

Software Engineering Project Milestone 3

About: Electric Buses Tracking App



Department of Computer Science
Namal University, Mianwali

Prepared by:

Mehroz Ali Khan	NUM-BSCS-2024-34
Ali Abbas	NUM-BSCS-2024-06
Laiba Taj	NUM-BSCS-2024-31

18 January 2026

Contents

1	Introduction	2
2	Design Assumptions and Constraints	2
3	Design Assumptions	2
3.1	Design Constraints	2
3.2	Key Design Decisions	2
4	System Design Diagrams	3
4.1	DFD Level 0 (Context Diagram)	3
4.1.1	DFD Level 1	3
4.1.2	DFD Level 2 (Tracking Decomposition)	4
4.2	Activity Diagrams	5
4.2.1	Passenger Activity Diagram	5
4.2.2	Driver Activity Diagram	6
4.2.3	Admin Activity Diagram	7
4.3	Sequence Diagrams	8
4.3.1	Real-Time Tracking Sequence Diagram	8
4.3.2	Authentication Sequence Diagram	9
4.4	Class Diagrams	11
4.4.1	Management Class Diagram	11
4.4.2	Information Providing Class Diagram	11
4.5	Component Diagrams	12
4.5.1	Client Component Diagram	12
4.5.2	Server Component Diagram	13
4.6	Use Case Diagram	14
5	Conclusion	16

1 Introduction

This report presents the system design of the Electric Bus Location Tracker App developed for Mianwali, Pakistan. The objective of this milestone is to transform the Software Requirements Specification (SRS) into a structured and scalable system architecture.

The system enables real-time tracking of electric buses, estimated arrival times (ETA), duty management for drivers, and monitoring tools for administrators. The design ensures low latency, scalability, security, and compliance with data protection laws.

2 Design Assumptions and Constraints

3 Design Assumptions

- Smartphones have working GPS and internet connectivity
- Punjab Mass Authority (PMA) provides route and bus data
- Google Maps API is available for map visualization
- Drivers share location continuously during duty

3.1 Design Constraints

- Android version 8.0 or above
- Minimum internet bandwidth of 256 kbps
- GPS accuracy within 10 meters
- Compliance with Pakistan Personal Data Protection Bill
- Battery usage limited to 5% per hour

3.2 Key Design Decisions

A three-tier modular architecture is selected to ensure scalability and maintainability. Flutter is used for mobile development, Node.js for backend services, MongoDB for persistent storage, and Redis for caching real-time locations. WebSockets enable real-time updates, while JWT ensures secure role-based authentication.

4 System Design Diagrams

4.1 DFD Level 0 (Context Diagram)

The Level 0 Data Flow Diagram provides a high-level view of the entire system. It shows how the Electric Bus Tracker System interacts with external entities such as Passengers, Drivers, Administrators, GPS System, PMA Database, and Google Maps API. The diagram highlights the flow of location data, user requests, and system responses.

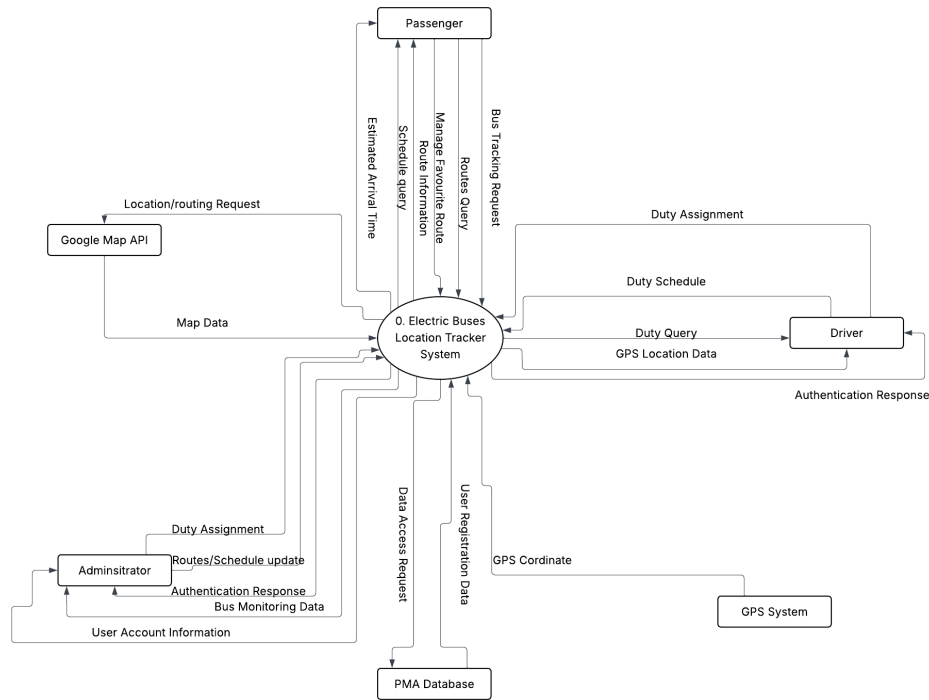


Figure 1: DFD Level 0 – Context Diagram

4.1.1 DFD Level 1

DFD Level 1 decomposes the system into major internal processes such as Authentication, Tracking Service, ETA Calculation, and Admin Management. It illustrates how data flows between processes and data stores like User Database, Bus Database, and Route Database.

