Bilkent University
Department of Computer Engineering

Senior Design Project

T2534

DriveMe

Detailed Design Report

22102334, Eda Alparslan, eda.alparslan@ug.bilkent.edu.tr
22102720, Duru Solakoğlu, duru.solakoglu@ug.bilkent.edu.tr
21901440, Berfin Çetinkaya, berfin.cetinkaya@ug.bilkent.edu.tr
22001734, Ufuk Baran Güler, baran.guler@ug.bilkent.edu.tr
21601625, Ege Kaan Eren, kaan.eren@ug.bilkent.edu.tr

Prof. İbrahim Körpeoğlu
Mert Bıçakçı & İlker Burak Kurt

13.03.2026

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

1.1 Purpose of the system

DriveMe is an innovative mobile application designed to connect vehicle owners with verified, professional drivers. The system addresses specific mobility challenges faced by individuals who own a car but are temporarily unable or unwilling to drive due to health issues, alcohol consumption, legal limitations like license suspension, and so on. By allowing a professional to drive the user's personal vehicle, the system ensures convenience while maintaining strict compliance with Turkish transportation laws.

The primary objectives of DriveMe include:

- **Enhancing Road Safety**: By providing on-demand professional drivers, the platform aims to reduce risks associated with impaired driving and accidents resulting from it.
- **Providing Transparent Pricing**: A dynamic pricing recommends fair price ranges based on distance, duration, and demand, allowing for a balanced marketplace.
- **Guaranteeing User Trust through Verification**: The system implements a rigorous registration process for both driver licenses and vehicle documentation.
- **Ensuring Real-Time Security**: Utilizing GPS tracking and code-based authentication, the platform monitors trip progress to ensure passenger and vehicle safety.
- **Optimizing User Experience**: With an intuitive UI/UX, the application provides seamless interaction for users of all technical proficiencies and from all different backgrounds.

1.2 Design goals

Performance: The mobile application should respond to user actions (login, creating a request, searching for offers) within seconds under normal conditions.

Availability: The system should achieve 99 % uptime and gracefully handle partial outages, ensuring that pending requests are cached locally so that drivers can still view them during brief connectivity issues.

Scalability: The system should be horizontally scalable to accommodate increases in the number of users, vehicles and drivers. It should allow database and server resources to be expanded without redesigning the application.

Security and Privacy: All personal data shall be stored securely using encryption.

The system must comply with the Turkish Personal Data Protection Law (KVKK) and GDPR, collecting only necessary data, storing location data for no longer than operationally required and protecting it from unauthorized access.

Reliability: The application should recover gracefully from network interruptions. Local caching of pending requests and robust retry mechanisms shall prevent data loss and ensure continuity.

Maintainability: The code base will be modular and well documented to simplify updates and bug fixing. The design should support clear separation of concerns among user management, matching, pricing and tracking modules.

Usability: The user interface shall be intuitive and accessible, with clear navigation and feedback. The app should be usable by elderly and disabled users, for example through appropriate font sizes and contrast.

Compatibility: The mobile application shall support Android version 8.0 or higher on devices with at least 4 GB of RAM.

Dependence on Third-Party Services: The system will use reliable third-party services such as Google Maps API for geolocation and routing. It should handle API limitations, outages or pricing changes by providing fallback options or notifications.

Compliance with Legal Constraints: The system should restrict functionality to driver services where a licensed driver operates the user's car. It shall not support shared rides, pooled trips or taxi-like operations, ensuring compliance with Turkish transportation regulations.

Data Integrity: All changes to trip states (requested, matched, started, completed) shall be transactional so that incomplete operations do not leave the system in an inconsistent state.

Aesthetic: The user interface will follow a professional and modern visual language. The design will prioritize a clean and clutter-free layout to enhance user confidence and ensure that critical information is easily distinguishable.

1.3 Definitions, acronyms, and abbreviations

KVKK: Turkish Personal Data Privacy Law.

GDPR: General Data Protection Regulation by the European Union.

GPS: Global Positioning System. A satellite-based navigation system.

API (Application Programming Interface): A set of rules and protocols for building and interacting with software applications.

RAM (Random Access Memory): A form of computer data storage.

UI (User Interface): The space where interactions between humans and the application occur.

UX (User Experience): The overall experience of a person using the application, especially in terms of how easy or pleasing it is to use.

SAAS: Software as a service.

IEEE: Institute of Electrical and Electronics Engineers. A professional association that also creates frameworks and documentation guidelines.

UML: Unified Modelling Language.

ACM (Association for Computing Machinery): An educational and scientific computing society.

SSL (Secure Sockets Layer): Standard technology for securing an internet connection by encrypting data sent between two server.

TLS (Transport Layer Security): More secure version of SSL.

JWT: JSON Web Token.

1.4 Overview

DriveMe consists of a front end mobile application (Android) and a back end service that handles user management, matching, pricing and trip tracking. Users can sign up as either passengers or drivers. Passengers register their personal details and vehicle

documentation; drivers register their licence and identity documents. An administrator verifies both vehicle and driver records. Once verified, passengers can post ride requests by providing their vehicle, current location and destination. The system's matching engine recommends a price range based on distance, duration and current demand. Licensed drivers view nearby pending requests, submit offers within the suggested range and receive notifications when a passenger accepts their offer.

Prior to departure, passenger and driver authenticate each other via a 4-digit code generated by the system. During the trip, the application activates GPS tracking to monitor route adherence. Upon arriving at the destination, the passenger ends the ride, and both parties confirm that the trip was completed and the driver received cash payment. Passengers can then rate and review drivers. Administrators oversee user registration, verify documentation and handle disputes. The system relies on third party services such as Google Maps API for geolocation and routing, and must comply with Turkey's Personal Data Protection Law (KVKK) and the General Data Protection Regulation (GDPR).

2. Current Software Architecture

Many established platforms dominate the mobility market for transportation and on-demand services. We analyzed these companies and identified a significant gap for users who want to travel in their own vehicles but need a professional driver. DriveMe is designed to fill this gap by combining features that differentiate it from the following existing solutions:

On Demand Personal Driver Services (e.g. Dryver): These are competitors using a Demand Response Service Model. However, most of these platforms rely more on scheduled bookings and lack dynamic geolocation based Real Time Matching [1]. Furthermore, these applications have shortcomings in vehicle and driver verification modules.

Commercial Ride Platforms (e.g., Uber, BiTaksi): These systems use a Centralized Brokerage Architecture optimized for "Commercial Vehicle-Driver". Data models are not built to accommodate third-party, user-owned assets, and users cannot use their

own vehicles on these platforms. This can make a gap for those who prioritize the safety and comfort of their own vehicles.

Traditional Driver and Valet Services (e.g., Istanbul Taxi): Existing local solutions in Türkiye generally rely on manual coordination and lack a fully digital software architecture. These "analog" solutions suffer from a lack of Persistent Data Management and Real Time Monitoring, leading to inconsistent pricing and safety risks for both vehicle owners and drivers [2].

DriveMe's Contribution:
DriveMe differentiates itself by proposing a Hybrid Service Architecture. Unlike its competitors, DriveMe integrates the user's vehicle-specific data directly into the matching logic and safety protocols. This "Triangle Data Model" provides a secure, on-demand, and legally compliant framework that existing commercial architectures cannot support.

