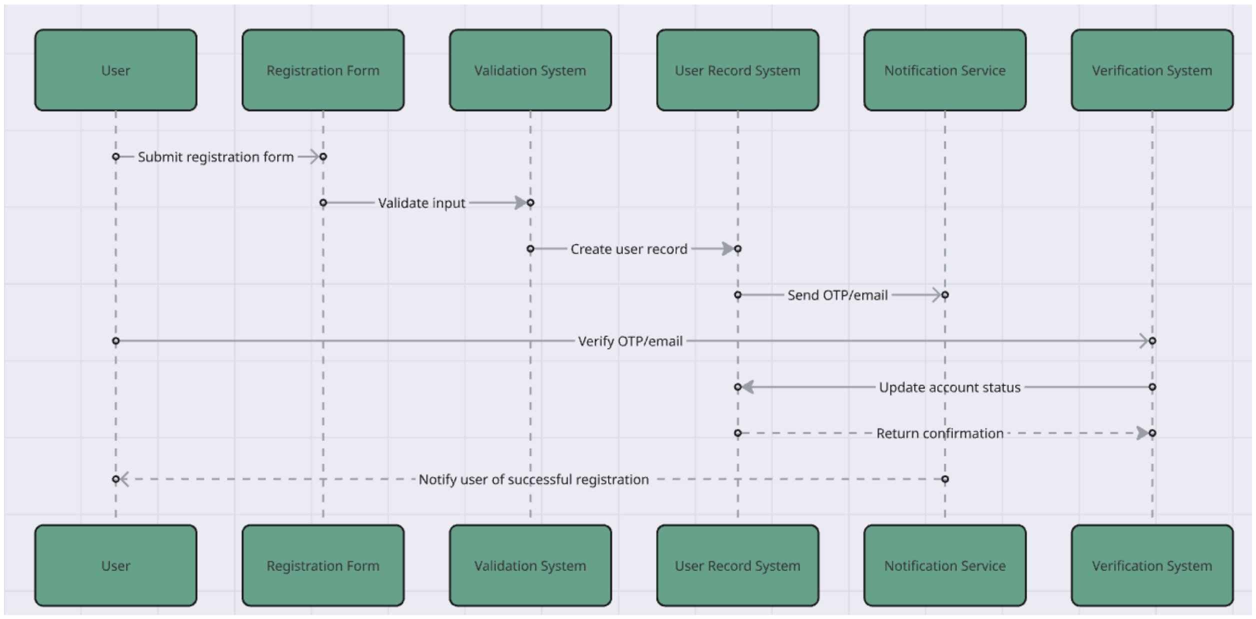
4.1. Module/Feature Descriptions

Below, each core module is broken down in detail, including its functionality, inputs/outputs, business rules, and data elements.

A. User Management Module

- **Functionality:**
  Handles registration, authentication (including OTP and social login), profile management, password resets, and user role assignment (customer, provider, admin).
- **Inputs:**
  - Registration form data (name, phone, email, password, vehicle info)
  - Login credentials
  - OTP codes (for phone/email verification)
  - Profile update requests
- **Outputs:**
  - JWT authentication tokens
  - User profile data
  - Success/error messages
  - Password reset emails/OTPs
- **Business Rules:**
  - All emails and phone numbers must be unique.
  - Passwords must meet minimum security criteria (length, character types).
  - Users must verify their account via OTP or email link before accessing services.
  - Roles are strictly enforced (RBAC).
- **Data Elements Involved:**
  - UserID (UUID)
  - Name
  - Email
  - Phone
  - Password hash
  - Role (customer, provider, admin)

- Profile photo URL
- Vehicle details (make, model, year, license plate)
- Account status (active, suspended, pending verification)



1.1 User Sequence Diagram

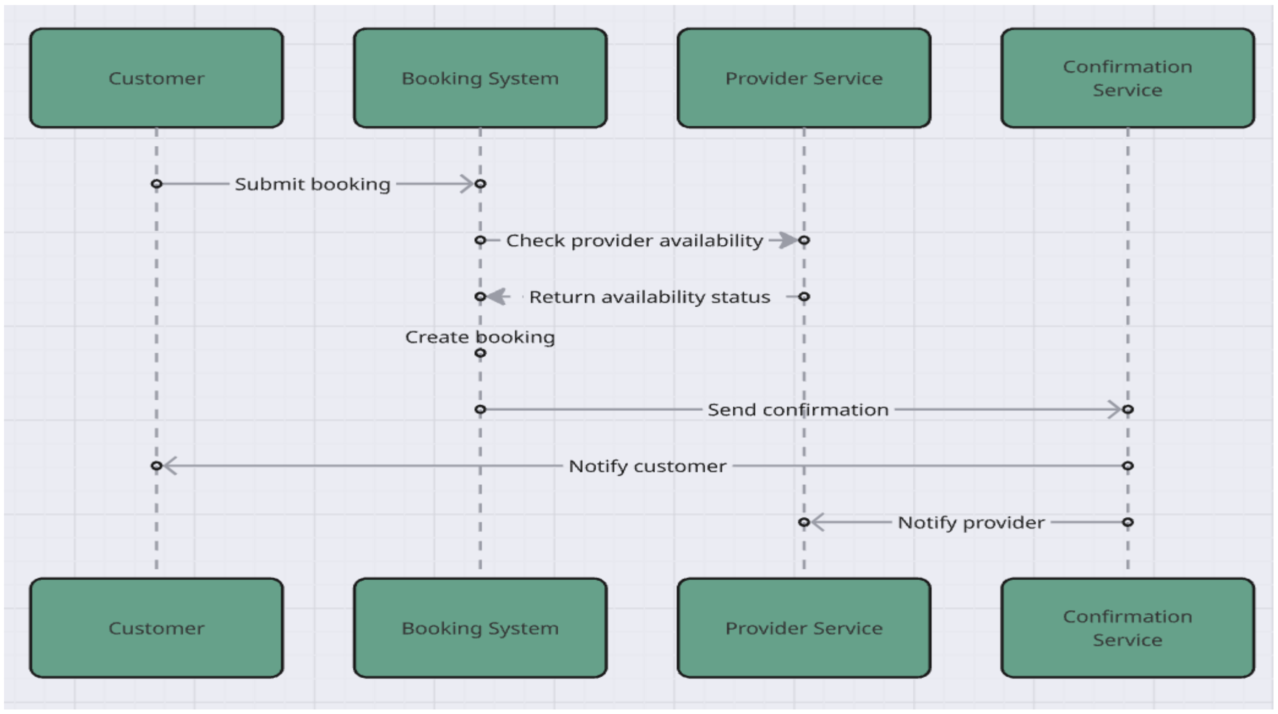
B. Booking & Scheduling Module

- **Functionality:**
  Allows customers to book car services, select service type, date, time, and location. Handles provider assignment, booking status updates, rescheduling, and cancellations.
- **Inputs:**
  - Booking request (service type, desired time/date, location, vehicle info)
  - Provider availability
  - Reschedule/cancel requests
- **Outputs:**
  - Booking confirmation
  - Booking status updates (pending, confirmed, in-progress, completed, cancelled)
  - Notifications to customer and provider
- **Business Rules:**

- Bookings must be made at least 1 hour in advance.
- Providers can only be assigned if available at requested slot.
- Customers can reschedule/cancel up to 30 minutes before appointment.
- No double-booking for same provider/time slot.

➢ **Data Elements Involved:**

- BookingID
- CustomerID
- ProviderID
- ServiceType
- ScheduledDateTime
- Location (GPS/address)
- Status
- Timestamps (created, updated, completed, cancelled)

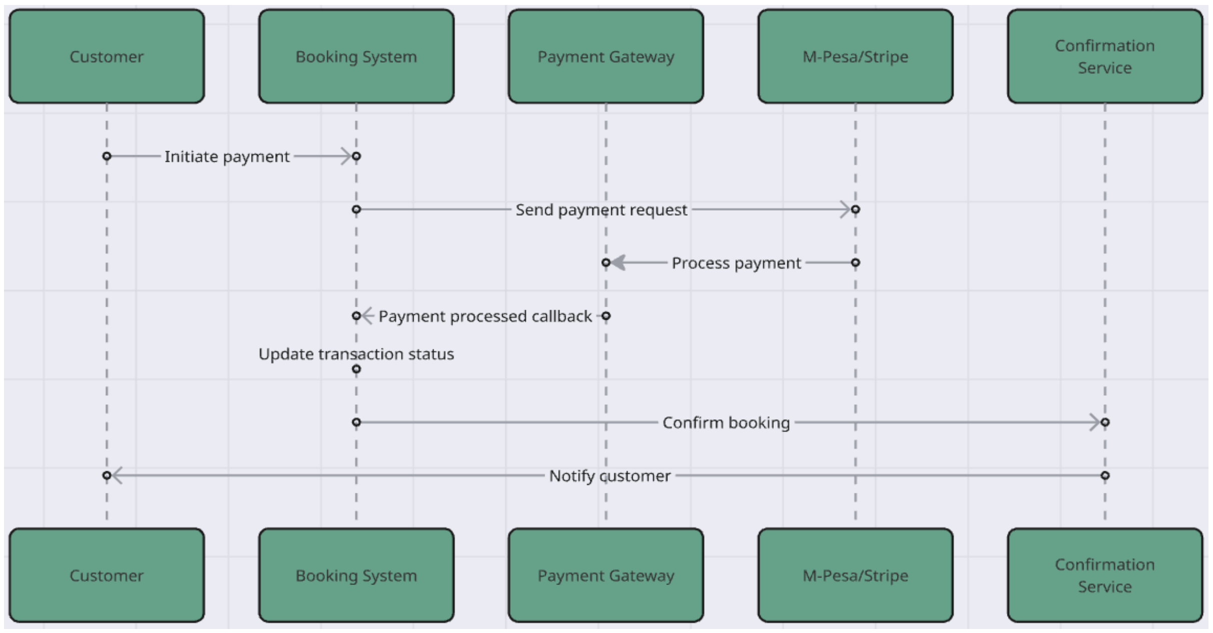


1.2 Booking & Scheduling Sequence Diagram

C. Payment Module

➢ **Functionality:**
Manages payment initiation, processing (M-Pesa, Stripe), confirmation, refunds, and transaction history.

- **Inputs:**
  - Payment initiation (bookingID, amount, payment method)
  - M-Pesa/Stripe transaction callbacks
  - Refund requests
- **Outputs:**
  - Payment confirmation/receipt
  - Transaction status updates
  - Error/failure messages
- **Business Rules:**
  - Payment must be completed before service is confirmed.
  - Refunds allowed only for cancelled/unfulfilled bookings.
  - All transactions logged with unique transaction IDs.
- **Data Elements Involved:**
  - TransactionID
  - BookingID
  - CustomerID
  - Amount
  - PaymentMethod
  - Status (pending, completed, failed, refunded)
  - Timestamp

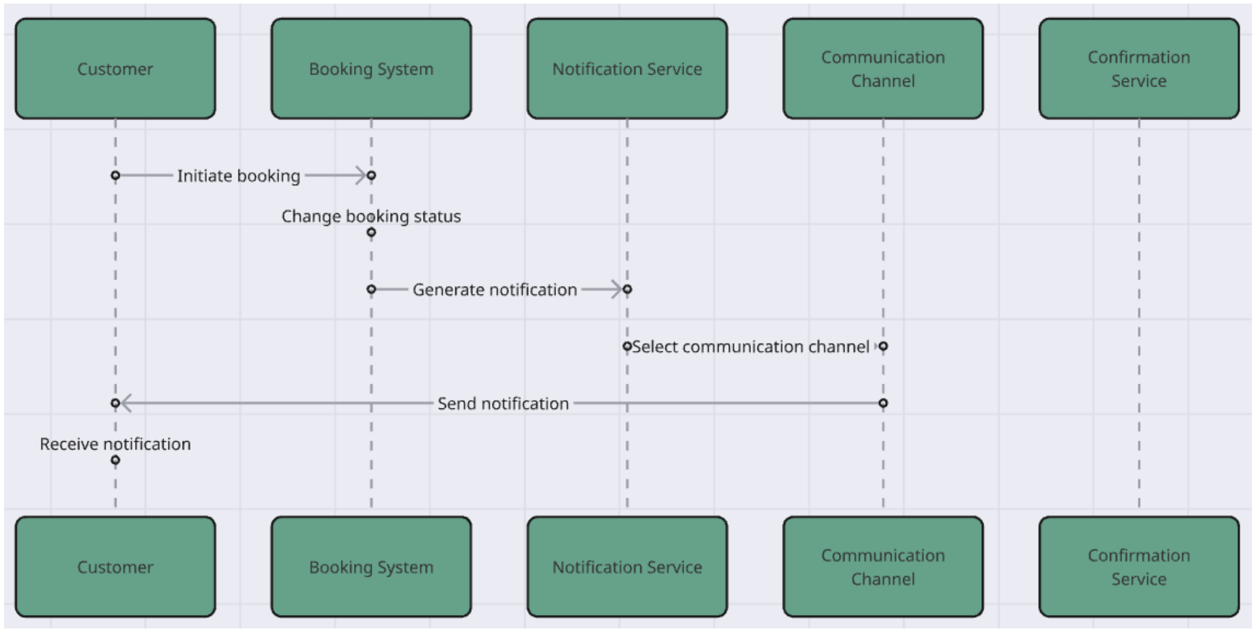


1.3 Payment Sequence Diagram

D. Notification Module

- **Functionality:**
  Sends real-time notifications (SMS, email, in-app) for booking confirmations, reminders, status changes, and promotions.
- **Inputs:**
  - Trigger events (booking created, status changed, payment received, etc.)
- **Outputs:**
  - SMS messages
  - Emails
  - Push/in-app notifications
- **Business Rules:**
  - All critical updates (booking, payment, cancellations) must trigger notifications.
  - Users can opt-in/out of marketing notifications.
- **Data Elements Involved:**
  - NotificationID
  - UserID

- Channel (SMS/email/app)
- Message content
- Status (sent, failed)
- Timestamp



1.4 Notification Sequence Diagram

E. Service Provider Management Module

➢ **Functionality:**
Manages provider onboarding, verification, scheduling, earnings, and ratings.