3. Proposed Software Architecture

3.1 Overview

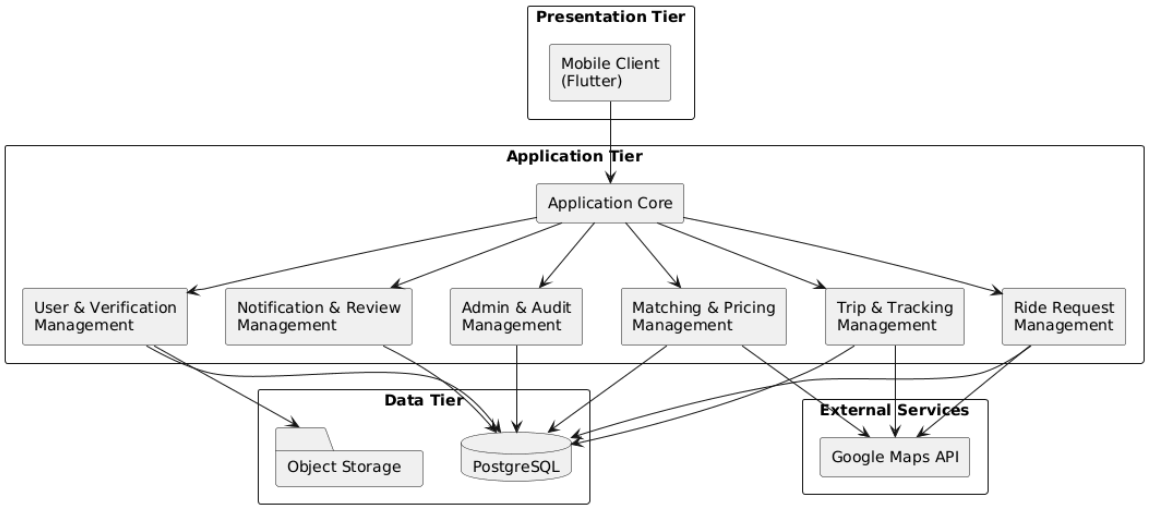
The proposed software architecture follows a layered structure consisting of a Flutter-based mobile client, a Spring Boot backend application, a PostgreSQL persistence layer, and selected external services for mapping and route estimation. This architecture was chosen to provide modularity, maintainability, and transactional consistency while remaining feasible within the project scope.

At the presentation level, the system is accessed through a single mobile application that supports both passenger and driver workflows. The client provides interfaces for registration, verification submission, ride request creation, offer management, trip execution, and post-trip review submission. Since the platform depends on real-time and location-aware interactions, the mobile client plays a central role in delivering a responsive user experience.

At the application level, DriveMe is implemented as a modular monolithic backend using Spring Boot. Although deployed as a single application, the backend is organized into separate subsystems responsible for user and authentication management, verification, ride request handling, matching and offer management, pricing, trip lifecycle management, notifications, reviews, and administrative auditing. This structure allows the system to remain manageable while maintaining clear separation of concerns.

The data layer is centered around PostgreSQL, which stores the system's authoritative structured data, including users, vehicles, requests, offers, trips, payments, ratings, and audit records. Verification files are stored separately in object storage, while only their metadata and file references are kept in the database. In addition, the system uses Google Maps services for geolocation, route estimation, and travel-time calculation.

Overall, the proposed architecture is intended to support a secure, traceable, and legally compliant platform with strong emphasis on verification, trip-state integrity, and operational reliability. It provides a solid foundation for the current implementation of DriveMe while remaining flexible for future improvements.



3.2 Subsystem decomposition

The DriveMe system is decomposed into a set of major subsystems in order to separate responsibilities clearly and support maintainability. At a high level, the architecture consists of a mobile client subsystem, an application subsystem that contains the core business logic, a persistence subsystem responsible for durable storage, and external integration services used for mapping and routing.

The **Mobile Client Subsystem** is responsible for all user-facing interactions. It provides the interfaces used by passengers and drivers for registration, verification, ride request creation, offer handling, trip execution, and review submission. It also manages temporary local state and communicates with backend services through API calls.

The **Application Subsystem** contains the core logic of the platform. This layer is divided into functional areas such as authentication and user management, vehicle and driver verification, ride request management, matching and offer management, pricing, trip initiation and tracking, notification handling, review and rating management, and administrative auditing. These subsystems collaborate to support the full request-to-trip lifecycle while preserving business rules and transactional consistency.

The **Persistence Subsystem** provides durable and authoritative storage for structured application data. PostgreSQL is used to store users, vehicles, verification states, ride requests, offers, trips, ratings, payments, and audit records. Verification files and similar uploaded assets are stored separately in object storage, while their metadata is maintained in the database.

The **External Integration Subsystem** encapsulates communication with third-party services required by the platform. In the current design, this mainly includes Google Maps services for geolocation, route estimation, and travel-time calculation. By isolating these dependencies behind dedicated integration logic, the rest of the system remains less coupled to provider-specific details.

This decomposition allows DriveMe to organize its functionality into coherent units, reduce direct dependencies between unrelated concerns, and support future extension without requiring major structural changes.

3.3 Hardware/software mapping

3.3.1 Server-Side Infrastructure

- **Backend Framework:** Spring Boot is used to implement the backend services of DriveMe.

- **Application Structure:** The backend follows a modular monolithic architecture in which all core services are deployed within a single application while remaining logically separated.

- **Core Responsibilities:** The server side handles authentication, user and role management, vehicle and driver verification, ride request handling, matching and offer management, pricing, trip lifecycle control, notifications, reviews, and administrative auditing.

- **Execution Environment:** The backend is intended to run on a server or cloud-based environment and can be deployed in a containerized form for easier portability and maintenance.

3.3.2 Client-Side Infrastructure

- **Devices:** The client side runs on mobile devices used by passengers and drivers.
- **Mobile Framework:** Flutter is used to provide a single codebase for the mobile application.
- **Client Responsibilities:** The mobile client supports registration, login, verification submission, ride request creation, offer browsing, trip execution, and post-trip review workflows.
- **Device Capabilities:** The application uses device capabilities such as location services, network connectivity, and limited local storage for temporary state and short-term caching.

3.3.3 Data and Storage Infrastructure

- **Relational Database:** PostgreSQL is used as the main persistence technology for structured and transactional system data.
- **Stored Data:** The database stores users, vehicles, verification states, ride requests, offers, trips, payments, ratings, notifications, and audit records.
- **File Storage:** Verification documents and similar uploaded assets are stored separately in object storage.
- **Storage Strategy:** Only metadata and file references are stored in the relational database, while the actual files remain in object storage.

3.3.4 External Service Infrastructure

- **Mapping Services:** DriveMe uses Google Maps services for geolocation, route estimation, and travel-time calculation.
- **Usage Areas:** These services support ride request creation, pricing, nearby request discovery, and trip tracking workflows.

3.3.5 DevOps and Deployment Infrastructure

- **Development Environment:** Local development is supported through Docker-based setups.

- **CI/CD:** GitHub Actions can be used to automate build, test, and deployment tasks.

- **Deployment Style:** The backend can be deployed as a containerized application in development and production environments.

- **Environment Separation:** The system is designed to support separate development, staging, and production configurations.