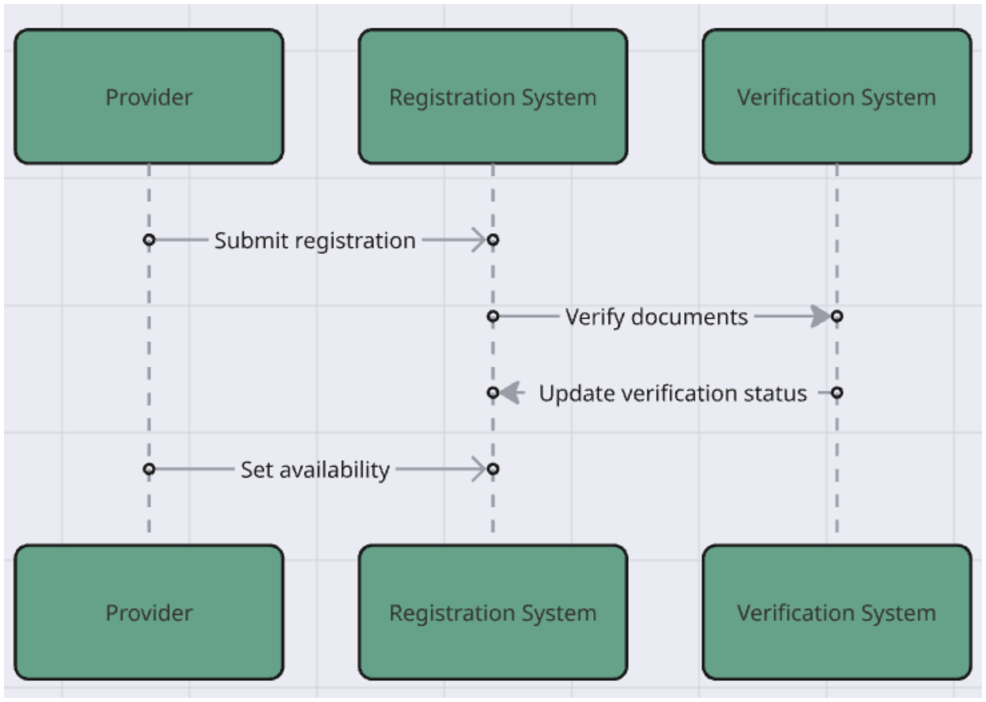
➢ **Inputs:**

- Provider registration data
- Verification documents
- Availability schedules
- Service completion updates

➢ **Outputs:**

- Provider profile status
- Earnings reports
- Ratings/feedback summaries

- **Business Rules:**
  - Providers must submit valid documents for verification.
  - Only verified providers can accept bookings.
  - Providers must update job status in real-time.
- **Data Elements Involved:**
  - ProviderID
  - Name
  - Contact details
  - Verification status
  - Availability schedule
  - Earnings
  - Ratings



1.5 Service Provider Sequence Diagram

F. AI & Analytics Module

- **Functionality:**
  Provides predictive maintenance reminders, analyzes service usage, and generates personalized recommendations.
- **Inputs:**

- User service history
- Vehicle data
- Booking patterns

➢ **Outputs:**

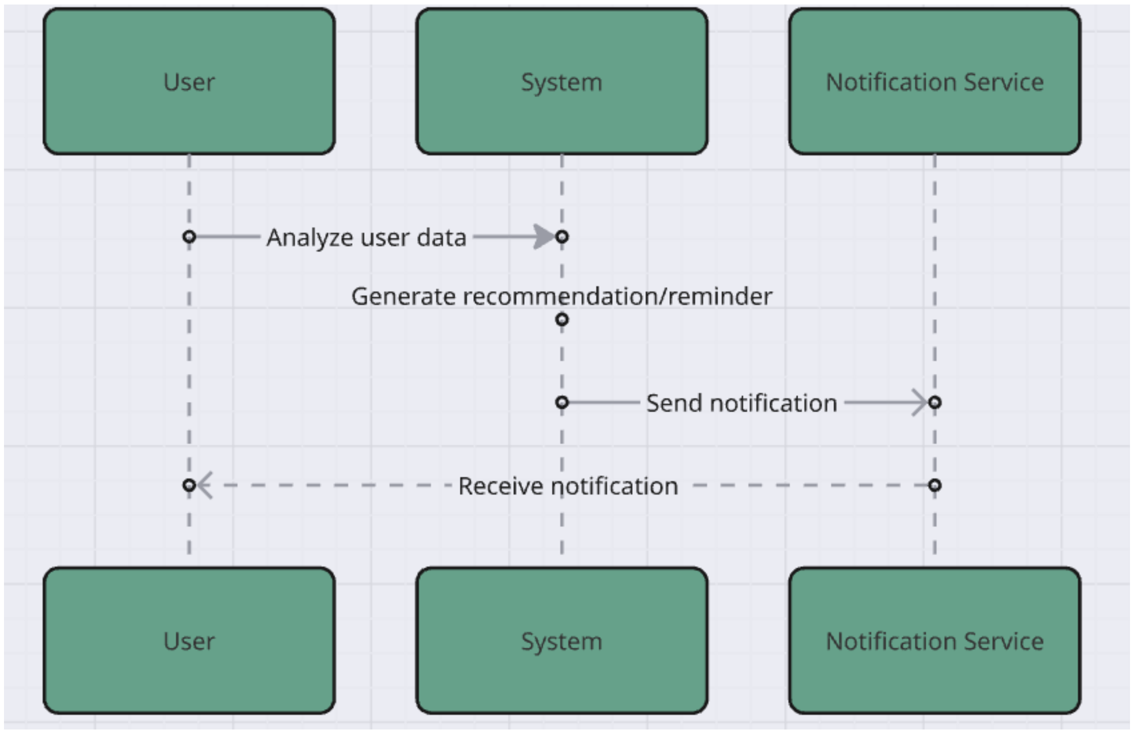
  - Maintenance reminders
  - Usage analytics
  - Service recommendations

➢ **Business Rules:**

- Reminders sent based on time/mileage/service type.
- Recommendations personalized per user history.

➢ **Data Elements Involved:**

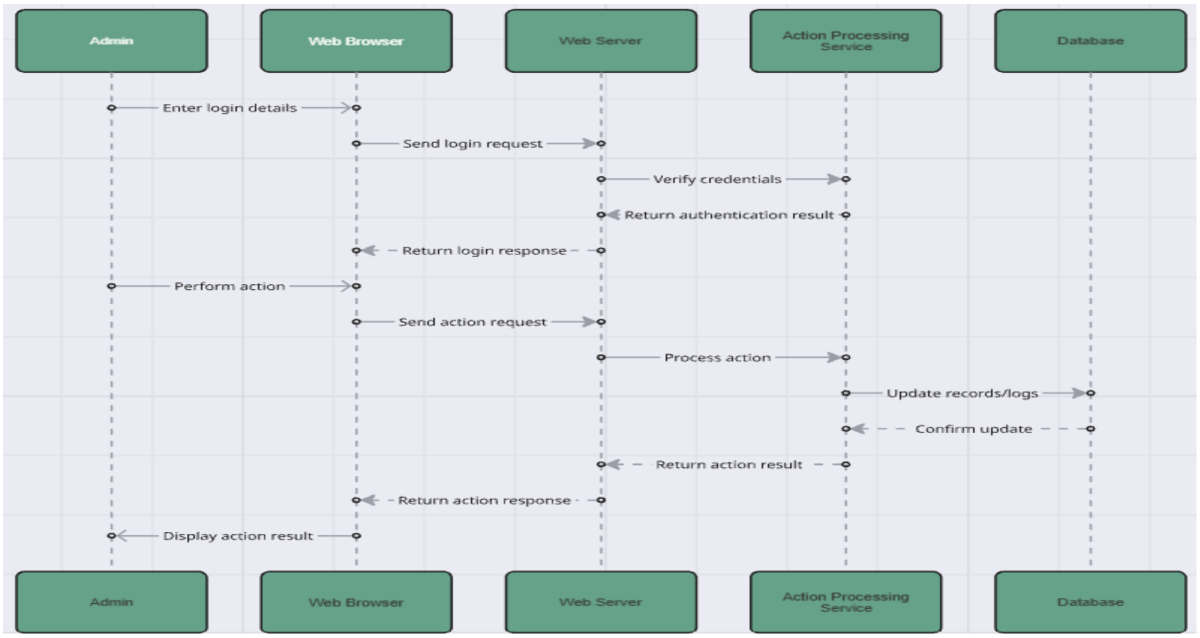
- AnalyticsID
- UserID
- VehicleID
- ServiceHistory
- Recommendation



1.6 AI & Analytics Sequence Diagram

G. Admin Dashboard Module

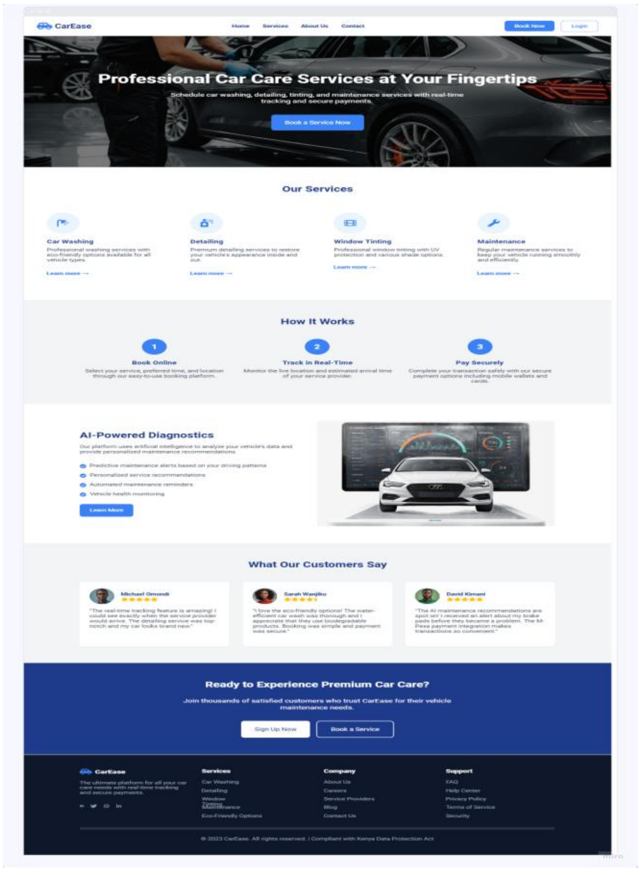
- **Functionality:**
  Allows admins to manage users, providers, bookings, disputes, and view analytics.
- **Inputs:**
  - Admin login
  - Management actions (approve, suspend, resolve dispute, etc.)
- **Outputs:**
  - System reports
  - User/provider status updates
  - Dispute resolutions
- **Business Rules:**
  - Only admins can access dashboard features.
  - All actions logged for audit.
- **Data Elements Involved:**
  - AdminID
  - User/provider records
  - Action logs
  - Analytics data



1.7 Admin Sequence Diagram

4.2 Interface Design

A. User Interface (UI) Mockups / Wireframes



Professional Car Care Services at Your Fingertips

Schedule car washing, detailing, tinting, and maintenance services with real-time tracking and secure payments.

Our Services

Car Washing
Professional washing services with eco-friendly options available for all vehicle types.

Detailing
Premium detailing services to restore your vehicle's appearance inside and out.

Window Tinting
Professional window tinting with UV protection and various shade options.

Maintenance
Regular maintenance services to keep your vehicle running smoothly and efficiently.

How It Works

1 Book Online
Select your service, preferred time, and location through our easy-to-use booking platform.

2 Track in Real-Time
Monitor the live location and estimated arrival time of your service provider.

3 Pay Securely
Complete your transaction safely with our secure payment options including mobile wallets and cards.

AI-Powered Diagnostics

Our platform uses artificial intelligence to analyze your vehicle's data and provide personalized maintenance recommendations.

- Predictive maintenance alerts based on your driving patterns
- Personalized service recommendations
- Automated maintenance reminders
- Vehicle health monitoring

What Our Customers Say

Michael Omondi ★★★★★
"The real-time tracking feature is amazing! I could see exactly when the service provider would arrive. The detailing service was top-notch and my car looks brand new."

Sarah Wanjiku ★★★★½
"I love the eco-friendly options! The water-efficient car wash was thorough and I appreciate that they use biodegradable products. Booking was simple and payment was secure."

David Kimani ★★★★★
"The AI maintenance recommendations are spot on! I received an alert about my brake pads before they became a problem. The M-Pesa payment integration makes transactions so convenient."

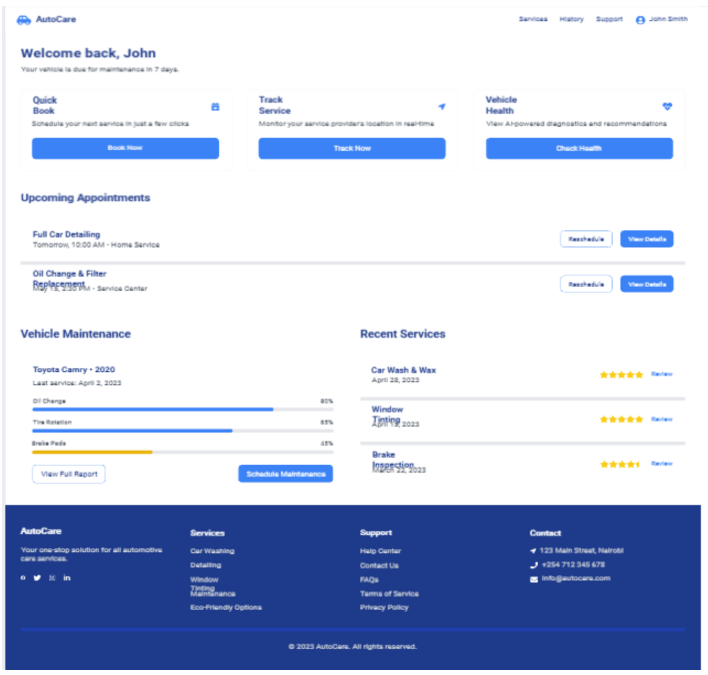
Ready to Experience Premium Car Care?

Join thousands of satisfied customers who trust CarEase for their vehicle maintenance needs.

CarEase
The ultimate platform for all your car care needs with real-time tracking and secure payments.

Services	Company	Support
Car Washing	About Us	FAQ
Detailing	Careers	Help Center
Window Tinting	Service Providers	Privacy Policy
Maintenance	Blog	Terms of Service
Eco-Friendly Options	Contact Us	Security

© 2023 CarEase. All rights reserved. | Compliant with Kenya Data Protection Act



Welcome back, John

Your vehicle is due for maintenance in 7 days.

Quick Book
Schedule your next service in just a few clicks

Track Service
Monitor your service provider's location in real-time

Vehicle Health
View AI-powered diagnostics and recommendations

Upcoming Appointments

Full Car Detailing
Tomorrow, 10:00 AM - Home Service

Oil Change & Filter Replacement
May 15, 2:30 PM - Service Center

Vehicle Maintenance

Toyota Camry • 2020
Last service: April 2, 2023

Oil Change	80%
Tire Rotation	65%
Brake Pads	40%

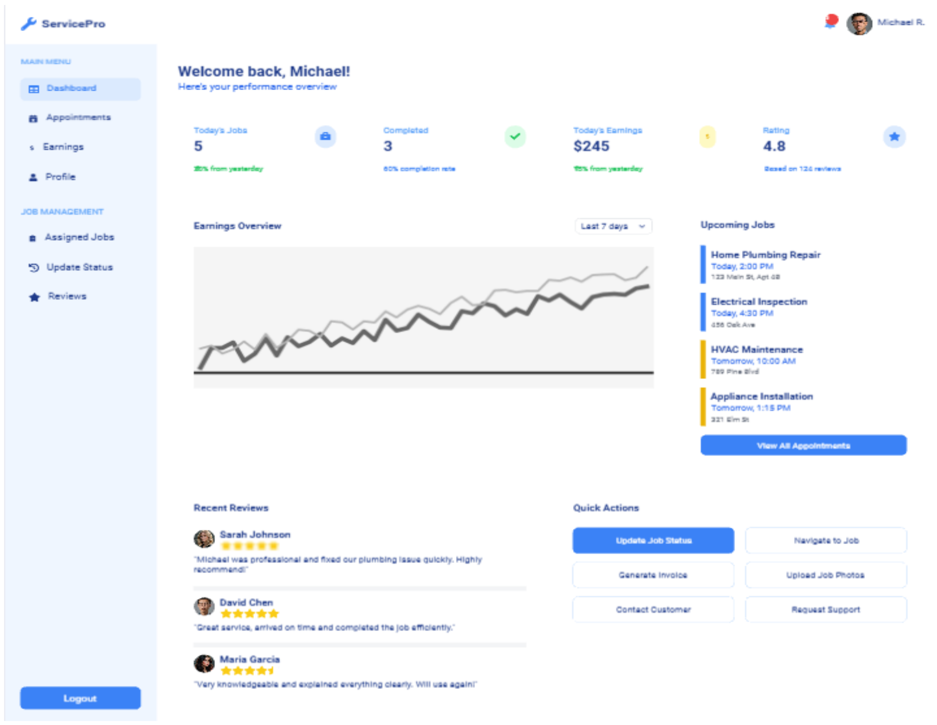
Recent Services

Car Wash & Wax April 28, 2023	★★★★★	Review
Window Tinting April 15, 2023	★★★★★	Review
Brake Inspection March 22, 2023	★★★★½	Review

AutoCare
Your one-stop solution for all automotive care services.

Services	Support	Contact
Car Washing	Help Center	123 Main Street, Nairobi
Detailing	Contact Us	+254 712 345 678
Window Tinting	FAQs	info@autocare.com
Maintenance	Terms of Service	
Eco-Friendly Options	Privacy Policy	

© 2023 AutoCare. All rights reserved.


ServicePro

Michael R.

MAIN MENU
- Dashboard
- Appointments
- Earnings
- Profile

JOB MANAGEMENT
- Assigned Jobs
- Update Status
- Reviews

Logout

Welcome back, Michael!
Here's your performance overview

Today's Jobs	Completed	Today's Earnings	Rating
5	3	$245	4.8
20% from yesterday	60% completion rate	15% from yesterday	Based on 124 reviews

Earnings Overview — Last 7 days

Upcoming Jobs

- **Home Plumbing Repair** — Today, 2:00 PM — 123 Main St, Apt 4B
- **Electrical Inspection** — Today, 4:30 PM — 456 Oak Ave
- **HVAC Maintenance** — Tomorrow, 10:00 AM — 789 Pine Blvd
- **Appliance Installation** — Tomorrow, 1:15 PM — 321 Elm St

View All Appointments

Recent Reviews

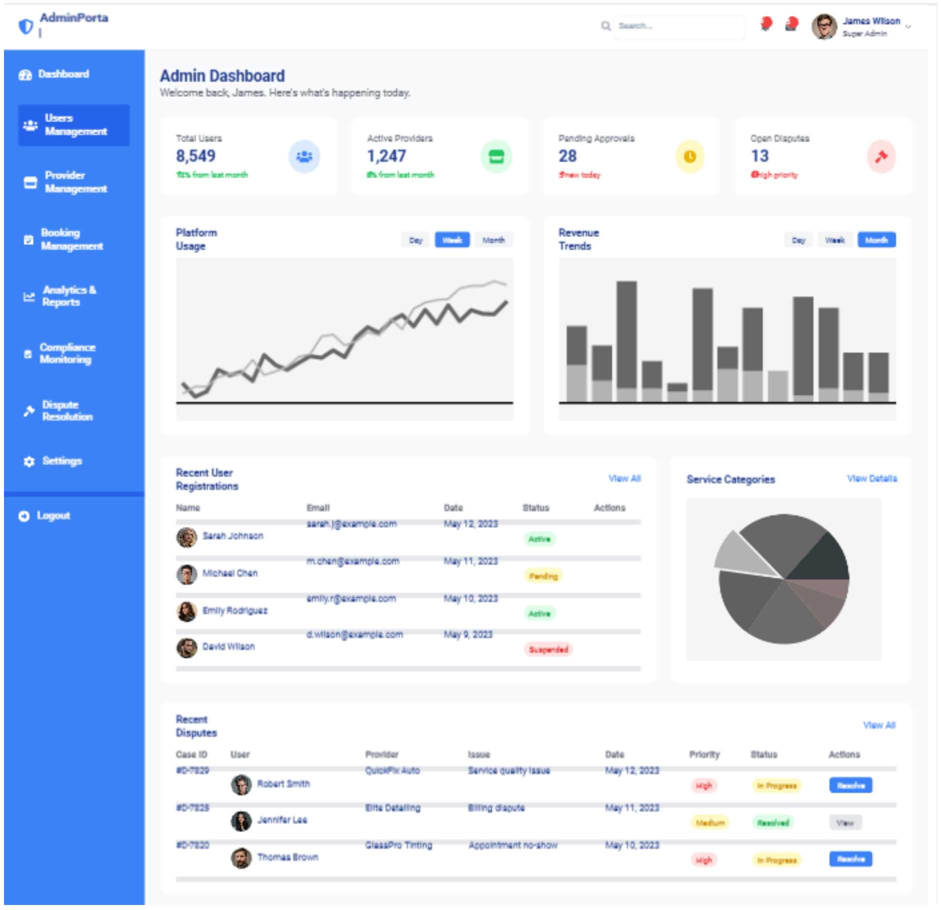
Sarah Johnson ★★★★★
"Michael was professional and fixed our plumbing issue quickly. Highly recommend!"

David Chen ★★★★★
"Great service, arrived on time and completed the job efficiently."

Maria Garcia ★★★★½
"Very knowledgeable and explained everything clearly. Will use again!"

Quick Actions

Update Job Status · Navigate to Job · Generate Invoice · Upload Job Photos · Contact Customer · Request Support


AdminPorta

Search...

James Wilson
Super Admin

- Dashboard
- Users Management
- Provider Management
- Booking Management
- Analytics & Reports
- Compliance Monitoring
- Dispute Resolution
- Settings
- Logout

Admin Dashboard
Welcome back, James. Here's what's happening today.

Total Users	Active Providers	Pending Approvals	Open Disputes
8,549	1,247	28	13
12% from last month	8% from last month	5 new today	3 high priority

Platform Usage — Day · Week · Month

Revenue Trends — Day · Week · Month

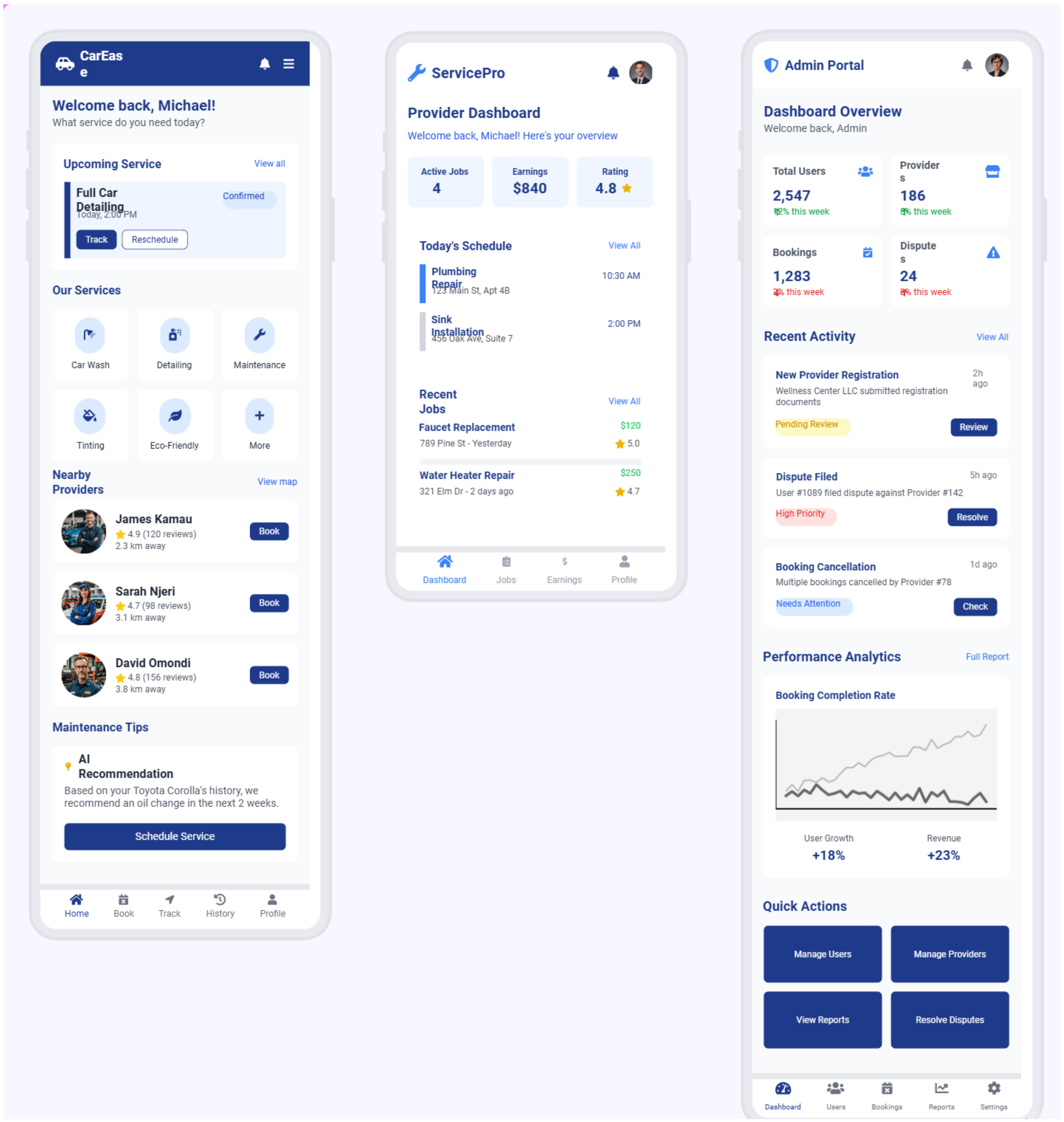
Recent User Registrations — View All

Name	Email	Date	Status	Actions
Sarah Johnson	sarah.j@example.com	May 12, 2023	Active	
Michael Chen	m.chen@example.com	May 11, 2023	Pending	
Emily Rodriguez	emily.r@example.com	May 10, 2023	Active	
David Wilson	d.wilson@example.com	May 9, 2023	Suspended	

Service Categories — View Details

Recent Disputes — View All

Case ID	User	Provider	Issue	Date	Priority	Status	Actions
#D-7829	Robert Smith	QuickFix Auto	Service quality issue	May 12, 2023	High	In Progress	Resolve
#D-7825	Jennifer Lee	Elite Detailing	Billing dispute	May 11, 2023	Medium	Resolved	View
#D-7820	Thomas Brown	GlassPro Tinting	Appointment no-show	May 10, 2023	High	In Progress	Resolve


CarEas
e

Welcome back, Michael!

What service do you need today?

Upcoming Service — View all

Full Car Detailing — Confirmed

Today, 2:00 PM

Track | Reschedule

Our Services

Car Wash | Detailing | Maintenance

Tinting | Eco-Friendly | More

Nearby Providers — View map

James Kamau — 4.9 (120 reviews) — 2.3 km away — Book

Sarah Njeri — 4.7 (98 reviews) — 3.1 km away — Book

David Omondi — 4.8 (156 reviews) — 3.8 km away — Book

Maintenance Tips

AI Recommendation

Based on your Toyota Corolla's history, we recommend an oil change in the next 2 weeks.

Schedule Service

Home | Book | Track | History | Profile

ServicePro

Provider Dashboard

Welcome back, Michael! Here's your overview

Active Jobs	Earnings	Rating
4	$840	4.8

Today's Schedule — View All

Plumbing Repair — 10:30 AM

123 Main St, Apt 4B

Sink Installation — 2:00 PM

456 Oak Ave, Suite 7

Recent Jobs — View All

Faucet Replacement — $120

789 Pine St - Yesterday — 5.0

Water Heater Repair — $250

321 Elm Dr - 2 days ago — 4.7

Dashboard | Jobs | Earnings | Profile

Admin Portal

Dashboard Overview

Welcome back, Admin

Total Users	Providers
2,547	186
12% this week	5% this week

Bookings	Disputes
1,283	24
3% this week	8% this week

Recent Activity — View All

New Provider Registration — 2h ago

Wellness Center LLC submitted registration documents

Pending Review — Review

Dispute Filed — 5h ago

User #1089 filed dispute against Provider #142

High Priority — Resolve

Booking Cancellation — 1d ago

Multiple bookings cancelled by Provider #78

Needs Attention — Check

Performance Analytics — Full Report

Booking Completion Rate

User Growth	Revenue
+18%	+23%

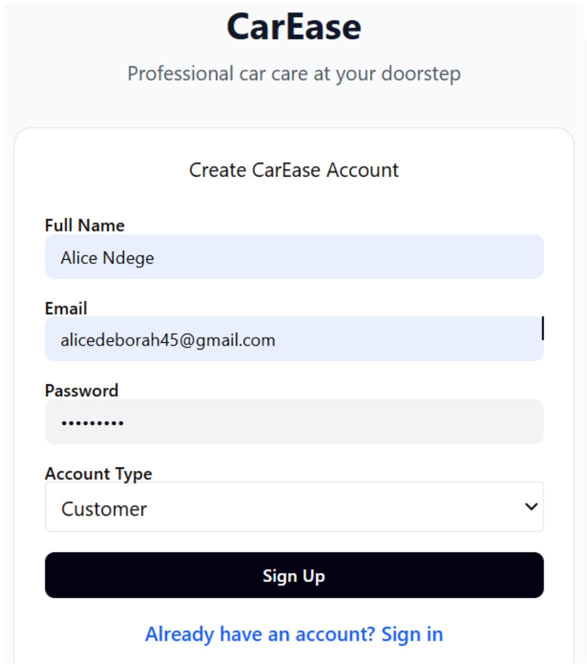
Quick Actions

Manage Users | Manage Providers

View Reports | Resolve Disputes

Dashboard | Users | Bookings | Reports | Settings

B. Navigation Flow (for Web/Mobile Apps)


CarEase

Professional car care at your doorstep

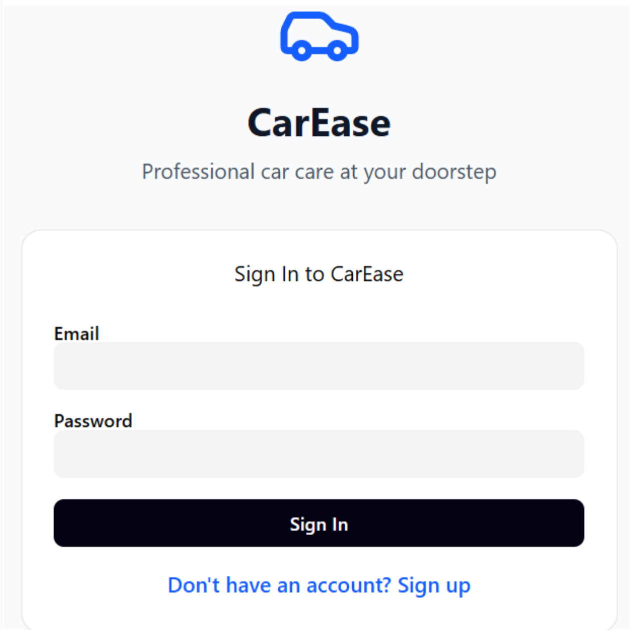
Create CarEase Account

Full Name: Alice Ndege

Email: alicedeborah45@gmail.com

Password: •••••••••

Account Type: Customer

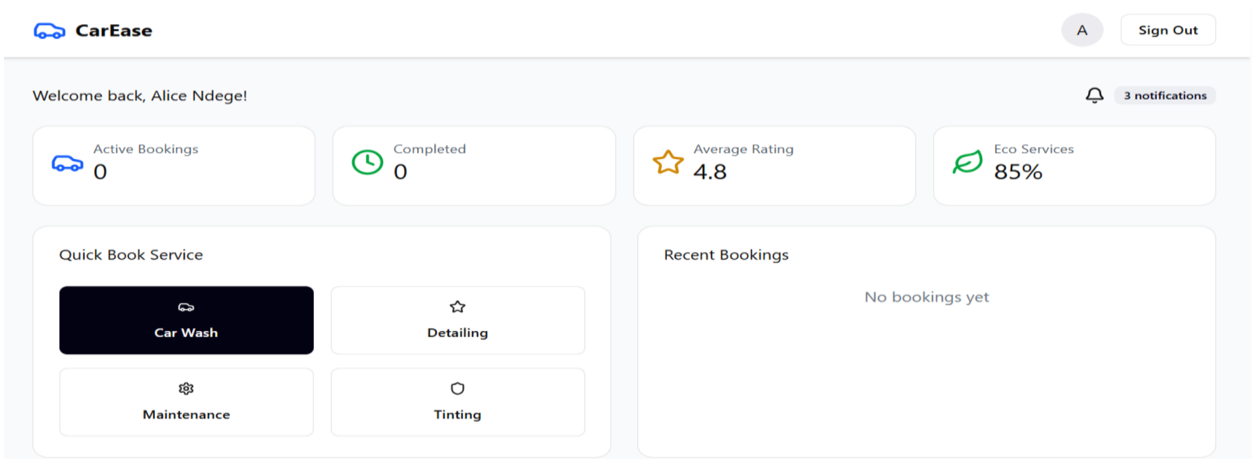

CarEase

Professional car care at your doorstep

Sign In to CarEase

Email

Password



Welcome back, Alice Ndege!