3.4 Persistent data management

DriveMe requires persistent storage for structured operational data such as users, vehicles, verification records, ride requests, offers, trips, payments, ratings, notifications, and audit logs. PostgreSQL is used as the main persistence technology because the platform depends on strongly related entities and transaction-safe state transitions throughout the trip lifecycle. A relational database is especially suitable for preserving referential integrity and maintaining consistency during critical operations such as request matching, trip initiation, trip completion, and payment confirmation. Uploaded verification documents and similar assets are stored separately in object storage, while only their metadata, ownership information, verification status, and file references are maintained in the database. This approach separates transactional system data from binary file storage, improves storage efficiency, and supports a more maintainable persistence design. In addition, the system keeps historical and traceability-related records such as route logs, review data, and audit events in order to support safety monitoring, dispute resolution, and administrative oversight. Overall, the persistent data management strategy of DriveMe is designed to ensure consistency, auditability, and secure handling of both structured data and uploaded records.

3.5 Access control and security

DriveMe uses role-based access control to protect sensitive user, vehicle, and trip data. The system defines passenger, driver, and administrator roles, and each role can access only the functions and records relevant to its responsibilities. Authentication is handled through Spring Security with JWT-based session management, while authorization rules prevent users from accessing data that does not belong to them. Since the platform processes identity information, verification documents, trip records, and location data, stored credentials are protected with secure password hashing. In addition, audit logs are maintained for critical actions such as verification decisions and trip state changes to support traceability and administrative oversight.

4. Subsystem Services

4.1 Mobile Client Subsystem

Purpose: Provides the primary Android-based user interface for passengers, drivers, and administrators and serves as the entry point for all user-facing workflows.

Responsibilities:

- Present screens for registration, login, profile management, verification, ride requests, offers, trip execution, notifications, and reviews.
- Collect and validate user input before submitting requests to backend services.
- Render current system state and workflow progress in a clear and accessible way.
- Maintain temporary local state and limited cached data to improve responsiveness and short-term resilience to connectivity loss.

Key Operations:

- User registration and login submission
- Profile update submission
- Vehicle and driver document upload
- Ride request creation and cancellation
- Offer browsing and acceptance
- Trip start verification and trip completion
- Review submission
- Notification display

Inputs / Outputs:

- **Inputs:** User interactions, form data, uploaded documents, device location, notification events, backend responses.
- **Outputs:** API requests, validated user input payloads, rendered views, local cached state, user feedback messages.

Data Owned: Owns transient client-side UI state, local session context, unsent temporary form data, and short-term cached presentation data. It does not own authoritative business data.

Dependencies / Integrations: Consumes services from the Authentication and User Management Service, Vehicle and Driver Verification Service, Ride Request Management Service, Matching and Offer Management Service, Pricing Service, Trip Initiation and Verification Service, Trip Tracking and Lifecycle Service, Notification and Alert Service, and Review and Rating Service. Uses device-level capabilities such as location access and local storage.

4.2 Authentication and User Management Service

Purpose: Manages user identity, authentication, session control, profile records, and role-based access across the system.

Responsibilities:

- Register passengers, drivers, and administrators.
- Authenticate users and issue secure session tokens.
- Support logout, password updates, and profile updates.
- Maintain user roles and authorization context.
- Enforce access restrictions for protected resources and operations.
- Support account lifecycle actions consistent with privacy requirements.

Key Operations:

- Register user
- Authenticate user
- Issue / validate JWT
- Update password
- Update profile
- Terminate session
- Process account deletion request

Inputs / Outputs:

- **Inputs:** Registration data, login credentials, password change requests, profile update requests, account management requests.
- **Outputs:** Authentication result, JWT/session context, user profile data, authorization context, account status changes.

Data Owned: Owns user identity records, credentials metadata, role assignments, profile information, and session-related authentication data.

Dependencies / Integrations: Uses the Data Persistence Layer for durable storage. Provides authenticated user context to the Vehicle and Driver Verification Service, Ride Request Management Service, Matching and Offer Management Service, Admin and Moderation Service, Notification and Alert Service, and Review and Rating Service. Works with security controls for endpoint protection.

4.3 Vehicle and Driver Verification Service

Purpose: Controls the submission, review lifecycle, and status management of vehicle and driver verification records.

Responsibilities:

- Accept vehicle documents from passengers and licence/identity documents from drivers.
- Validate submission completeness and format.
- Maintain verification statuses such as pending, approved, and rejected.
- Restrict access to operational workflows unless verification requirements are satisfied.
- Expose verification status to users and administrators.

Key Operations:

- Submit vehicle documents
- Submit driver documents
- Validate submission
- Update verification
- Retrieve verification
- Approve or reject submission

Inputs / Outputs:

- **Inputs:** Uploaded documents, authenticated user identity, admin review decision.

- **Outputs:** Verification record, verification status, eligibility flags for downstream services, rejection / approval result.

Data Owned: Owns vehicle verification records, driver verification records, document metadata, and verification status history.

Dependencies / Integrations: Consumes user identity from the Authentication and User Management Service. Receives submissions from the Mobile Client Subsystem. Exposes records to the Admin and Moderation Service for review. Stores verification records through the Data Persistence Layer. Supplies eligibility information to the Ride Request Management Service and Matching and Offer Management Service.

4.4 Ride Request Management Service

Purpose: Owns the creation, validation, storage, and lifecycle control of passenger ride requests.

Responsibilities:

- Create requests for passengers with verified vehicles.
- Store pickup, destination, vehicle, and request-related context.
- Validate request eligibility before publication to drivers.
- Support request viewing, update, and cancellation when permitted.
- Maintain request lifecycle states and ensure invalid requests do not enter the matching workflow.

Key Operations:

- Create ride request
- Retrieve ride request
- Update ride request
- Cancel ride request
- Validate request eligibility
- Transition request state

Inputs / Outputs:

- **Inputs:** Authenticated passenger context, verified vehicle status, pickup location, destination, selected vehicle, pricing request.
- **Outputs:** Ride request record, request state, request visibility to drivers, cancellation result, request update result.

Data Owned: Owns ride request records, request status, pickup and destination data, associated passenger and vehicle references, and request lifecycle metadata.

Dependencies / Integrations: Consumes passenger context from the Authentication and User Management Service, vehicle eligibility from the Vehicle and Driver Verification Service, and route-related data from the Mobile Client Subsystem and Mapping and Geolocation Integration Service. Requests price estimates from the Pricing Service. Publishes eligible requests to the Matching and Offer Management Service. Persists request state through the Data Persistence Layer and emits status changes to the Notification and Alert Service.

4.5 Matching and Offer Management Service

Purpose: Provides driver-side ride discovery and manages the offer submission and passenger acceptance workflow.

Responsibilities:

- Expose eligible nearby pending ride requests to verified drivers.
- Validate submitted offers against pricing constraints.
- Associate offers with the relevant request and driver.
- Allow passengers to review and accept one offer.
- Close competing offers after selection.
- Transition the request from pending to matched.

Key Operations:

- Discover nearby ride requests
- Submit offer
- Validate offer amount
- List offers for passenger
- Accept offer

- Finalize match
- Close remaining offers

Inputs / Outputs:

- **Inputs:** Pending request data, driver identity, driver verification status, driver location, pricing bounds, passenger acceptance action.
- **Outputs:** Offer record, visible request list, matched request result, driver notification event, closed-offer state.

Data Owned: Owns offer records, match records, driver-to-request associations, and offer acceptance/rejection status.