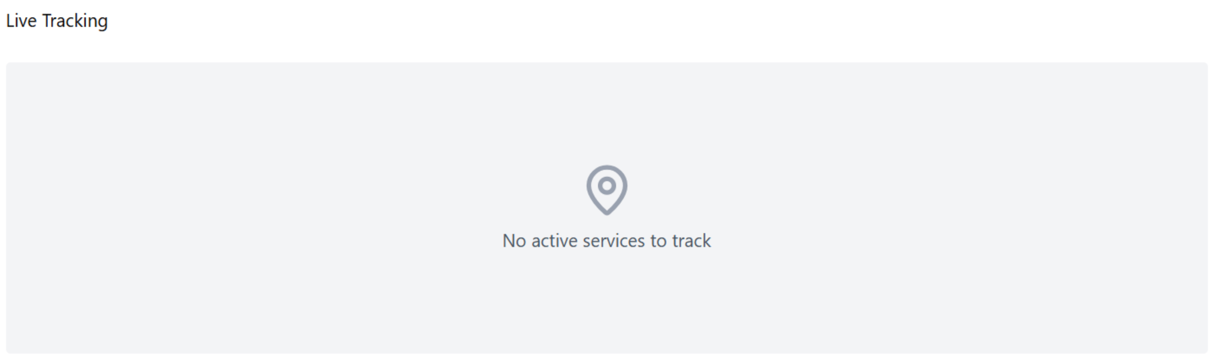
3 notifications

Active Bookings	0
Completed	0
Average Rating	4.8
Eco Services	85%

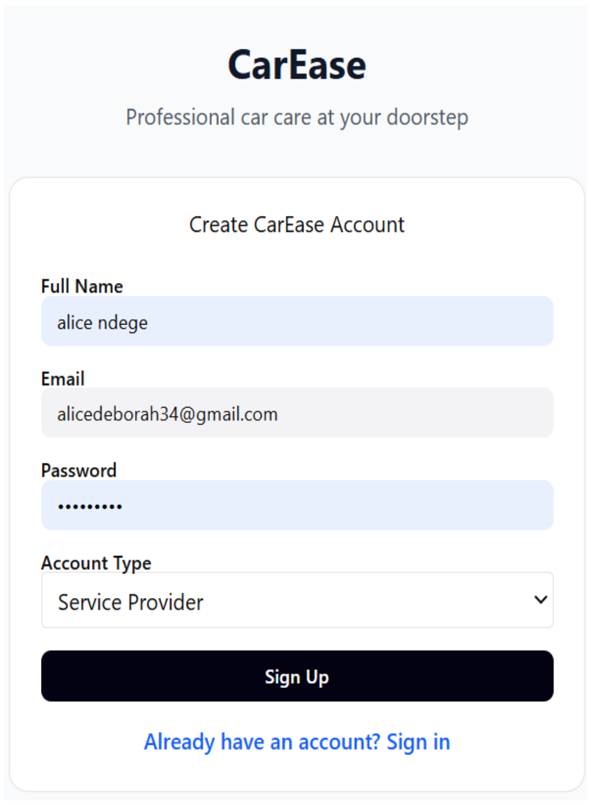
Quick Book Service

Recent Bookings

No bookings yet


Live Tracking

No active services to track


CarEase

Professional car care at your doorstep

Create CarEase Account

Full Name: alice ndege

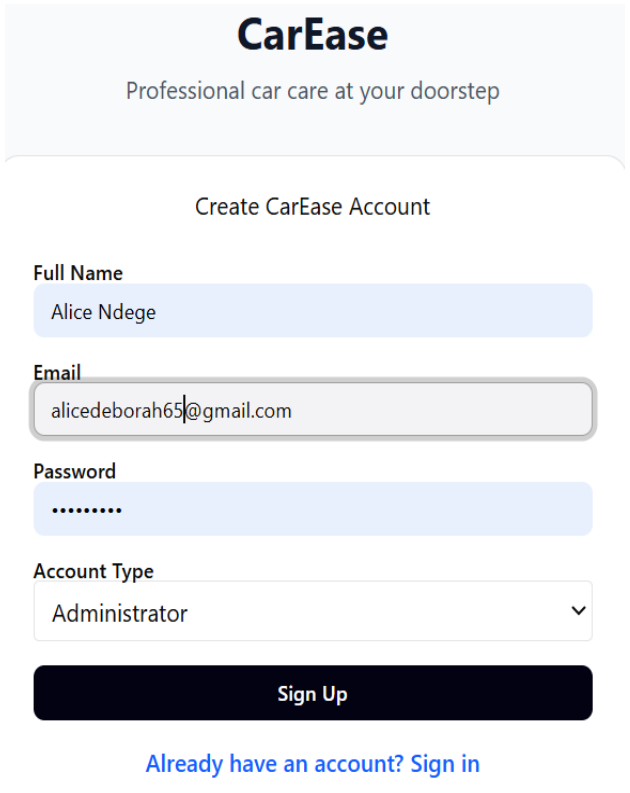
Email: alicedeborah34@gmail.com

Password: •••••••••

Account Type: Service Provider

Sign Up

Already have an account? Sign in


CarEase

Professional car care at your doorstep

Create CarEase Account

Full Name: Alice Ndege

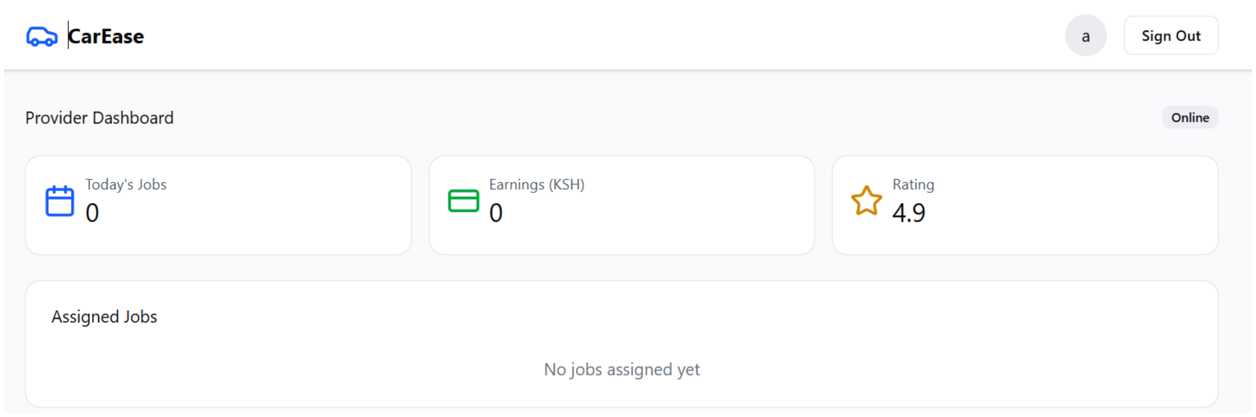
Email: alicedeborah65@gmail.com

Password: •••••••••

Account Type: Administrator

Sign Up

Already have an account? Sign in


CarEase a Sign Out

Provider Dashboard Online

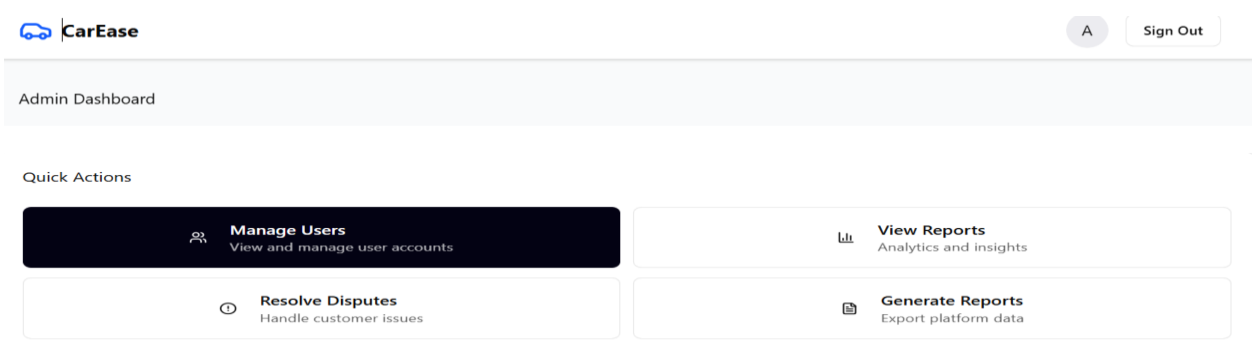
Today's Jobs 0

Earnings (KSH) 0

Rating 4.9

Assigned Jobs

No jobs assigned yet


CarEase A Sign Out

Admin Dashboard

Quick Actions

Manage Users
View and manage user accounts

View Reports
Analytics and insights

Resolve Disputes
Handle customer issues

Generate Reports
Export platform data

C. API Interface Specifications

Example: Booking API

Endpoint	Method	Request Schema	Response Schema	Status Codes
/api/bookings	POST	{ userId, serviceType, dateTime, location, vehicleId }	{ bookingId, status, providerId, confirmationMessage }	201, 400, 401, 409
/api/bookings/:id	GET	N/A	{ bookingId, userId, providerId, serviceType, status, dateTime, location }	200, 404, 401
/api/payments	POST	{ bookingId, amount, paymentMethod }	{ transactionId, status, receiptUrl }	201, 400, 402, 500
/api/users	POST	{ name, email, phone, password, vehicleDetails }	{ userId, status, verificationRequired }	201, 400, 409
/api/providers	POST	{ name, contactDetails, documents }	{ providerId, status, verificationStatus }	201, 400, 409
/api/notifications	POST	{ userId, channel, message }	{ notificationId, status }	201, 400, 401

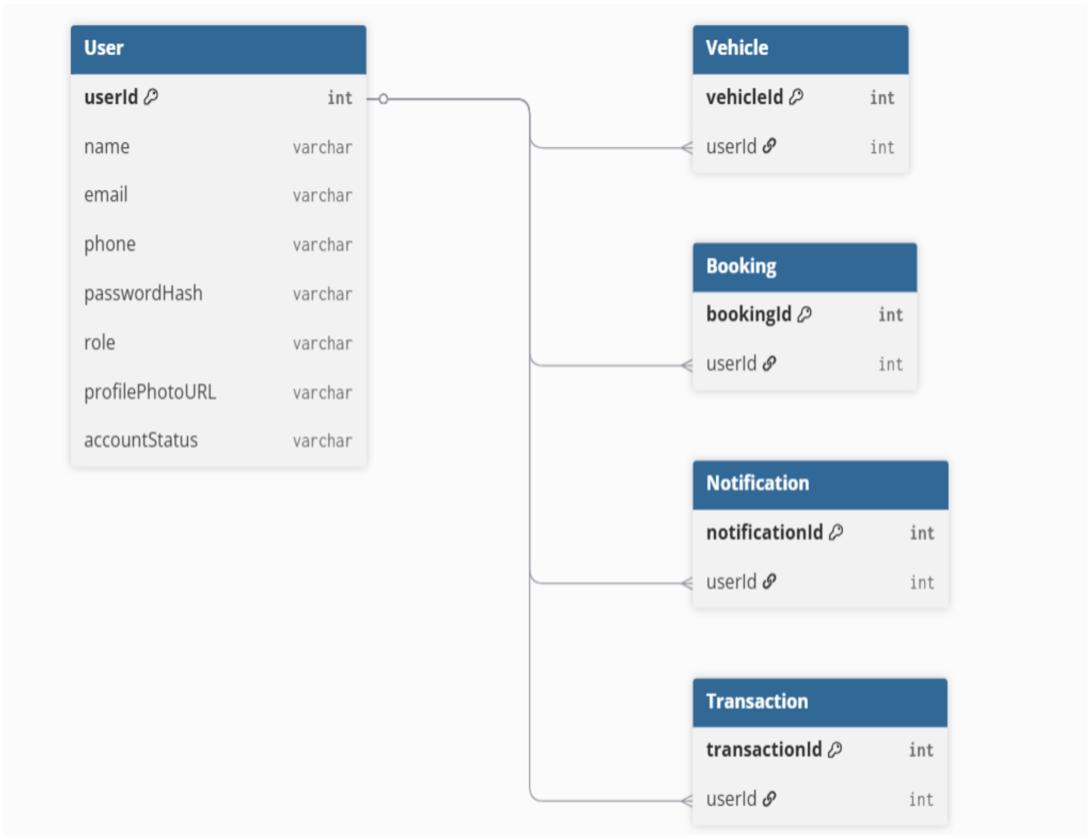
Status Codes:

- 200: OK
- 201: Created
- 400: Bad Request
- 401: Unauthorized
- 402: Payment Required
- 404: Not Found
- 409: Conflict
- 500: Server Error

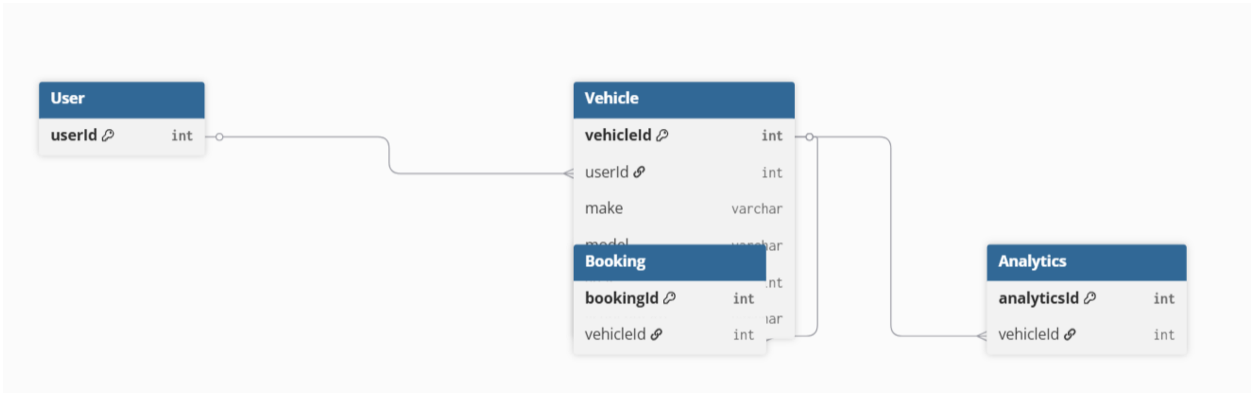
5. Database Design

5.1 ER Diagram (Entity-Relationship Diagram)

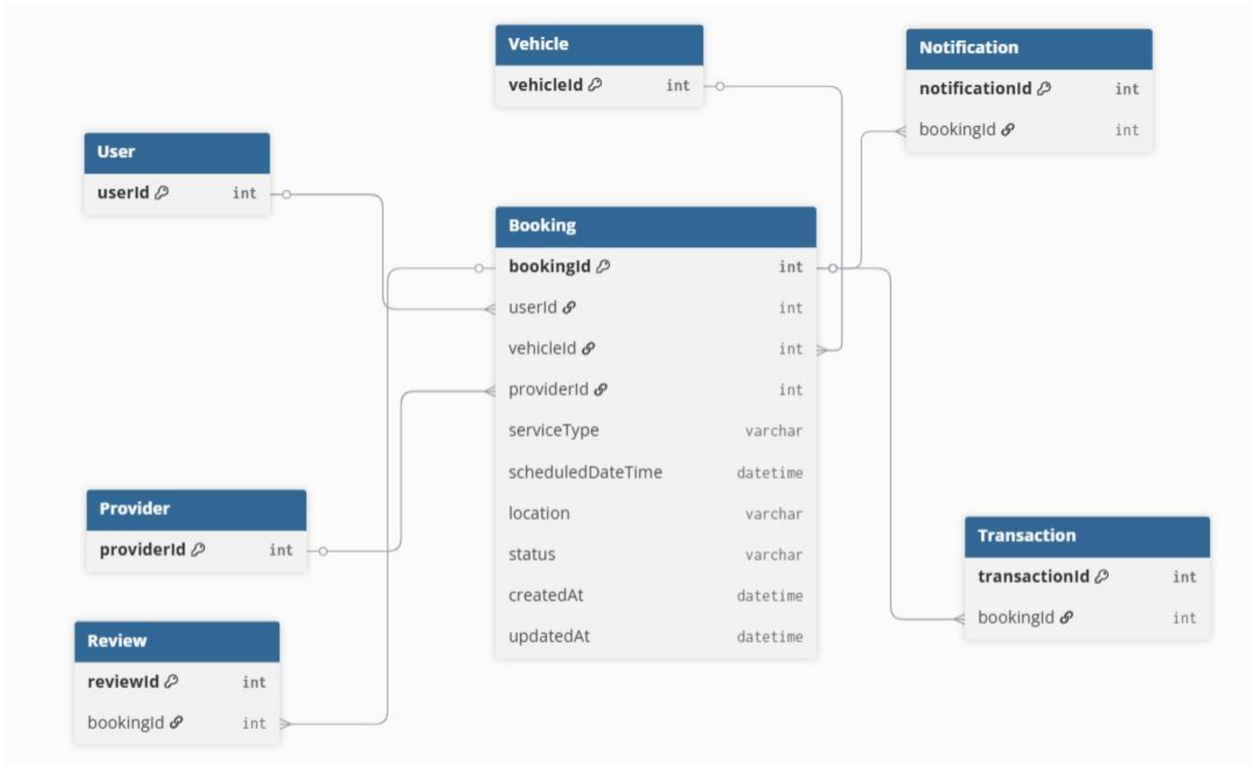
Central User entity ER Diagram.



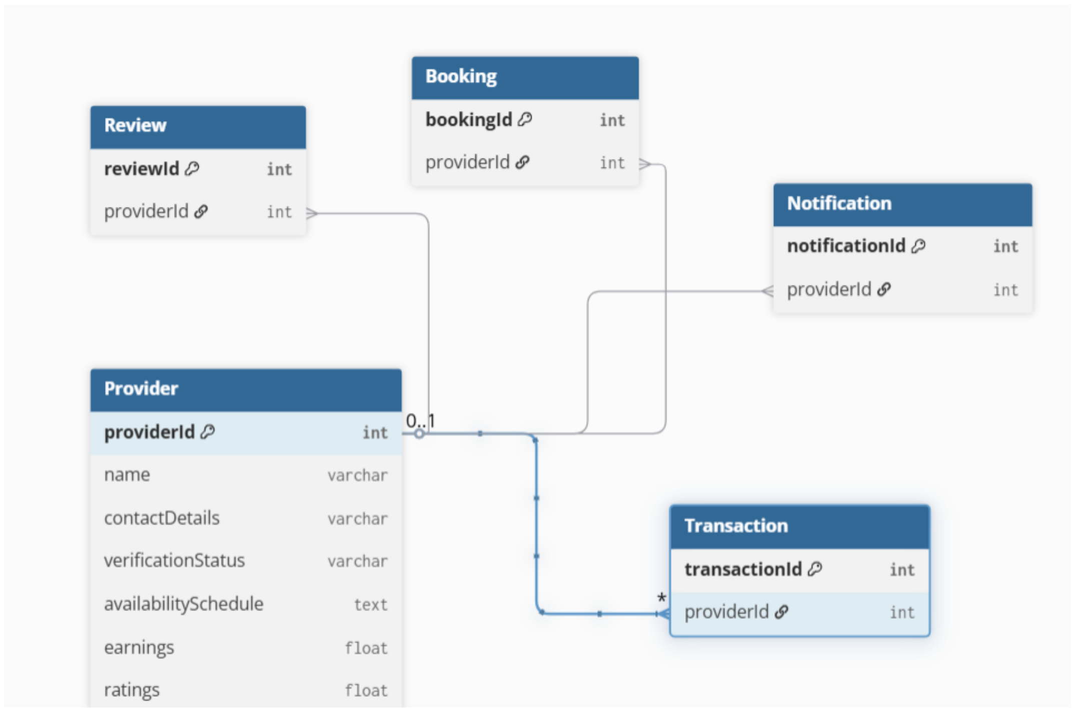
Vehicle entity ER Diagram..



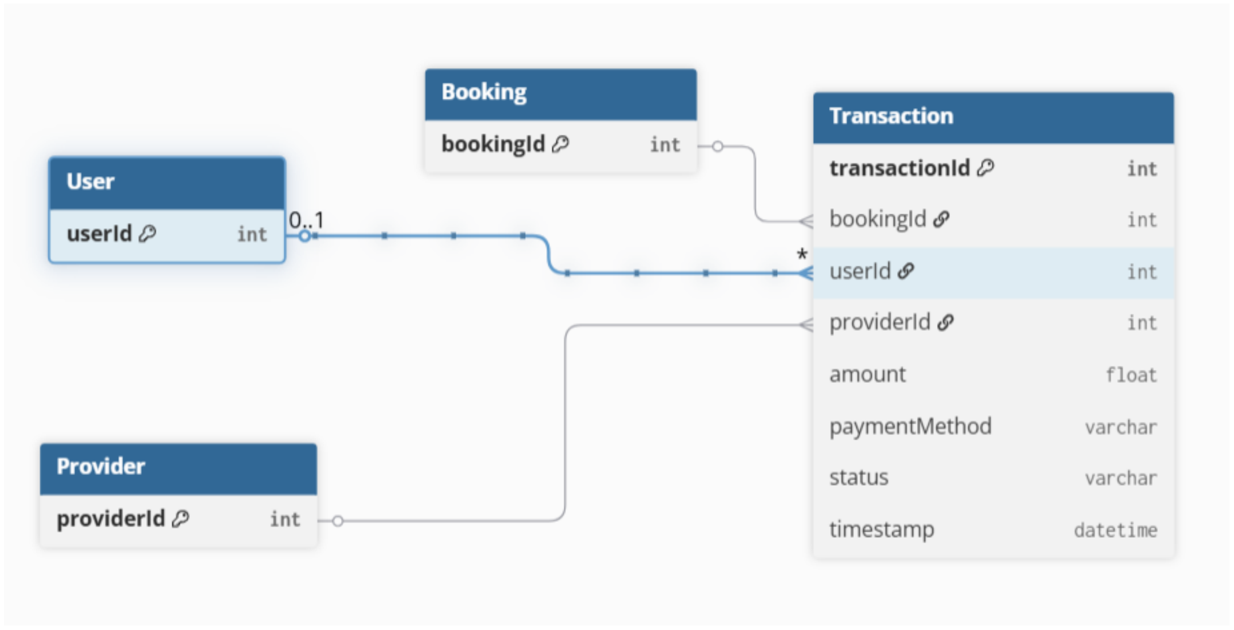
Booking entity ER Diagram.



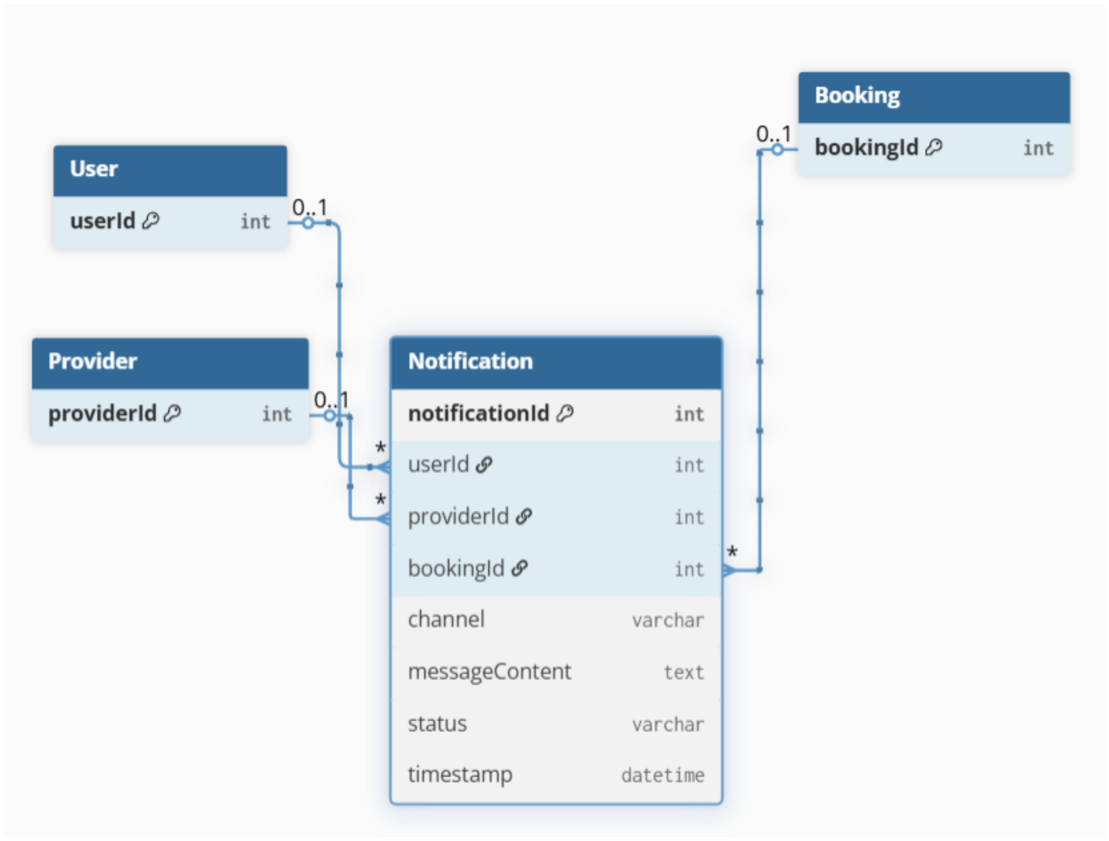
Provider entity ER Diagram.



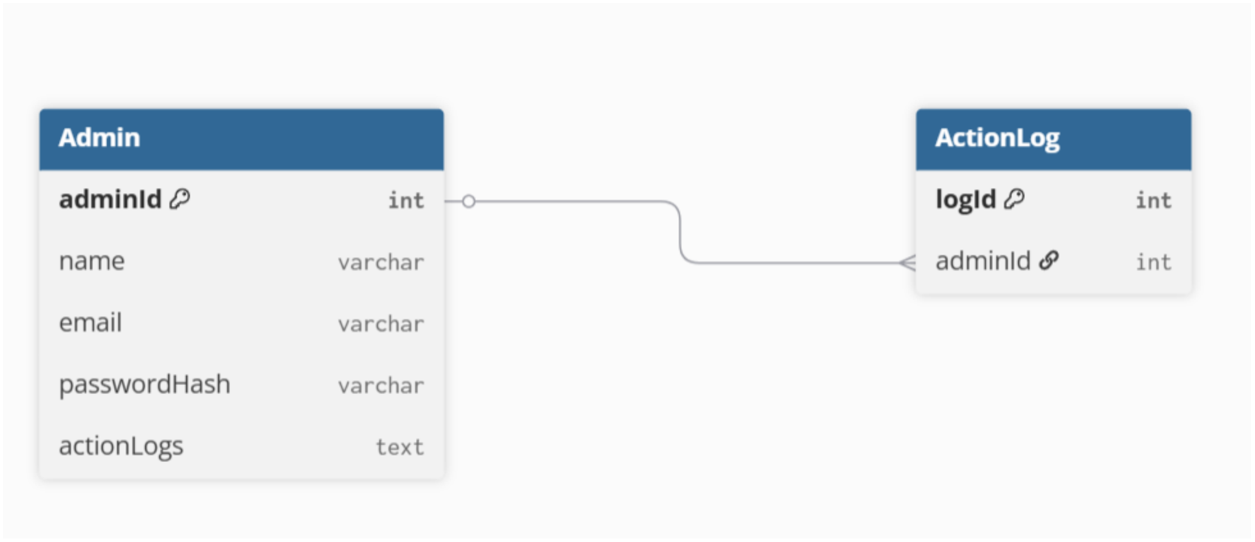
Transaction entity ER Diagram.



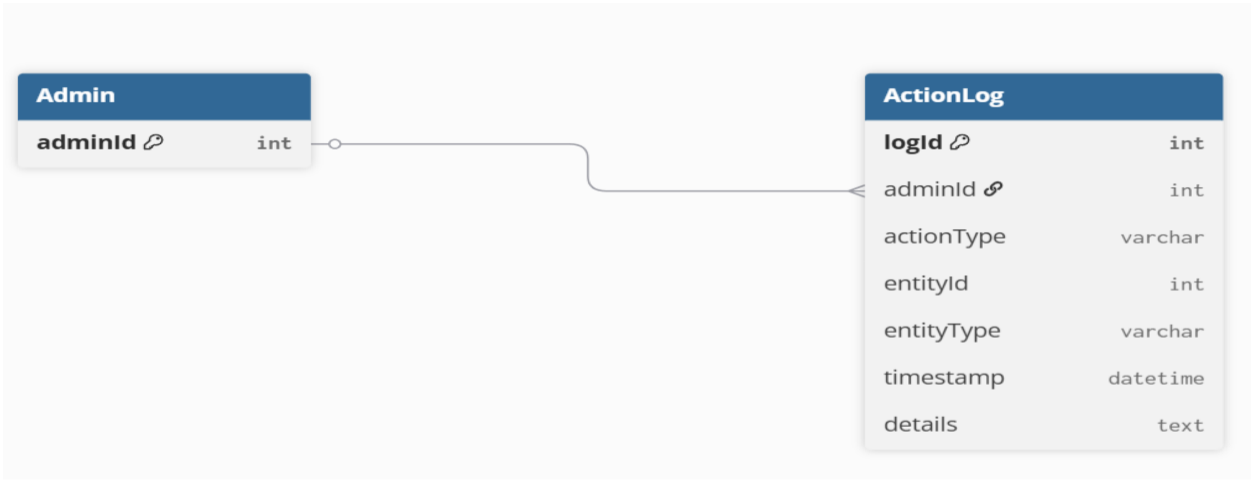
Notification entity ER Diagram.



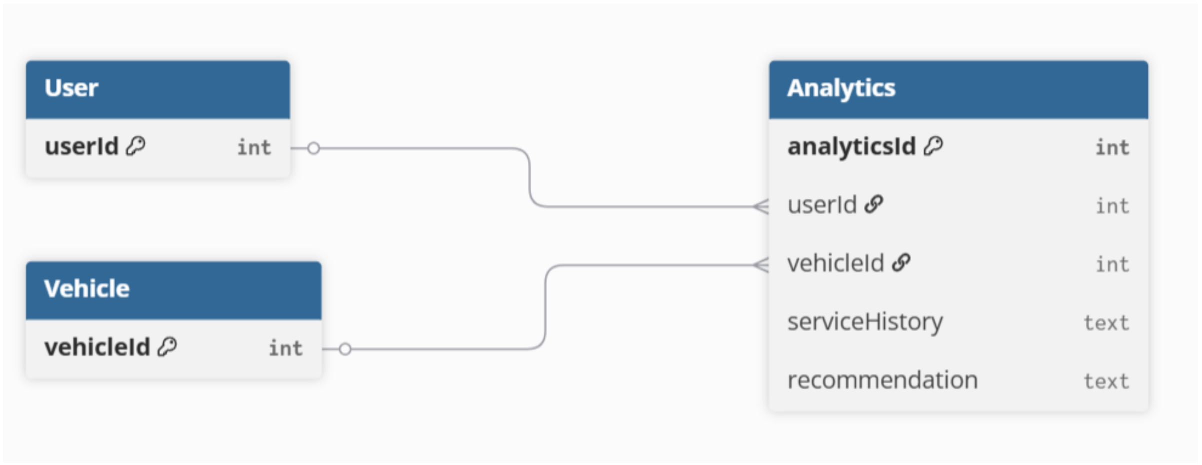
<u>Admin entity ER Diagram.</u>



ActionLog entity.