Dependencies / Integrations: Consumes pending requests from the Ride Request Management Service, driver eligibility from the Vehicle and Driver Verification Service, pricing bounds from the Pricing Service, and geographic context from the Mapping and Geolocation Integration Service. Persists offers and matches data through the Data Persistence Layer and emits match-related events to the Notification and Alert Service.

4.6 Pricing Service

Purpose: Calculates a recommended fare range for each ride request to support controlled price negotiation.

Responsibilities:

- Estimate price ranges using distance, travel time, and demand-related parameters.
- Provide consistent price recommendations across the platform.
- Supply pricing constraints used to validate driver offers.
- Retain pricing outputs for traceability where needed.

Key Operations:

- Calculate estimated fare range
- Retrieve route-based cost inputs
- Apply pricing rules
- Return minimum / maximum recommendation
- Validate pricing boundaries for offers

Inputs / Outputs:

- **Inputs:** Route distance, estimated travel duration, configurable pricing parameters, demand-related inputs.
- **Outputs:** Recommended minimum fare, recommended maximum fare, pricing metadata, validation boundaries.

Data Owned: Owns pricing parameters, pricing-rule configuration, and generated fare recommendation records where persistence is required.

Dependencies / Integrations: Consumes route and travel estimates from the Mapping and Geolocation Integration Service. Supplies price ranges to the Ride Request Management Service and Matching and Offer Management Service. Stores pricing outputs through the Data Persistence Layer when auditability is needed.

4.7 Trip Initiation and Verification Service

Purpose: Ensures that only the correct matched passenger and matched driver can start the trip.

Responsibilities:

- Generate a one-time credential for trip initiation.
- Validate the credential at the start of the trip.
- Prevent unauthorized or duplicate trip-start attempts.
- Transition a matched trip into the started state after successful verification

Key Operations:

- Generate verification code
- Validate verification code
- Authorize trip start
- Reject invalid initiation attempt
- Update trip state to started

Inputs / Outputs:

- **Inputs:** Match record, trip identifier, driver-submitted verification code, authenticated participant context.
- **Outputs:** Verification result, trip-start authorization result, updated trip state, trip-start event.

Data Owned: Owns trip initiation credentials, verification attempt records, and trip-start authorization metadata.

Dependencies / Integrations: Triggered by a successful match from the Matching and Offer Management Service. Reads trip and user context through the Data Persistence Layer. Activates the Trip Tracking and Lifecycle Service after successful verification and emits trip-start events to the Notification and Alert Service.

4.8 Trip Tracking and Lifecycle Service

Purpose: Manages active trip execution, real-time route progression, and completion-related state transitions.

Responsibilities:

- Activate tracking after a trip is started.
- Record periodic location and route progress during the trip.
- Support monitoring of trip continuity and route adherence.
- Validate destination proximity before trip completion.
- Manage recoverable and consistent state transitions from started to completed.

Key Operations:

- Start trip tracking
- Record GPS update
- Log route progression
- Validate destination proximity
- Complete trip
- Restore trip session after interruption

Inputs / Outputs:

- **Inputs:** Trip-start event, device location updates, route data, destination data, completion request.
- **Outputs:** Trip progress records, lifecycle state updates, completion result, route history, audit events.

Data Owned: Owns active trip state, trip progress records, route logs, destination-validation results, and trip completion metadata.

Dependencies / Integrations: Activated by the Trip Initiation and Verification Service. Uses location and device context from the Mobile Client Subsystem and routing support from the Mapping and Geolocation Integration Service. Persists trip data through the Data Persistence Layer, emits milestone events to the Notification and Alert Service, and sends operational records to the Logging and Audit Service. Supplies completed-trip eligibility to the Review and Rating Service.

4.9 Notification and Alert Service

Purpose: Provides event-driven communication to users during verification, matching, trip execution, and review workflows.

Responsibilities:

- Deliver push notifications and in-app alerts for important workflow events.
- Route each event to the correct user and device context.
- Support timely user feedback for both normal state transitions and exceptional conditions.

Key Operations:

- Send push notification
- Send in-app alert
- Route event to recipient
- Format notification content
- Deliver milestone notification

Inputs / Outputs:

- **Inputs:** Domain events from backend subsystems, recipient identity, device/session context, notification template parameters.
- **Outputs:** Push notification, in-app alert, delivery status, notification history entry.

Data Owned: Owns notification templates, delivery records, notification history, and delivery status metadata.

Dependencies / Integrations: Consumes events from the Authentication and User Management Service, Vehicle and Driver Verification Service, Ride Request Management Service, Matching and Offer Management Service, Trip Initiation and Verification Service, Trip Tracking and Lifecycle Service, Review and Rating Service, and Admin and Moderation Service. Uses recipient and device context stored through the Data Persistence Layer.

4.10 Review and Rating Service

Purpose: Supports post-trip passenger feedback and maintains driver rating summaries.

Responsibilities:

- Allow eligible passengers to submit ratings and optional reviews after completed trips.
- Prevent review submission for invalid or incomplete trips.
- Update aggregate driver ratings.
- Store review records for later display and moderation.

Key Operations:

- Submit rating
- Submit review
- Validate review eligibility
- Recalculate aggregate rating
- Retrieve driver rating summary

Inputs / Outputs:

- **Inputs:** Completed-trip reference, authenticated passenger context, rating value, optional review text.

- **Outputs:** Review record, updated rating summary, review-submission result.

Data Owned: Owns rating records, written reviews, aggregate driver scores, and review eligibility metadata.

Dependencies / Integrations: Consumes trip-completion eligibility from the Trip Tracking and Lifecycle Service. Persists review and rating data through the Data Persistence Layer. Exposes updated rating summaries to the Mobile Client Subsystem and makes relevant data available to the Admin and Moderation Service.

4.11 Admin and Moderation Service

Purpose: Provides privileged operational oversight for verification, audit inspection, moderation, and dispute-related actions.

Responsibilities:

- Review and decide on pending vehicle and driver verification submissions.
- Inspect audit records and completed trip data.
- Monitor suspicious activity and disputed cases requiring intervention.
- Enforce administrator-only operations across the platform.

Key Operations:

- Review verification submission
- Approve or reject verification
- Inspect audit trail
- View completed trip records
- Handle moderation action
- Review suspicious activity

Inputs / Outputs:

- **Inputs:** Administrator identity and role context, pending verification records, audit records, dispute-related data.
- **Outputs:** Approval / rejection decisions, moderation actions, administrative review results, notification-triggering events.

Data Owned: Owns administrative review decisions, moderation records, and privileged action history.

Dependencies / Integrations: Consumes administrator authorization context from the Authentication and User Management Service. Reads and updates verification data through the Vehicle and Driver Verification Service. Accesses historical records through the Logging and Audit Service and Data Persistence Layer. Emits administrative decision events to the Notification and Alert Service.

4.12 Data Persistence Layer

Purpose: Provides durable, transactional storage for the structured application data required by DriveMe.

Responsibilities:

- Store and retrieve core domain data across all business workflows.
- Preserve consistency during critical multi-step state transitions.
- Support durable recovery and backup of core application records.

Key Operations:

- Create, read, update, and delete persisted records
- Execute transactional updates
- Enforce referential consistency
- Support recovery and backup procedures

Inputs / Outputs:

- **Inputs:** Structured persistence requests from application services.
- **Outputs:** Stored records, query results, transactional commit / rollback results.

Data Owned: Owns authoritative persisted records for users, vehicles, verification status, ride requests, offers, trips, notifications, reviews, and audit data.

Dependencies / Integrations: Serves as the persistence backbone for the Authentication and User Management Service, Vehicle and Driver Verification Service, Ride Request Management Service, Matching and Offer Management Service, Pricing Service, Trip

Initiation and Verification Service, Trip Tracking and Lifecycle Service, Notification and Alert Service, Review and Rating Service, Admin and Moderation Service, and Logging and Audit Service.

4.13 Logging and Audit Service

Purpose: Captures operational, security-relevant, and compliance-relevant records for monitoring, debugging, and traceability.

Responsibilities:

- Record significant application events and exceptions.
- Maintain audit trails for authentication, verification, administrative actions, and trip lifecycle transitions.
- Support post-incident analysis, monitoring, and dispute resolution.
- Preserve traceability for sensitive actions and privileged operations.

Key Operations:

- Record application log
- Record exception
- Record audit event
- Retrieve audit trail
- Support operational inspection

Inputs / Outputs:

- **Inputs:** System events, exception events, authentication actions, admin actions, trip and verification state changes.
- **Outputs:** Log entries, audit records, traceability data, inspection-ready historical records.

Data Owned: Owns system logs, audit trails, exception records, and security-relevant event history.

Dependencies / Integrations: Consumes event data from the Authentication and User Management Service, Vehicle and Driver Verification Service, Ride Request Management Service, Matching and Offer Management Service, Trip Initiation and

Verification Service, Trip Tracking and Lifecycle Service, Notification and Alert Service, and Admin and Moderation Service. Persists these records through the Data Persistence Layer and exposes them to the Admin and Moderation Service.

4.14 Mapping and Geolocation Integration Service

Purpose: Encapsulates external mapping, routing, geocoding, and travel-estimation capabilities required by the platform.

Responsibilities:

- Retrieve map, route, travel time, geocoding, and reverse-geocoding data from external providers.
- Expose a normalized internal interface so that application services are isolated from third-party API details.
- Provide route and location data required by request creation, pricing, matching, and trip tracking.
- Isolate and report external-service failure conditions.

Key Operations:

- Geocode address
- Reverse-geocode coordinates
- Estimate route
- Retrieve distance and duration
- Return normalized location data
- Report provider error or rate-limit condition

Inputs / Outputs:

- **Inputs:** Addresses, coordinates, route origin and destination pairs, external API requests.
- **Outputs:** Coordinates, resolved address data, route geometry, estimated distance, estimated duration, provider status/error response.

Data Owned: Owns integration configuration, provider request/response handling rules, and optionally cached mapping response metadata where such caching is implemented.

Dependencies / Integrations: Integrates with Google Maps APIs and provides standardized outputs to the Mobile Client Subsystem, Ride Request Management Service, Matching and Offer Management Service, Pricing Service, and Trip Tracking and Lifecycle Service.

5. Test Cases

5.1 Functional Tests

5.1.1 User Authentication & Account Management Tests

Test ID	**F001**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a new user can successfully register with valid personal information, a unique email/phone, and proof of legal age.				
Steps	1. Open the DriveMe application on an Android device. 2. Tap 'Sign Up' on the welcome screen. 3. Select account type (Passenger or Driver). 4. Enter valid personal information: full name, date of birth (legal age), unique email, unique phone number. 5. Create a password that meets the system's requirements. 6. Submit the registration form.				
Expected Results	A new account is created successfully. The user receives a confirmation message and is redirected to their home screen. The email/phone is registered in the system.				
Results	No results for this report.				

Test ID	**F002**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a registered user can log in successfully with valid credentials.				
Steps	1. Open the DriveMe application. 2. Enter a valid registered username/email and correct password. 3. Complete 2FA verification if prompted. 4. Tap the 'Login' button.				
Expected Results	The user is authenticated and redirected to their home screen. Session is established securely.				
Results	No results for this report.				

Test ID	**F003**	**Category**	Functional	**Severity**	High
Objective	Verify that login is rejected when invalid credentials are provided.				
Steps	1. Open the DriveMe application. 2. Enter a valid email but an incorrect password. 3. Tap the 'Login' button.				
Expected Results	Login is denied. An appropriate error message is displayed. The user remains on the login screen.				
Results	No results for this report.				

Test ID	**F004**	**Category**	Functional	**Severity**	Medium
Objective	Verify that a logged-in user can edit their profile information and changes have persisted.				
Steps	1. Log in as a registered user. 2. Navigate to the 'Edit Profile' page. 3. Modify at least one field (e.g., phone number or display name). 4. Submit the updated form.				
Expected Results	The profile information is updated in the database. The updated details are reflected immediately on the profile page.				
Results	No results for this report.				

Test ID	**F005**	**Category**	Functional	**Severity**	Medium
Objective	Verify that a logged-in user can reset their password successfully.				
Steps	1. Log in as a registered user. 2. Navigate to the 'Reset Password' page. 3. Enter the current password and a new valid password. 4. Submit the form.				

Expected Results	The password is updated in the database. The user can subsequently log in using the new password.
Results	No results for this report.

5.1.2 Vehicle & Driver Registration Tests

Test ID	**F006**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a passenger can register a vehicle by uploading documentation, and the vehicle is flagged as 'WAITING APPROVAL'.				
Steps	1. Log in as a passenger. 2. Navigate to the 'Add Vehicle' page. 3. Enter vehicle details and upload the required documentation (e.g., registration, insurance). 4. Submit the vehicle registration form.				
Expected Results	The vehicle entry is created in the system with status 'WAITING APPROVAL'. A confirmation is shown to the passenger.				
Results	No results for this report.				

Test ID	**F007**	**Category**	Functional	**Severity**	Critical

Objective	Verify that an administrator can approve a vehicle, changing its status to 'VERIFIED'.
Steps	1. Log in as an administrator. 2. Navigate to the pending vehicle approvals list. 3. Select a vehicle with status 'WAITING APPROVAL'. 4. Review the documents uploaded and approve the vehicle.
Expected Results	The vehicle status is updated to 'VERIFIED'. The passenger is notified of the approval.
Results	No results for this report.

Test ID	**F008**	**Category**	Functional	**Severity**	High
Objective	Verify that an administrator can reject a vehicle registration.				
Steps	1. Log in as an administrator. 2. Navigate to the pending vehicle approvals list. 3. Select a vehicle with status 'WAITING APPROVAL'. 4. Review the documents and select 'Reject'.				
Expected Results	The vehicle status is updated to 'REJECTED'. The passenger is notified that the vehicle was not approved.				
Results	No results for this report.				

Test ID	**F009**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a driver can get verified by uploading licence and identity documents.				
Steps	1. Log in as a driver. 2. Navigate to the 'Get Verified' page. 3. Upload valid driver's licence and identity documents. 4. Submit the registration form.				
Expected Results	The driver's profile is flagged as 'WAITING APPROVAL'. A confirmation is shown. Administrator review is triggered.				
Results	No results for this report.				

5.1.3 Ride Request & Matching Tests

Test ID	**F010**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a verified passenger can create a ride request with vehicle, location, and destination, and the system displays an estimated price range.				
Steps	1. Log in as a passenger with a 'VERIFIED' vehicle. 2. Navigate to the home screen and tap 'Request a Ride'. 3. Confirm current location. 4. Select the destination on the map.				