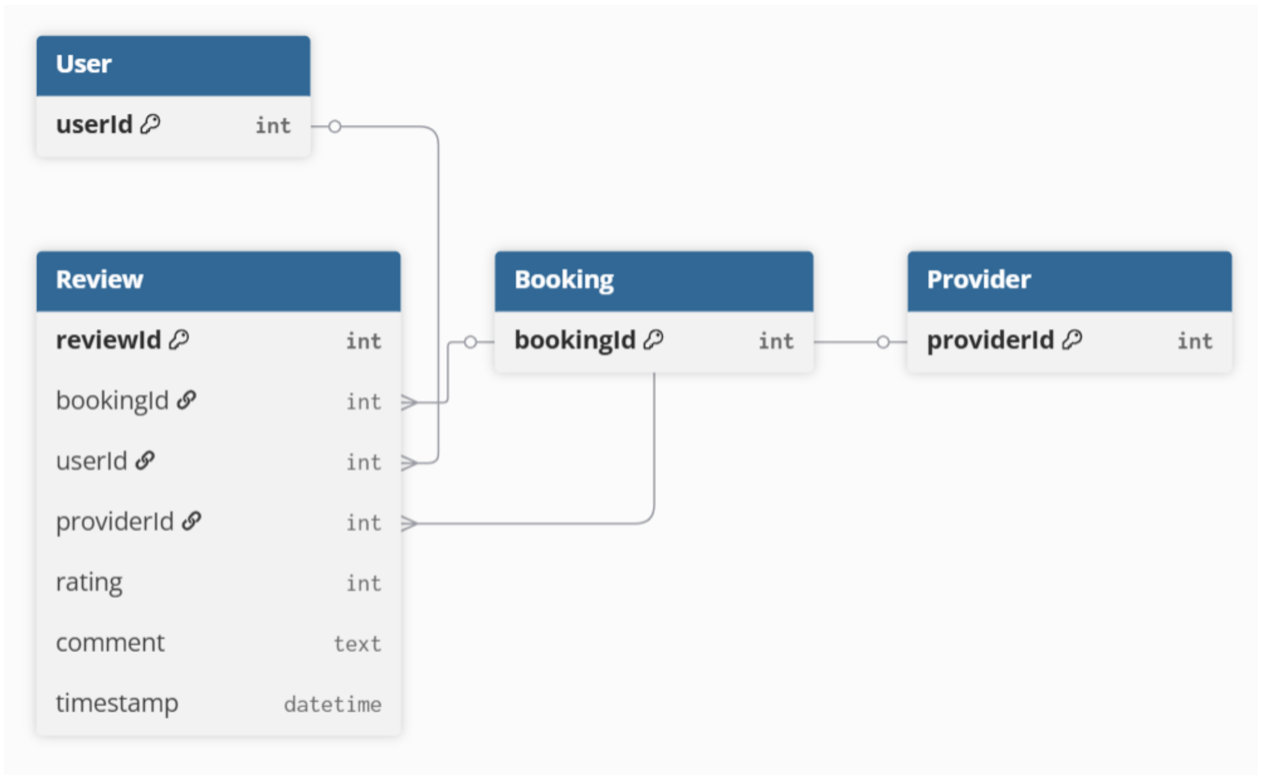


Analytics entity.



Optional:

Review entity for customer feedback ER Diagram.



5.2Data Dictionary

User Table

Field Name	Type	Nullability	Description
userId	UUID	NOT NULL	Primary key, unique user identifier
name	String	NOT NULL	User’s full name
email	String	NOT NULL	User’s email address, unique
phone	String	NOT NULL	User’s phone number, unique

Field Name	Type	Nullability	Description
passwordHash	String	NOT NULL	Hashed password
role	Enum	NOT NULL	User role: customer, provider, admin
profilePhotoURL	String	NULL	URL to profile photo
accountStatus	Enum	NOT NULL	active, suspended, pending verification

Vehicle Table

Field Name	Type	Nullability	Description
vehicleId	UUID	NOT NULL	Primary key, unique vehicle identifier
userId	UUID	NOT NULL	Foreign key to User
make	String	NOT NULL	Vehicle make
model	String	NOT NULL	Vehicle model
year	Integer	NOT NULL	Year of manufacture
licensePlate	String	NOT NULL	Vehicle license plate, unique per user

Provider Table

Field Name	**Type**	**Nullability**	**Description**
providerId	UUID	NOT NULL	Primary key, unique provider identifier
name	String	NOT NULL	Provider’s name
contactDetails	String	NOT NULL	Phone/email
verificationStatus	Enum	NOT NULL	pending, verified, rejected
availabilitySchedule	JSON	NOT NULL	Time slots available for bookings
earnings	Decimal	NOT NULL	Total earnings
ratings	Decimal	NULL	Average rating

Booking Table

Field Name	**Type**	**Nullability**	**Description**
bookingId	UUID	NOT NULL	Primary key, unique booking identifier
userId	UUID	NOT NULL	Foreign key to User
vehicleId	UUID	NOT NULL	Foreign key to Vehicle

Field Name	Type	Nullability	Description
providerId	UUID	NOT NULL	Foreign key to Provider
serviceType	Enum	NOT NULL	car_wash, detailing, tinting, maintenance
scheduledDateTime	DateTime	NOT NULL	Scheduled date and time
location	String	NOT NULL	Address or GPS coordinates
status	Enum	NOT NULL	pending, confirmed, in-progress, completed, cancelled
createdAt	DateTime	NOT NULL	Booking creation timestamp
updatedAt	DateTime	NOT NULL	Last update timestamp

Transaction Table

Field Name	Type	Nullability	Description
transactionId	UUID	NOT NULL	Primary key, unique transaction identifier
bookingId	UUID	NOT NULL	Foreign key to Booking
userId	UUID	NOT NULL	Foreign key to User

Field Name	Type	Nullability	Description
providerId	UUID	NOT NULL	Foreign key to Provider
amount	Decimal	NOT NULL	Transaction amount
paymentMethod	Enum	NOT NULL	mpesa, stripe, card
status	Enum	NOT NULL	pending, completed, failed, refunded
timestamp	DateTime	NOT NULL	Transaction timestamp

Notification Table

Field Name	Type	Nullability	Description
notificationId	UUID	NOT NULL	Primary key, unique notification identifier
userId	UUID	NOT NULL	Foreign key to User
providerId	UUID	NULL	Foreign key to Provider (optional)
bookingId	UUID	NULL	Foreign key to Booking (optional)
channel	Enum	NOT NULL	sms, email, in-app
messageContent	String	NOT NULL	Notification message
status	Enum	NOT NULL	sent, failed
timestamp	DateTime	NOT NULL	Notification timestamp

Admin Table

Field Name	Type	Nullability	Description
adminId	UUID	NOT NULL	Primary key, unique admin identifier
name	String	NOT NULL	Admin’s name
email	String	NOT NULL	Admin’s email, unique
passwordHash	String	NOT NULL	Hashed password

ActionLog Table

Field Name	Type	Nullability	Description
logId	UUID	NOT NULL	Primary key, unique log identifier
adminId	UUID	NOT NULL	Foreign key to Admin
actionType	String	NOT NULL	Type of action performed
entityId	UUID	NOT NULL	ID of the entity affected
entityType	String	NOT NULL	Type of entity affected
timestamp	DateTime	NOT NULL	When the action occurred
details	String	NULL	Additional details

Analytics Table

Field Name	**Type**	**Nullability**	**Description**
analyticsId	UUID	NOT NULL	Primary key, unique analytics identifier
userId	UUID	NOT NULL	Foreign key to User
vehicleId	UUID	NOT NULL	Foreign key to Vehicle
serviceHistory	JSON	NOT NULL	Historical service data
recommendation	String	NULL	AI-generated recommendation

Review Table (Optional)

Field Name	**Type**	**Nullability**	**Description**
reviewId	UUID	NOT NULL	Primary key, unique review identifier
bookingId	UUID	NOT NULL	Foreign key to Booking
userId	UUID	NOT NULL	Foreign key to User
providerId	UUID	NOT NULL	Foreign key to Provider
rating	Integer	NOT NULL	Rating (1-5)
comment	String	NULL	Review text
timestamp	DateTime	NOT NULL	When review was submitted

5.3Normalization Level

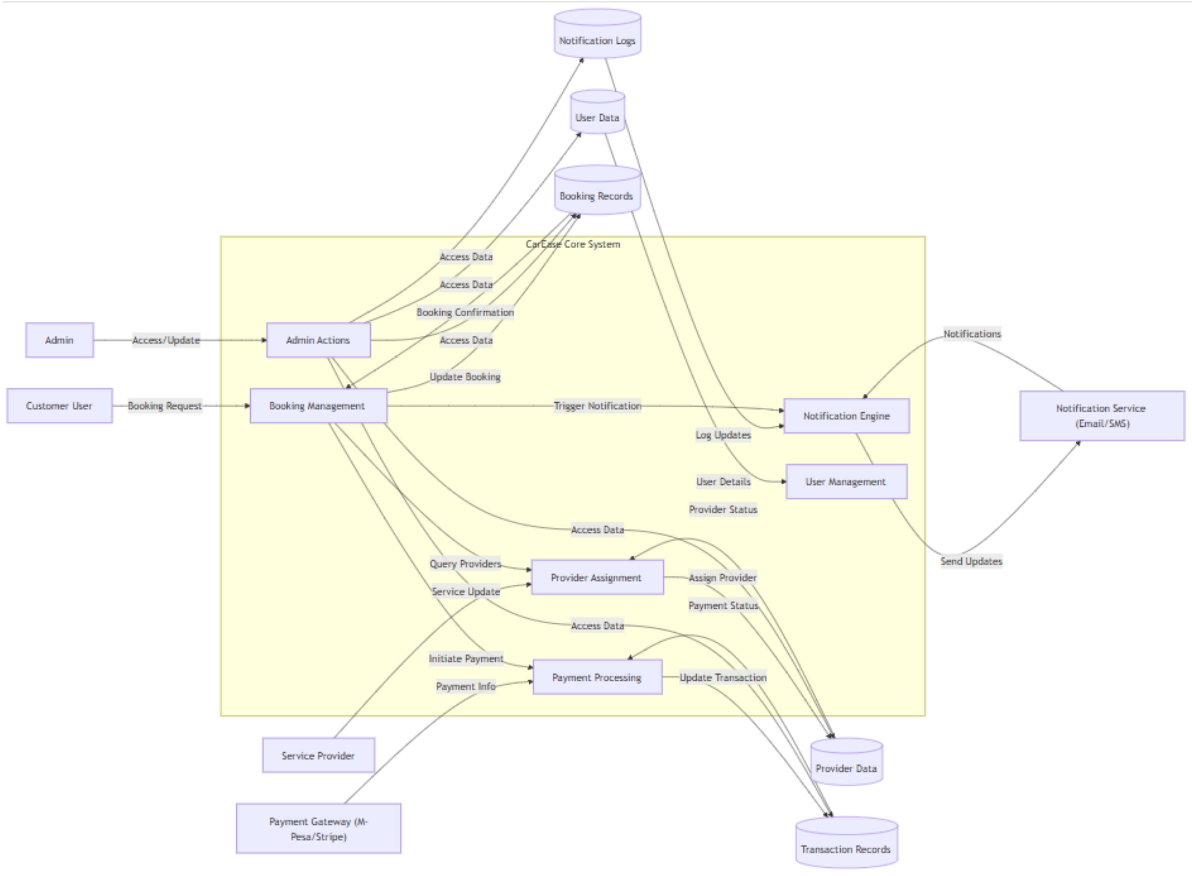
The database design **adheres to Third Normal Form (3NF)**:

- Each table contains only data related to a single entity.
- All non-key attributes are fully functionally dependent on the primary key.
- No transitive dependencies exist between non-key attributes.
- Redundancy is minimized, and relationships are enforced via foreign keys.

6 Data Flow and Control Flow

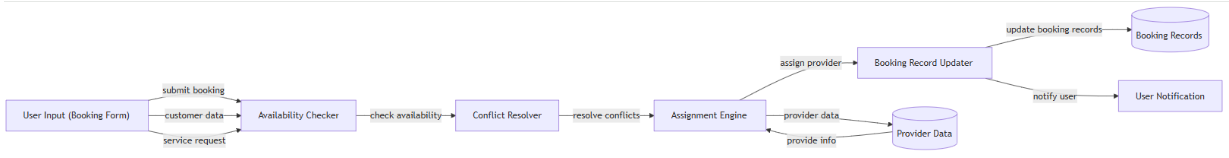
This section details the flow of data and control within the CarEase platform.

6.1 Level 1 DFD: Breaks down major processes such as service booking, payments, and tracking.



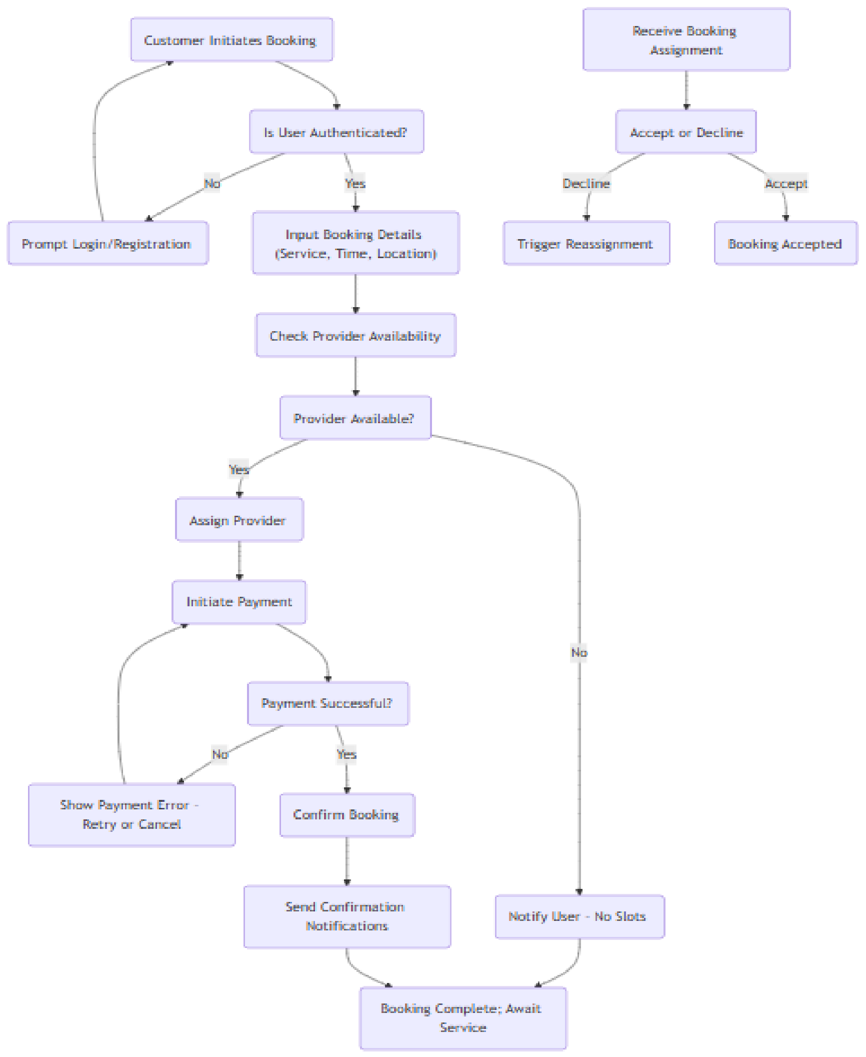
1.0 Service Booking and Fulfilment

Level 2 DFD – "Booking Management Detailed Flow"