	5. Select the registered and verified vehicle. 6. Submit the ride request.
Expected Results	A ride request is created. The system displays an estimated price range based on distance, time, and demand. The request status is set to 'PENDING'.
Results	No results for this report.

Test ID	**F011**	**Category**	Functional	**Severity**	High
Objective	Verify that a ride request cannot be created if the passenger has no verified vehicle.				
Steps	1. Log in as a passenger whose vehicle is in 'WAITING APPROVAL' or 'REJECTED' status. 2. Attempt to create a ride request.				
Expected Results	The system prevents the ride request creation and displays an appropriate message indicating that a verified vehicle is required.				
Results	No results for this report.				

Test ID	**F012**	**Category**	Functional	**Severity**	Critical

Objective	Verify that a verified driver can search for and view nearby pending ride requests.
Steps	1. Log in as a verified driver. 2. Navigate to the 'Search Requests' screen. 3. Allow location access. 4. View the list of nearby pending ride requests.
Expected Results	A list of pending ride requests within a suitable radius is displayed, showing destination, vehicle type, and price range for each.
Results	No results for this report.

Test ID	**F013**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a driver can submit a price offer for a selected ride request within the recommended price range.				
Steps	1. Log in as a verified driver. 2. Search for nearby ride requests and select one. 3. View the details and the recommended price range. 4. Enter an offer price within the recommended range. 5. Submit the offer.				
Expected Results	The offer is recorded and associated with the ride request. The driver sees a confirmation. The passenger is notified of the new offer.				

Results	No results for this report.

Test ID	**F014**	**Category**	Functional	**Severity**	High
Objective	Verify that a driver cannot submit an offer outside the recommended price range.				
Steps	1. Log in as a verified driver. 2. Select a pending ride request. 3. Enter a price that is outside the recommended range (either too low or too high). 4. Attempt to submit the offer.				
Expected Results	The system rejects the offer and shows an error indicating the price must be within the recommended range.				
Results	No results for this report.				

Test ID	**F015**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a passenger can view all offers on their request and accept one, triggering a 'MATCHED' status.				

Steps	1. Log in as a passenger with an active ride request that has received at least one offer. 2. Navigate to the 'Offers' screen for the request. 3. Review the available offers. 4. Accept one offer.
Expected Results	The ride request status changes to 'MATCHED'. The accepted driver receives a notification that their offer was accepted. Other offers are no longer visible.
Results	No results for this report.

5.1.4 Trip Lifecycle Tests

Test ID	**F016**	**Category**	Functional	**Severity**	Critical
Objective	Verify that the system generates a unique 4-digit code for a matched trip that initiates the trip when scanned/entered.				
Steps	1. Complete a match between a passenger and driver (ride status is 'MATCHED'). 2. Both passenger and driver navigate to the trip initiation screen. 3. The passenger's app displays the 4-digit verification code. 4. The driver scans or enters the code.				
Expected Results	The trip status is updated to 'TRIP STARTED'. Both parties receive confirmation that the trip has begun. GPS tracking is activated.				

Results	No results for this report.

Test ID	**F017**	**Category**	Functional	**Severity**	Critical
Objective	Verify that real-time GPS tracking is active and visible during a trip.				
Steps	1. Start a trip (status = 'TRIP STARTED'). 2. As the driver moves the vehicle, observe the trip tracking screen. 3. Check that the route is being logged.				
Expected Results	The GPS position is updated in real time on the map for both passenger and driver. The route is logged by the system for safety monitoring.				
Results	No results for this report.				

Test ID	**F018**	**Category**	Functional	**Severity**	Critical
Objective	Verify that a passenger can end the trip, the system validates destination proximity, the driver confirms payment, and the ride is marked 'COMPLETED'.				
Steps	1. During an active trip (status = 'TRIP STARTED'), navigate to the in-trip screen. 2. Tap the 'End Ride' button. 3. System verifies the passenger's current location matches the destination.				

	4. Driver receives a notification and taps to confirm receipt of cash payment. 5. Both parties confirm trip completion.
Expected Results	The ride status is updated to 'COMPLETED'. A cash transaction record is logged. Both parties see a trip completion confirmation screen.
Results	No results for this report.

Test ID	**F019**	**Category**	Functional	**Severity**	Medium
Objective	Verify that a passenger cannot end a trip if they are not at (or near) the destination.				
Steps	1. During an active trip, tap 'End Ride' when the passenger is not at the destination.				
Expected Results	The system rejects the end-trip request and notifies the passenger that they must be at the destination to complete the ride.				
Results	No results for this report.				

5.1.5 Ratings & Notifications Tests

Test ID	**F020**	**Category**	Functional	**Severity**	High

Objective	Verify that a passenger can rate a driver after a completed trip and the rating is reflected on the driver's profile.
Steps	1. Log in as a passenger who has at least one completed trip. 2. Navigate to trip history and select a completed trip. 3. Submit a star rating and optional review for the driver. 4. Confirm submission.
Expected Results	The rating is saved and the driver's cumulative rating on their profile is updated accordingly.
Results	No results for this report.

Test ID	**F021**	**Category**	Functional	**Severity**	High
Objective	Verify that the driver receives notifications when their offer is accepted, when the trip starts, and when it ends.				
Steps	1. Submit an offer as a driver. 2. Have the passenger accept the offer and observe the driver's device. 3. Initiate the trip via code and observe the driver's device. 4. Complete the trip and observe the driver's device.				
Expected Results	The driver receives timely push notifications at each milestone: offer accepted, trip started, and trip ended.				

Results	No results for this report.

5.1.6 Pricing Engine Tests

Test ID	**F022**	**Category**	Functional	**Severity**	Critical
Objective	Verify that the pricing engine generates a recommended price range based on distance, time, and demand.				
Steps	1. Log in as a passenger with a verified vehicle. 2. Create a ride request with varying distances (short, medium, long). 3. Observe the estimated price range displayed for each request.				
Expected Results	The system displays a price range for each request that varies meaningfully with distance. The range is shown before the request is submitted.				
Results	No results for this report.				

5.1.7 Admin & Audit Tests

Test ID	**F023**	**Category**	Functional	**Severity**	High
Objective	Verify that completed trip data and cash transaction values are logged for audit purposes.				
Steps	1. Complete a full trip lifecycle (request → match → start → complete).				

	2. Log in as an administrator. 3. Navigate to the audit/transaction log.
Expected Results	The completed trip record, including trip ID, passenger, driver, route, and cash transaction amount, is visible and accurate in the audit log.
Results	No results for this report.

Test ID	**F024**	**Category**	Functional	**Severity**	High
Objective	Verify that session management prevents unauthorized access to user accounts.				
Steps	1. Log in as a user. 2. Copy the session token. 3. Log out. 4. Attempt to access a protected endpoint/screen using the old session token.				
Expected Results	The system rejects access with the invalidated session token. The user is redirected to the login screen.				
Results	No results for this report.				

5.2 Non-Functional Tests

5.2.1 Performance and Scalability Testing

Test ID	**NF001**	**Category**	Performance	**Severity**	Critical
Objective	Application Startup Time				
Steps	1.Clear the application cache on the test device. 2. Force stop the application.. 3.Launch the app and start a timer. 4.Stop the timer when the dashboard is fully loaded.				
Expected Results	< 2.0 seconds in 95% of cases.				
Results	No results for this report.				