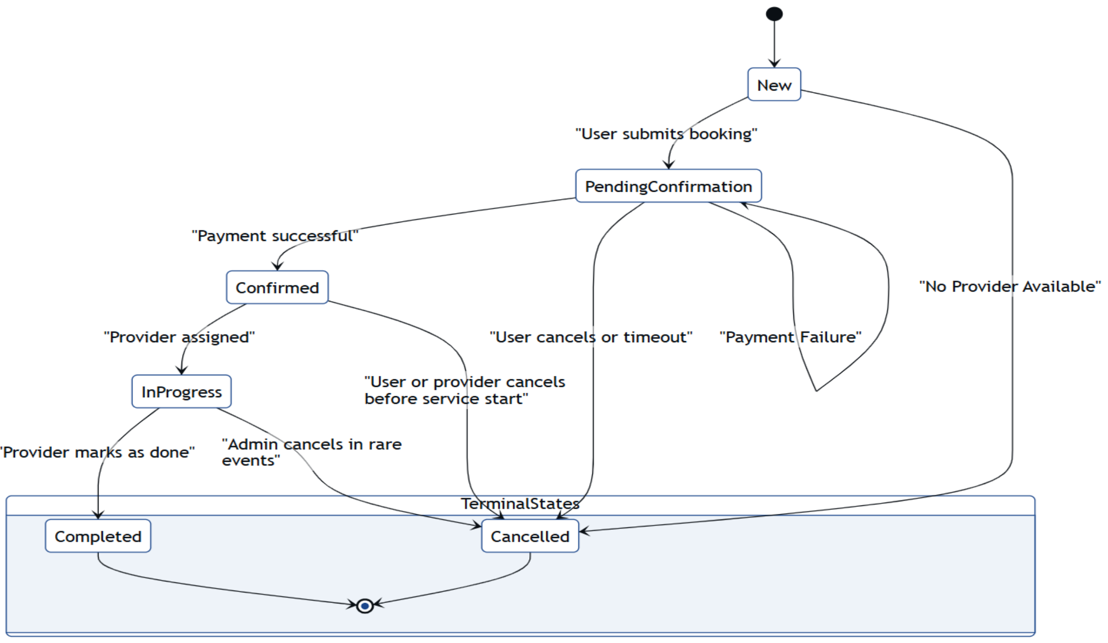
1.1 Booking management flow

6.2 Control Flow Diagram


Customer Initiates Booking
Is User Authenticated?
No
Yes
Prompt Login/Registration
Input Booking Details (Service, Time, Location)
Check Provider Availability
Provider Available?
Yes
No
Assign Provider
Initiate Payment
Payment Successful?
No
Yes
Show Payment Error - Retry or Cancel
Confirm Booking
Send Confirmation Notifications
Notify User - No Slots
Booking Complete; Await Service
Receive Booking Assignment
Accept or Decline
Decline
Accept
Trigger Reassignment
Booking Accepted

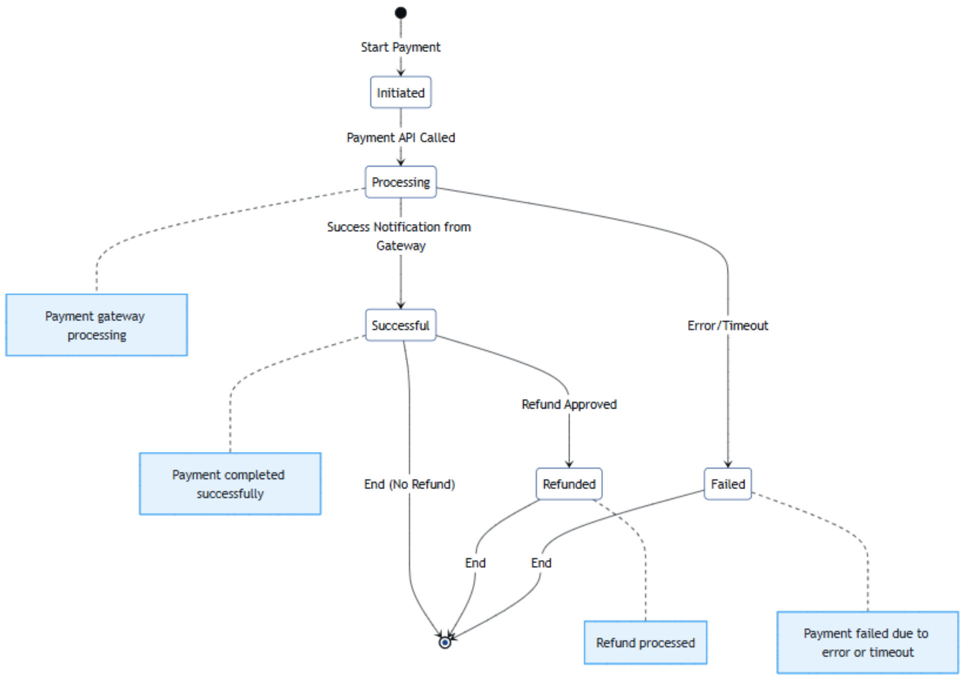
6.3 State Diagrams

State Diagram: "Booking Object"



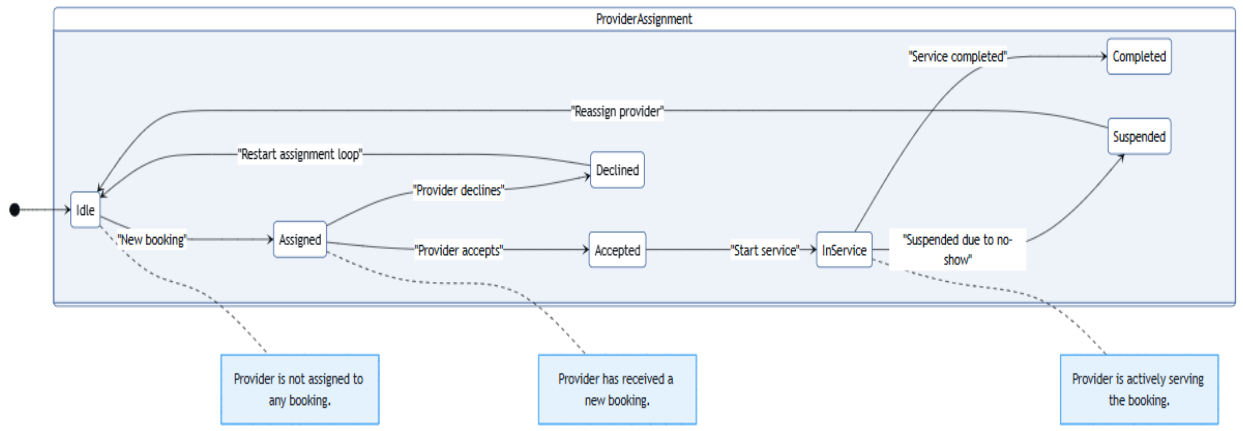
1.2 Booking State Diagram

6.4 State Diagram: "Payment Transaction"



1.3 Payment State Diagram

6.5 Optional: "Provider Assignment State"



1.4 Provider assignment state diagram

7. Non-Functional Design Considerations

This section details the non-functional design strategies for the CarEase platform, ensuring the system is robust, secure, scalable, highly available, and user-friendly. Each aspect aligns with the non-functional requirements established in the requirements analysis and industry best practices.

7.1 Performance Optimization

Caching Strategies

- **Client-Side Caching:** Frequently accessed data (e.g., user profile, service listings, booking history) is cached in the browser using localStorage/sessionStorage for web apps and IndexedDB for PWA/mobile.
- **Server-Side Caching:** Backend APIs leverage Redis or Memcached to cache hot data such as popular service providers, price lists, and recent bookings, reducing database load and improving response times.
- **API Response Caching:** HTTP cache headers are implemented for static resources and GET endpoints, allowing CDN and browser-level caching.

Lazy Loading

- **Frontend:** Images, service provider profiles, and booking history are loaded on demand as the user scrolls (infinite scroll), reducing initial page load time and bandwidth usage.
- **Backend:** Data-intensive operations (e.g., analytics, reports) are computed asynchronously and delivered incrementally to the UI.

Database Indexing

- **MongoDB Indexing:** Indexes are created on frequently queried fields such as userID, providerID, bookingID, and status to accelerate search and retrieval.
- **Compound Indexes:** Used for queries involving multiple fields (e.g., bookings by provider and status).
- **Performance Monitoring:** Automated tools (e.g., MongoDB Atlas Performance Advisor) are used to identify and optimize slow queries.

Load Testing and Profiling

- Regular load testing using JMeter or Locust to ensure the system can handle peak loads (e.g., >10,000 concurrent users).
- Continuous profiling of backend APIs and database queries to identify bottlenecks.

7.2Security Design

Authentication and Authorization

- **OAuth2 & JWT:** All API endpoints are protected using OAuth2 for delegated authorization and JWT (JSON Web Tokens) for stateless, secure user sessions.
- **Role-Based Access Control (RBAC):** Access to system resources is strictly controlled based on user roles (customer, provider, admin).

Transport Security

- **HTTPS Everywhere:** All data in transit is encrypted using TLS 1.2/1.3; HTTP requests are automatically redirected to HTTPS.
- **HSTS (HTTP Strict Transport Security):** Enforced to prevent protocol downgrade attacks.

Data Encryption

- **At Rest:** Sensitive data (user credentials, payment info, PII) is encrypted in MongoDB using field-level encryption and encrypted storage volumes.
- **In Transit:** End-to-end encryption for all communications between frontend, backend, and third-party services.

API Security

- **Input Validation and Sanitization:** All incoming data is validated and sanitized to prevent injection attacks (SQL/NoSQL injection, XSS).
- **Rate Limiting & Throttling:** API Gateway enforces strict rate limits to prevent brute-force and DDoS attacks.
- **Audit Logging:** All authentication attempts, admin actions, and sensitive operations are logged for security audits.

Compliance

- **PCI-DSS:** All payment processing (M-Pesa, Stripe) follows PCI-DSS standards.
- **Kenya Data Protection Act (KDPA):** User consent, data minimization, and right-to-erasure are enforced.
- **GDPR (where applicable):** For users outside Kenya or data stored in the EU.

7.3 Scalability Plans

Horizontal Scaling

- **Backend Services:** All Node.js microservices are stateless and containerized (Docker), allowing new instances to be added or removed based on demand using Kubernetes or AWS ECS.
- **Database:** MongoDB is deployed in a sharded cluster, supporting horizontal scaling for large datasets and high throughput.

Vertical Scaling

- **Cloud Infrastructure:** Compute resources (CPU, RAM) can be increased for EC2 instances or containers as needed for short-term spikes.

Cloud Deployment

- **AWS EC2/S3:** All components are deployed in the cloud for elastic scaling, with auto-scaling groups configured for backend services.
- **CDN (CloudFront):** Static assets are served via CDN for global reach and low latency.
- **Container Orchestration:** Kubernetes or AWS ECS manages container deployment, scaling, and health checks.

Future-Proofing

- System is designed to support multi-region deployments and additional microservices as business needs grow.

7.4 Availability & Fault Tolerance

Load Balancing

- **Elastic Load Balancer (ELB):** Distributes incoming traffic across multiple backend instances, ensuring even load and high availability.
- **API Gateway:** Acts as a single entry point, providing failover and routing logic.

Backup and Disaster Recovery

- **Automated Backups:** MongoDB and file storage (AWS S3) are backed up regularly (daily incremental, weekly full) with retention policies.
- **Point-in-Time Recovery:** Enabled for critical databases.
- **Geo-Redundant Storage:** Backups and static assets are stored in multiple AWS regions for disaster recovery.

Retry Logic and Graceful Degradation

- **API Retry:** Client and backend retry failed requests (e.g., payment, notification) with exponential backoff.

- **Circuit Breaker Pattern:** Temporarily disables calls to failing services to prevent cascading failures.
- **Graceful Degradation:** If a non-critical service (e.g., AI recommendations) fails, the core booking and payment flows remain operational.

Monitoring and Alerting

- **Centralized Logging:** All logs aggregated via ELK Stack (Elasticsearch, Logstash, Kibana) or AWS CloudWatch.
- **Real-Time Monitoring:** Prometheus/Grafana dashboards monitor system health, latency, and error rates.
- **Automated Alerts:** Critical incidents trigger alerts to DevOps via SMS, email, or Slack for rapid response.

7.5 Usability & Accessibility Design

Usability Standards

- **Consistent UI/UX:** Adheres to Material Design or similar guidelines for intuitive navigation and clear feedback.
- **Progressive Disclosure:** Complex features are revealed as needed, reducing cognitive load for new users.
- **Localization:** UI supports English and Swahili, with local time and currency formats (KES).

Accessibility

- **WCAG 2.1 AA Compliance:** All user-facing interfaces are designed to meet or exceed Web Content Accessibility Guidelines, ensuring support for screen readers, keyboard navigation, and high-contrast modes.
- **Responsive Design:** Frontend is fully responsive, optimized for desktops, tablets, and mobile devices (PWA standards).
- **Accessible Forms:** All forms use semantic HTML, ARIA labels, and clear error messages for users with disabilities.

User Feedback and Support

- **In-App Help:** Tooltips, FAQs, and guided tours assist users in navigating the platform.
- **Feedback Channels:** Users can report issues or suggest improvements directly from the app.

8 Deployment and Infrastructure Design

8.1 Target Platforms

- **Web Application:**
  - Main user interface for customers, service providers, and administrators.
  - Built with React.js (SPA), responsive and PWA-enabled for mobile usability.
- **Mobile Web (PWA):**
  - Optimized for Android and iOS browsers, installable as a Progressive Web App.
- **Cloud Infrastructure:**
  - Hosted on AWS (Amazon Web Services) for scalability, reliability, and global reach.
- **Authentication:**
  - Managed by Firebase Authentication for secure, scalable user management.
- **Payment Integrations:**
  - M-Pesa (via Daraja API) and Stripe for mobile money and card payments.

8.2 Infrastructure Diagram (Plain Language for Drawing)

Plain Language for Drawing the Deployment/Infrastructure Diagram:

- At the top, draw **users**:
  - Customers (Web/PWA)
  - Service Providers (Web/PWA)
  - Admins (Web)
- All users connect via the **Internet** to the **Load Balancer (AWS ELB)**.
- The Load Balancer routes traffic to multiple **Docker containers** running:
  - **Nginx** (as reverse proxy/static asset server)
  - **Node.js/Express.js** backend services (API Gateway, Booking, Payment, Notification, AI/Analytics, Provider Management, Admin)
- All containers are orchestrated by **Kubernetes** (or AWS ECS), enabling auto-scaling and self-healing.

- **MongoDB Cluster** (hosted on AWS, sharded/replicated for HA) handles all persistent data storage.
- **AWS S3** stores user-uploaded files (vehicle photos, documents, receipts).
- **Redis/Memcached** is used for caching frequently accessed data.
- **Third-party APIs** (M-Pesa, Stripe, Google Maps, Firebase Auth) are accessed securely by backend services.
- **CI/CD Pipeline** (GitHub Actions or Jenkins) pulls code from the repository, runs tests, builds Docker images, and deploys to Kubernetes.
- **Monitoring/Logging**:
    - **Prometheus/Grafana** for metrics and dashboards
    - **ELK Stack (Elasticsearch, Logstash, Kibana)** or AWS CloudWatch for centralized logging and alerting
- **Backup/Recovery:**
    - Automated backups of MongoDB and S3 to separate AWS regions.

8.3 Hosting

- **Docker:**
  - All backend services, including Node.js APIs and Nginx, are containerized for consistency across environments.
  - Containers are defined in a **Docker Compose** file for local development and orchestrated via Kubernetes in staging/production.
- **Nginx:**
  - Serves static frontend assets and proxies API requests to backend containers.
- **Node.js/Express.js:**
  - Runs all backend logic as stateless microservices.

8.4 CI/CD Pipeline

- **Source Control:**
  - All code is managed in GitHub (or GitLab/Bitbucket).
- **CI/CD Tools:**

- **GitHub Actions** or **Jenkins** used for:
    - Automated code linting and static analysis
    - Running unit, integration, and end-to-end tests
    - Building Docker images for frontend and backend
    - Pushing images to AWS ECR (Elastic Container Registry)
    - Deploying to Kubernetes clusters (dev, staging, production)
    - Rolling updates and automated rollback on failure

- **Secrets Management:**
    - Sensitive credentials (API keys, DB passwords) are stored in AWS Secrets Manager or Kubernetes Secrets.

8.5 Containerization (Docker Compose File Structure)

Plain Language for Docker Compose File:

- Define services:
    - **frontend**: React.js app, built and served via Nginx
    - **api-gateway**: Node.js/Express.js
    - **booking-service**: Node.js/Express.js
    - **payment-service**: Node.js/Express.js
    - **notification-service**: Node.js/Express.js
    - **ai-analytics-service**: Node.js/Express.js
    - **provider-service**: Node.js/Express.js
    - **admin-service**: Node.js/Express.js
    - **mongodb**: Official MongoDB image, with volumes for data persistence
    - **redis**: For caching
- Networks:
    - All services connected via a shared internal Docker network.
- Volumes:
    - Persistent storage for MongoDB data and logs.

8.6 Environments

Development Environment

- Local Docker Compose setup with all services, MongoDB, and Redis.
- Hot-reloading enabled for frontend and backend.
- Mock integrations for third-party APIs.
- Developer-specific configuration (e.g., local S3, test payment gateways).

Staging Environment

- Mirrors production as closely as possible.
- Deployed on AWS with Kubernetes/ECS.
- Uses real cloud services (MongoDB Atlas, AWS S3, Redis Cloud).
- Connected to sandbox/test versions of payment and map APIs.
- Used for QA, UAT, and pre-release validation.

Production Environment

- Fully managed on AWS.
- Auto-scaling Kubernetes cluster for backend services.
- Production MongoDB cluster (sharded, replicated).
- AWS S3 for file storage, with strict access controls.
- Real M-Pesa, Stripe, and Google Maps integrations.
- 24/7 monitoring, alerting, and automated backups.
- Blue/green or canary deployment strategies for zero-downtime releases.

9 <u>Testing Design</u>

9.1 Testing Strategy

The CarEase platform will employ a comprehensive, multi-layered testing strategy to ensure the reliability, security, and quality of the system. The following testing levels will be rigorously applied:

Unit Testing

- **Scope:**
    - Individual functions, classes, and methods within backend services (Node.js/Express.js) and frontend components (React.js).
    - Examples: Booking creation logic, payment calculation, user authentication, form validation.
- **Tools:**
    - Backend: Jest, Mocha, Chai
    - Frontend: Jest, React Testing Library

Integration Testing

- ➢ **Scope:**
    - Interactions between modules/services (e.g., booking and payment, user registration and notification).
    - API endpoint testing, database integration, third-party API integrations (M-Pesa, Stripe, Google Maps, Firebase Auth).
- ➢ **Tools:**
    - Supertest (Node.js API integration), Postman/Newman (API workflows), Cypress (end-to-end flows)

System Testing

- ➢ **Scope:**
    - End-to-end workflows covering the entire system, including booking, payment, notifications, and provider assignment.
    - Full-stack validation with real or mock external services.
- ➢ **Tools:**
    - Cypress (E2E browser automation), Selenium (cross-browser testing), Puppeteer

User Acceptance Testing (UAT)

- ➢ **Scope:**
    - Real users (customers, providers, admins) execute test scenarios in the staging environment.
    - Focus on business requirements, usability, and real-world workflows.

- **Process:**
    - UAT scripts are derived from user stories and acceptance criteria.
    - Feedback is collected and used to refine the system before production deployment.

Non-Functional Testing

- **Performance Testing:**
  - Load, stress, and scalability testing using JMeter or Locust.
- **Security Testing:**
  - Vulnerability scanning (OWASP ZAP), manual penetration testing, and static code analysis.
- **Accessibility Testing:**
  - Axe, Lighthouse, and manual screen reader/keyboard navigation checks for WCAG 2.1 AA compliance.

9.2 Test Data Requirements

- **User Data:**
    - Multiple user roles (customers, providers, admins) with valid and invalid credentials.
    - Edge cases: duplicate emails/phones, missing fields, invalid formats.
- **Booking Data:**
    - Bookings in various states (pending, confirmed, in-progress, completed, cancelled).
    - Overlapping and conflicting bookings for providers.
- **Payment Data:**
    - Valid and invalid payment scenarios for M-Pesa and Stripe.
    - Refunds, failed transactions, duplicate payments.
- **Provider Data:**
    - Verified and unverified providers, varying availability schedules, incomplete profiles.

- **Notifications:**
    - Triggers for all supported channels (SMS, email, in-app).
- **Analytics:**
    - Simulated service histories for AI recommendations and reminders.

Test Data Management:

- Synthetic data is generated for automated tests.
- Anonymized production-like data is used in staging for realistic UAT.

9.3 Automated Tests

Automated Testing Tools

- **Backend:**
    - Jest, Mocha, Chai for unit/integration tests.
    - Supertest for API endpoint validation.
- **Frontend:**
    - Jest, React Testing Library for unit/component tests.
    - Cypress for end-to-end UI workflows.
- **API:**
    - Postman/Newman for automated API contract and regression testing.
- **Performance:**
    - JMeter or Locust for automated load/stress testing.
- **Accessibility:**
    - Axe-core, Lighthouse for automated accessibility checks.

Coverage Goals

- **Unit Test Coverage:**
    - Minimum 80% code coverage for critical backend and frontend modules.
- **Integration Test Coverage:**
    - All major API flows and service integrations covered.

- **E2E Test Coverage:**
    - All user journeys (booking, payment, provider assignment, notifications, admin actions) automated in Cypress.
- **Regression Suite:**
    - Automated regression suite runs on every code change.

9.4 CI/CD Testing Integration

- **Continuous Integration (CI):**
    - All code commits trigger automated build and test pipelines (GitHub Actions or Jenkins).
    - Linting, unit, and integration tests must pass before merging to main branches.
- **Continuous Deployment (CD):**
    - Automated deployment to development, staging, and production environments.
    - E2E and regression tests run against staging before production releases.
    - Canary/blue-green deployment strategies allow for staged rollouts and rollback on failure.
- **Test Reporting:**
    - Test results, code coverage, and performance metrics are published to dashboards and sent to developers via Slack/email.
- **Quality Gates:**
    - Builds are blocked if test coverage or quality thresholds are not met.

10 Risk and Mitigation Plans

Below are the primary design and operational risks identified for the CarEase platform, along with detailed mitigation strategies for each:

Risk 1: Third-Party API Limits and Downtime

Description: The platform relies on external APIs (M-Pesa, Stripe, Google Maps, Firebase Auth). These services may impose rate limits, experience downtime, or change their interfaces.

- **Mitigation:**

- Implement retry queues for failed API calls with exponential backoff.
- Cache non-sensitive, non-volatile data from APIs (e.g., static map data, provider locations).
- Design the system to gracefully degrade: if a third-party API fails, inform the user and allow them to retry or use fallback options.
- Monitor API usage and set up alerts for approaching rate limits.
- Maintain up-to-date documentation and test suites for all integrated APIs.

Risk 2: Data Security Breaches

Description: Sensitive user data (personal info, payment details) could be exposed due to vulnerabilities.

- **Mitigation:**

- Enforce **end-to-end encryption** (TLS/SSL in transit, field-level encryption at rest).
- Use **OAuth2/JWT** for secure authentication and role-based access control.
- Regularly conduct **security audits** and penetration testing.
- Store secrets in secure vaults (AWS Secrets Manager, Kubernetes Secrets).
- Ensure compliance with **PCI-DSS** and **Kenya Data Protection Act**.

Risk 3: Scalability Bottlenecks

Description: Rapid growth in users or service requests could overwhelm backend services or the database.

- **Mitigation:**

- Use **horizontal scaling** (Kubernetes, AWS ECS) for stateless services.
- Employ **MongoDB sharding** and **Redis caching** for high-throughput data access.
- Conduct regular **load testing** and monitor system metrics to anticipate scaling needs.
- Design microservices to be stateless and independently deployable.

Risk 4: Data Loss or Corruption

Description: System or infrastructure failures could result in loss of critical data (bookings, payments).

- **Mitigation:**

  - Implement **automated, geo-redundant backups** (daily incremental, weekly full) for MongoDB and S3.
  - Enable **point-in-time recovery** for databases.
  - Use **transactional integrity** and idempotency in payment and booking flows.
  - Regularly test backup restoration procedures.

Risk 5: Service Provider No-Shows or Delays

Description: Providers may miss appointments, causing poor user experience.

- **Mitigation:**

  - Automated **reminders** and real-time notifications for providers.
  - Penalize repeated no-shows through the rating system and possible suspension.
  - Allow users to rebook quickly with alternate providers.
  - Track provider reliability metrics for admin review.

Risk 6: Regulatory Compliance Changes

Description: Changes in Kenyan or international data/privacy/payment regulations could impact operations.

- **Mitigation:**

  - Regularly review compliance requirements (KDPA, PCI-DSS, GDPR).
  - Modularize compliance-related code for easier updates.
  - Engage legal counsel for ongoing monitoring.

Risk 7: Poor User Adoption/Usability

Description: If the platform is not intuitive or accessible, users may abandon it.

- **Mitigation:**

  - Conduct **usability testing** and collect user feedback during UAT and after launch.
  - Adhere to **WCAG 2.1 AA** accessibility standards and responsive design.
  - Provide in-app help, FAQs, and customer support channels.

Risk 8: Vendor Lock-In

Description: Over-reliance on a single cloud provider or third-party service could limit flexibility.

- **Mitigation:**

➢ Use **containerization** (Docker/Kubernetes) for portability.
➢ Abstract third-party integrations behind service interfaces.
➢ Regularly review alternative providers for critical services.

Risk 9: Fraudulent Transactions

Description: Users or providers may attempt to game the payment or booking system.

- **Mitigation:**

➢ Implement **fraud detection** rules (e.g., flagging rapid repeat bookings or unusual payment patterns).
➢ Require **multi-factor authentication** for sensitive actions.
➢ Manual review of flagged transactions by admins.

Risk 10: Performance Degradation Under Load

Description: High traffic (e.g., during promotions) could slow down or crash the system.

- **Mitigation:**

➢ Use **auto-scaling**, load balancing, and CDN for static assets.
➢ Monitor performance in real time and alert on anomalies.
➢ Optimize code and database queries for high concurrency.

11 Appendices

A. Diagrams

Included Visuals and Models

- **System Architecture Diagrams**
  High-level deployment and component diagrams illustrating frontend-backend separation, infrastructure layout (including API gateways, databases, third-party integrations), and hosting on AWS using Docker, Nginx, and Node.js.
- **ER Diagram (Entity-Relationship Diagram)**
  Visualizes relationships between core entities such as User, Vehicle, Provider, Booking, Transaction, Notification, Admin, ActionLog, Analytics, and (optional) Review.
- **UML Class and Sequence Diagrams**
  Provided for major modules:
  - User Management, with registration and authentication flows
  - Booking/Provider assignment
  - Payment and transaction status
  - Notification creation and delivery
- **Data Flow Diagrams (DFD)**
  - Level 1: Service booking and fulfillment—from user booking request to provider assignment, payment, and notification.
  - Level 2: Booking management workflow, showing internal validations and status transitions.
- **Control and State Diagrams**
  - State transitions for Booking and Payment Transaction objects, including states like Pending, Confirmed, In Progress, Completed, Cancelled, Failed, Refunded.
  - Provider assignment lifecycle (Idle, Assigned, Accepted, In Service, Completed, Suspended).
- **User Navigation Flowcharts**
  - For customer, provider, and admin applications.
- **Architecture Diagrams:**
  - High-level component/deployment diagrams (see Section 3).

- **UML Class and Sequence Diagrams:**
    - For each major module (see Section 4).
- **ER Diagrams:**
    - Database structure (see Section 5).
- **Data Flow Diagrams:**
    - Booking, payment, and notification flows (see Section 6).
- **Infrastructure Diagram:**
    - Cloud hosting, container orchestration, and CI/CD pipeline (see Section 8).
- **Navigation Flowcharts:**
    - For customer, provider, and admin UIs (see Section 4.2).

B. Design Decision Logs

Major Technical and Product Decisions

Feature/Component	**Decision & Rationale**
Front-End Framework	React.js chosen for modularity, PWA support, and rapid development.
Backend Technology	Node.js and Express.js selected for scalable API development and asynchronous event handling.
Database	MongoDB for flexible data modeling, scalability, and easy integration with Node.js.
Authentication	Firebase Authentication for secure and scalable user/session management and multi-factor auth.
Payment Integrations	M-Pesa (Daraja API) for Kenya's dominant mobile money market; Stripe for card payments.
Cloud Infrastructure	AWS (EC2, S3, ECR) for auto-scaling, managed storage, and global deployment.

Feature/Component	**Decision & Rationale**
Containerization	Docker and Kubernetes/ECS for portability, auto-scaling, and simplified rollout/rollback.
Accessibility	Conformance with WCAG 2.1 AA to ensure inclusive access for all users.
Testing & CI/CD	GitHub Actions/Jenkins for automated build, test, and deployment; Cypress/Jest for coverage.
System Modularity	Microservice-based backend design for independent scaling and resilience.

C. Tools and Libraries

Frontend Stack

- React.js (SPA & PWA)
- Redux (state management)
- Axios (HTTP client)
- Material-UI or Bootstrap (UI components)
- Figma/Balsamiq (mockups and wireframes)
- Cypress, Jest, React Testing Library (testing)

Backend Stack

- Node.js, Express.js (API and service logic)
- Mongoose (ODM for MongoDB)
- Passport.js, JWT (authentication)
- Redis (caching)
- Supertest, Mocha, Chai, Jest (testing)

Database & Storage

- MongoDB (core data store)
- MongoDB Atlas (cloud-managed DB)

- AWS S3 (file/document storage)

DevOps & Infrastructure

- Docker, Docker Compose
- Kubernetes (or AWS ECS/EKS)
- Nginx (reverse proxy and static file serving)
- AWS EC2, ECR, CloudWatch, Secrets Manager

CI/CD and Version Control

- Git (code versioning)
- GitHub/GitLab (repository and project management)
- GitHub Actions/Jenkins (CI/CD pipelines for build, test, deploy)

Monitoring & Logging

- Prometheus, Grafana (metrics, dashboards)
- ELK Stack (Elasticsearch, Logstash, Kibana; centralized logs)
- Sentry (runtime error/reporting)

Integrations

- M-Pesa Daraja API (mobile payments)
- Stripe (international card payment handling)
- Google Maps API (real-time geolocation, route optimisation)
- Firebase Auth (user login and session management)
- Postman, Swagger (API documentation/testing)

Testing & Analytics

- JMeter, Locust (performance/load)
- OWASP ZAP, Axe, Lighthouse (security/accessibility)
- Manual and automated accessibility and performance audits

Design & Modeling

- Figma, Balsamiq (UI/UX)
- Lucidchart, draw.io, dbdiagram.io (all diagrams: ER, UML, flowcharts)

Notes

- All diagrams, logs, and technical references are maintained in the shared project repository according to the submission guidelines and are available for review as per requirements.
- Tools and frameworks were selected for compatibility with the target environment, scalability, and team skill profiles.
- Updates to the appendices will be tracked via project log commits and accessible through the version history.