Test ID	**NF002**	**Category**	Performance	**Severity**	Medium
Objective	Image Loading Speed				
Steps	1. Upload a 2MB image to the profile section.. 2.Request the image from the server. 3. Measure the time until the image is fully visible. 4.Check for any UI freezes during loading.				
Expected Results	Rendering completed within 1.5s.				

Results	No results for this report.

Test ID	**NF003**	**Category**	Performance	**Severity**	High
Objective	Load Tolerance				
Steps	1.Simulate 1000 virtual users using a load testing tool. 2. Execute login and search actions simultaneously. 3. Monitor the server success rate. 4.Confirm no system crashes occur under peak load.				
Expected Results	Stable operation with 1000 concurrent users.				
Results	No results for this report.				

Test ID	**NF004**	**Category**	Performance	**Severity**	High
Objective	API Latency				
Steps	1.Create a script to send 100 consecutive requests to the API. 2.Log the response time for each request.				

	3.Calculate the average latency. 4. Verify the average is below 400ms.
Expected Results	Average response time for data fetching must be < 400ms.
Results	No results for this report.

Test ID	**NF005**	**Category**	Performance	**Severity**	Medium
Objective	Database Growth				
Steps	1.Insert 100,000 mock records into the database. 2. Run complex filtering and search queries. 3. Measure the execution time for each query. 4. Compare performance with a smaller dataset.				
Expected Results	No significant lag with 100k+ trip records.				
Results	No results for this report.				

Test ID	**NF006**	**Category**	Performance	**Severity**	High
Objective	Session Management				
Steps	1.Trigger 500 login attempts within 5 seconds. 2. Monitor server CPU and RAM usage. 3. Inspect logs for dropped connections. 4. Verify all sessions stay active.				
Expected Results	No dropped connections during traffic spikes.				
Results	No results for this report.				

5.2.2 Security and Authentication Testing

Test ID	**NF007**	**Category**	Security	**Severity**	Critical
Objective	Data Encryption				
Steps	1.Use a network proxy tool to capture traffic. 2.Perform a transaction with sensitive data. 3. Inspect the captured data packets. 4. Confirm all data is encrypted and unreadable.				

Expected Results	All transit data must be encrypted with TLS/SSL.
Results	No results for this report.

Test ID	**NF008**	**Category**	Security	**Severity**	Critical
Objective	Authentication				
Steps	1.Access a protected API endpoint without a token. 2. Send the request via a terminal or network tool. 3. Inspect the response status code. 4. Confirm the system returns 401 Unauthorized.				
Expected Results	Access without JWT must return a 401 Unauthorized error.				
Results	No results for this report.				

Test ID	**NF009**	**Category**	Security	**Severity**	High

Objective	Password Policy
Steps	1.Open the registration screen. 2. Enter "123" as the password. 3. Click the submit button. 4. Verify the system rejects the weak password.
Expected Results	Rejection of weak passwords like “123”.
Results	No results for this report.

Test ID	**NF010**	**Category**	Security	**Severity**	High
Objective	Resource Authorization				
Steps	1.Log in with a valid account. 2. Attempt to delete another users car record. 3. Check the backend response. 4. Confirm the system returns 403 Forbidden.				
Expected Results	Users cannot access or delete other users' data.				

Results	No results for this report.

Test ID	**NF011**	**Category**		**Severity**	
Objective	Data Privacy (KVKK Compliance)				
Steps	1.Select "Delete My Account" in settings. 2. Confirm the deletion process. 3. Access the database management panel. 4. Verify all personal data is permanently removed.				
Expected Results	Full data clean upon account deletion request.				
Results	No results for this report.				

Test ID	**NF012**	**Category**	Security	**Severity**	High
Objective	Consent Management				

Steps	1.Start the sign up process. 2. Leave the "Terms of Service" box unchecked. 3. Try to click the "Sign Up" button. 4. Verify the button is disabled or blocked.
Expected Results	Mandatory Terms acceptance before sign-up.
Results	No results for this report.

5.2.3 Reliability and Accuracy Testing

Test ID	**NF013**	**Category**	Reliability	**Severity**	High
Objective	Network Persistence				
Steps	1.Start an active request on a Wi-Fi network. 2. Disable Wi-Fi to force a switch to mobile data. 3. Observe the request status. 4. Confirm the session remains active.				
Expected Results	Seamless transition between Wi-Fi and 4G/LTE.				

Results	No results for this report.

Test ID	**NF014**	**Category**	Reliability	**Severity**	High
Objective	Data Recovery				
Steps	1.Start a trip on a 2km route. 2. Log the GPS points from the application. 3. Compare points with physical landmarks on a map. 4. Calculate the deviation margin.				
Expected Results	Active trip restoration after an application crash.				
Results	No results for this report.				

Test ID	**NF015**	**Category**	Accuracy	**Severity**	Critical
Objective	GPS Precision				
Steps	1.Start a trip on a pre-measured 2km route. 2. Log the GPS points from the application. 3. Compare points with physical landmarks on a map.				

	4. Calculate the deviation margin.
Expected Results	Location deviation must be less than 10m in open areas.
Results	No results for this report.

Test ID	**NF016**	**Category**	Accuracy	**Severity**	High
Objective	Pricing Logic				
Steps	1.Input a 10km trip into the app. 2. Calculate the price manually using the formula. 3. Compare the manual price with the app's price. 4. Verify the margin of error is below 0.1%.				
Expected Results	Calculation error margin within ±0.1%.				
Results	No results for this report.				

5.2.4 Usability Test

Test ID	**NF017**	**Category**	Usability	**Severity**	High
Objective	UI Responsiveness				
Steps	1.Open the app on a phone and a tablet. 2. Compare the layout on both screens. 3. Check for overlapping buttons or text. 4. Verify all features are accessible on both devices.				
Expected Results	Consistent layout across all screen aspect ratios.				
Results	No results for this report.				

Test ID	**NF018**	**Category**	Usability	**Severity**	Medium
Objective	User Feedback & Error Prevention				
Steps	1.Initiate a trip request or update a profile. 2. Check for a loading indicator during the process. 3. Inspect notification messages for success or failure. 4. Verify messages correctly guide the user.				

Expected Results	Clear visual feedback for all actions within 500ms.
Results	No results for this report.

6. Consideration of Various Factors in Engineering Design

6.1 Constraints

This section discusses the constraints of this project in detail. Since the practical aspects extend to the real world, solutions that are offered will most likely be affected by various factors. Below is a table that displays possible factors that might affect the design process.

Factor	Effect Level	Effect
Global Factors	8	Take different road maps and documentation needs for different countries and regions into consideration.
Economical Factors	9	The price that the application offers shall be within a reasonable economical range and the local currency.
Public Safety	10	● The drivers that the app lists must be legally accountable for traffic safety. ● The vehicles must undergo proper maintenance and should be able to provide proof if necessary.
Environmental Factors	5	The roads that are taken or proposed by the application shall minimize gas consumption in an eco-friendly approach.

Factor	Effect Level	Effect
Accessibility	7	The application should be accessible to the elderly and the disabled since they might be possible users

TABLE 1: FACTORS TABLE

6.1.1. Developmental Constraints

- **No Online Payment Services:** Following CS491/492 guidelines, the system cannot integrate credit cards, digital wallets, or payment gateways; only cash transaction logging is permitted.
- **Mobile-Only Deployment:** The project scope limits functional deployment to mobile platforms (Android), restricting development of web-based front-end features.
- **Dependence on Third-Party APIs:** Geolocation and real-time tracking rely on external services (Google Maps). API limitations, outages, or pricing changes directly constrain system functionality.

6.1.2 Legal and Regulatory Constraints

- **Compliance with Turkish Personal Data Protection Law (KVKK):** Since the system processes personal data such as names, phone numbers, location information, and driver license photos, all database design, storage, and access-control mechanisms must comply with KVKK requirements. This imposes constraints on data retention, encryption, access permissions, and auditability.
- **Restrictions on Shared-Ride / Commercial Transport Services in Türkiye:** Turkish transportation regulations prohibit informal ride-sharing or peer-to-peer commercial passenger transport. Therefore, the DriveMe platform cannot support shared rides, pooled trips, or commercial taxi-like operations. The application must restrict itself to the allowed scenario of driver services where a driver operates the user's own car, not shared passenger transportation.

6.1.3 Technological Constraints

- **Internet Connection:** The application will need internet connection to function properly and in-real-time.
- **Location/GPS Access:** The application will need to access the location of the user in certain scenarios like the location tracking through the trip.
- **Device and OS Limitations:** The system must perform reliably on standard mid-range Android devices due to limited testing hardware availability. The user devices will need at least 4 GBs of RAM and Android 8.0+.

6.1.4 Sustainability Constraints

- **Database Enlargement:** In the case of unexpectedly high volume of users or vehicles the databases may need to be enlarged, which will cause a higher cost for renting servers.
- **Annual Fees:** The SAAS's and possible memberships, as well as app distribution service registrations like Google Play Store are needed to be paid each year.

6.2 Standards

The development of DriveMe is being conducted in a way that all the source code and documentation follow these standards:

- IEEE 1471 is a documentation standard that focuses on system architecture. It is a recommended framework for software systems like this as per its software-intensive structure. This style offers the ability to perceive and understand the architecture, its components, and the relationships between the components. It highlights the actors and different parties that need to be considered while making some critical decisions about the architecture and/or the system. The architectural documentation follows IEEE 1471 standard.
- UML is a standardized modeling language that is often used to create Object-Oriented documentation models for software systems. The three types that are offered in UML are structural diagrams, behavioral diagrams and interactional diagrams. Each of these categories have

various sub-types that may be useful for capturing different aspects and functions of a software system.

- ACM Code of Ethics and Professional Conduct is followed by all engineers/developers to maintain a professional lifecycle.
- IEEE Citation Style is used for each document and their references.

7. Teamwork Details

7.1 Contributing and functioning effectively on the team

Throughout the project, our team demonstrated a fair contribution of team members. Each member contributed significantly to improving DriveMe.

Duru Solakoğlu worked on how to eliminate legal constraints with the help of a specialist. Additionally, she designed and implemented the frontend while ensuring the project has an user-centric and visually appealing UI design.

Berfin Çetinkaya contributed extensively to the backend design and implementation. She contributed to the backend side of the project, including the design of the system's data models, database structure, and core service architecture. She also contributed to the preparation of the architecture-related parts of this report. In addition, she took a role in the technical design and implementation of several core features throughout the project.

Ufuk Baran Güler took a major role in the technical implementation of the mobile application. He led the mobile development process and implemented the integration between the frontend mobile client and backend services. He worked on connecting API endpoints, handling communication between the Flutter application and backend services, and ensuring that major functionalities such as authentication, ride requests, and trip lifecycle operations work correctly within the mobile environment. He also contributed to the preparation of the architecture and subsystem service descriptions in the Detailed Design Report.

Eda Alparslan mostly contributed to the backend implementation and business logic. She created the RestAPI endpoints and the data layer architecture in the backend. She determined the use cases and the related test cases as well user stories. She proofread the reports as well as contributing in the Test Cases section of this report.

Ege Kaan contributed to the backend implementation by assisting with business logic and data layer operations. His primary responsibility in the Detailed Design Report was conducting the current software architecture analysis of competitors. He also developed a comprehensive set of non functional test cases with detailed procedures to ensure system quality. Additionally, he supported the team in refining the system's behavioral models.

7.2 Helping creating a collaborative and inclusive environment

Responsibilities within the DriveMe project were assigned based on individual interest to maximize engagement and motivation of team members. Project progression is systematically monitored through communication and management tools such as WhatsApp, Zoom, and Google Teams, ensuring that every member actively participates in the development and decision-making process.

We also hold regular weekly meetings through Zoom or Google Teams to discuss the progress of the project, review completed work, and plan upcoming tasks. These meetings are arranged through our WhatsApp group, which we use for quick communication and coordination. In addition, we manage our tasks using Jira, where we list project tasks and backlogs and track the progress of development. Team members can choose a task from the Jira board and assign it to themselves. This helps distribute the workload fairly and keeps everyone aware of what others are working on. Occasionally, we also meet face-to-face to work together, discuss technical problems, and write code collaboratively. These sessions help us solve issues faster and improve cooperation within the team.

7.3 Taking lead role and sharing leadership on the team

Since project parts are assigned with respect to individual interest, we worked as a cross-functional and self-managing team.

Duru Solakoğlu took a lead role in designing the user interaction aspect of DriveMe, and she led the legal aspect of our project.

Berfin Çetinkaya led the backend side development and contributed to the application in areas ranging from feature development to database design and core service architecture.

Ufuk Baran Güler led the mobile application development and the integration of backend services with the mobile client. He coordinated the implementation of core functionalities on the mobile side and ensured that communication between the frontend application and backend APIs was properly established.

Eda Alparslan led the team work and arranged the team meetings, planning the work distribution among team members. She ensured that the data layers of the backend stayed organized and in correct order.

Ege Kaan led the technical validation and competitive analysis phases of the project. He spearheaded testing strategy and defined the system's performance and security standards.

8. Glossary

Administrator: A privileged system user responsible for approving or rejecting driver and vehicle verification requests, monitoring system logs, and overseeing platform operations.

Audit Log: A record of important system events such as authentication attempts, document approvals, trip state transitions, and transaction confirmations.

Driver: A verified professional user of the system who can browse ride requests, submit price offers, and drive a passenger's vehicle.

Matching: The process of associating a passenger's ride request with one of the submitted driver offers.

Offer: A price proposal submitted by a verified driver in response to a passenger's ride request.

Passenger: A user who owns or has access to a vehicle and requests a professional driver through the system.

Pricing Engine: The subsystem responsible for calculating and recommending a reasonable price range for a trip based on distance, estimated duration, and demand conditions.

Ride Request: A request created by a passenger specifying pickup location, destination, and the vehicle to be driven.

Trip Lifecycle: The sequence of states a trip goes through, including pending request, matched driver, trip started, and trip completed.

Trip Tracking: The monitoring of a vehicle's location and route progress during an active trip using GPS data.

Verification Code: A temporary one-time code generated by the system to confirm the identities of the passenger and driver before starting the trip.

Vehicle Verification: The process of validating vehicle documentation submitted by passengers to ensure that only approved vehicles can be used in the system.

9. References

[1] J. Xu, R. Rahmatizadeh, L. Bölöni and D. Turgut, "Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2572-2581, Aug. 2018, doi: 10.1109/TITS.2017.2755684.

[2] Rayle, Lisa, et al. "App-based, on-demand ride services: Comparing taxi and ridesourcing trips and user characteristics in san francisco university of california transportation center (uctc)." *University of California, Berkeley, United States* 2 (2014): 49-52.