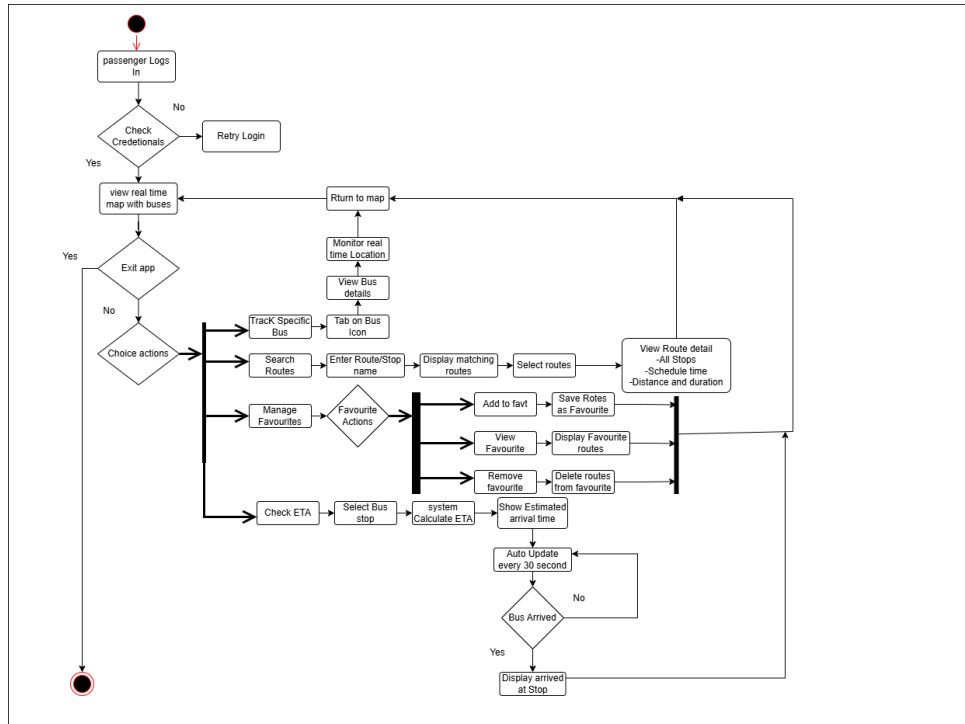


Figure 4: Passenger Activity Diagram

4.2.2 Driver Activity Diagram

The driver activity diagram illustrates the steps followed by a driver during duty. These steps include logging in, checking assigned duties, starting a trip, sharing GPS location continuously, and completing the duty upon reaching the endpoint.

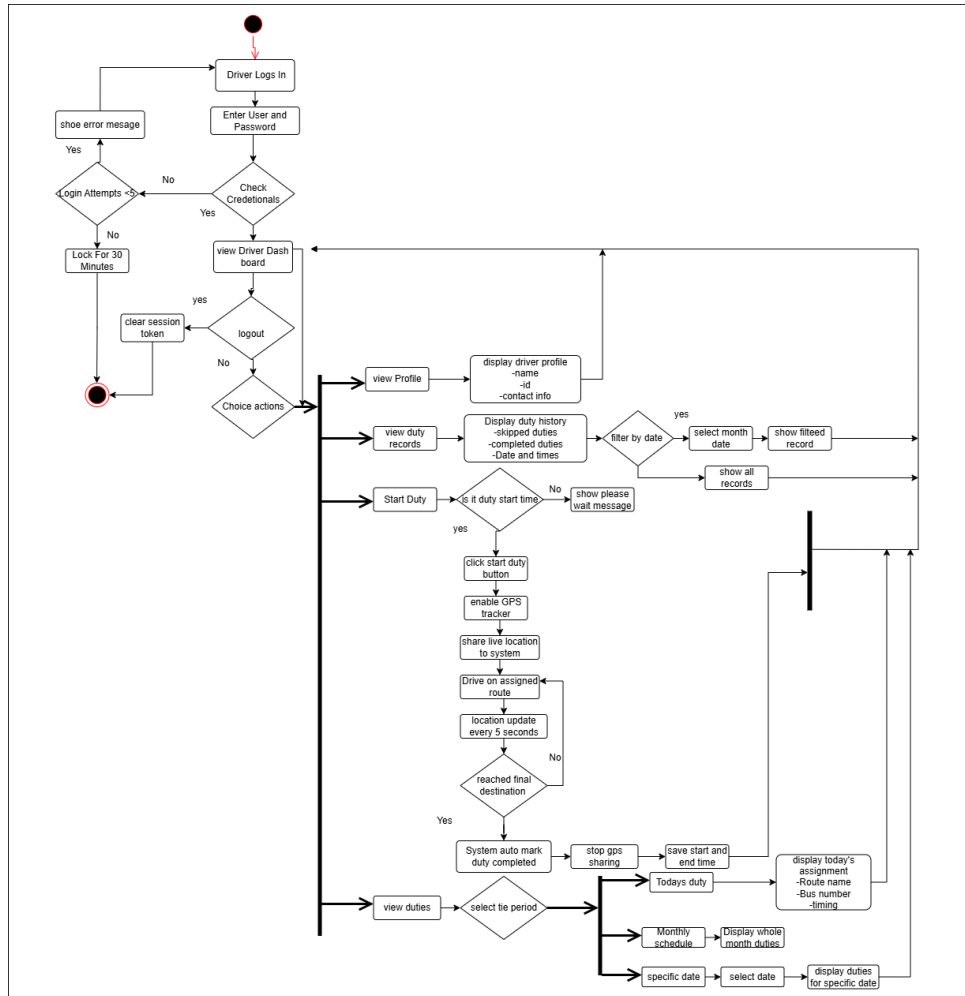


Figure 5: Driver Activity Diagram

4.2.3 Admin Activity Diagram

This diagram shows the admin workflow, including monitoring buses, assigning duties, managing drivers, and generating reports. The admin interacts with the system to ensure smooth operations and system oversight.

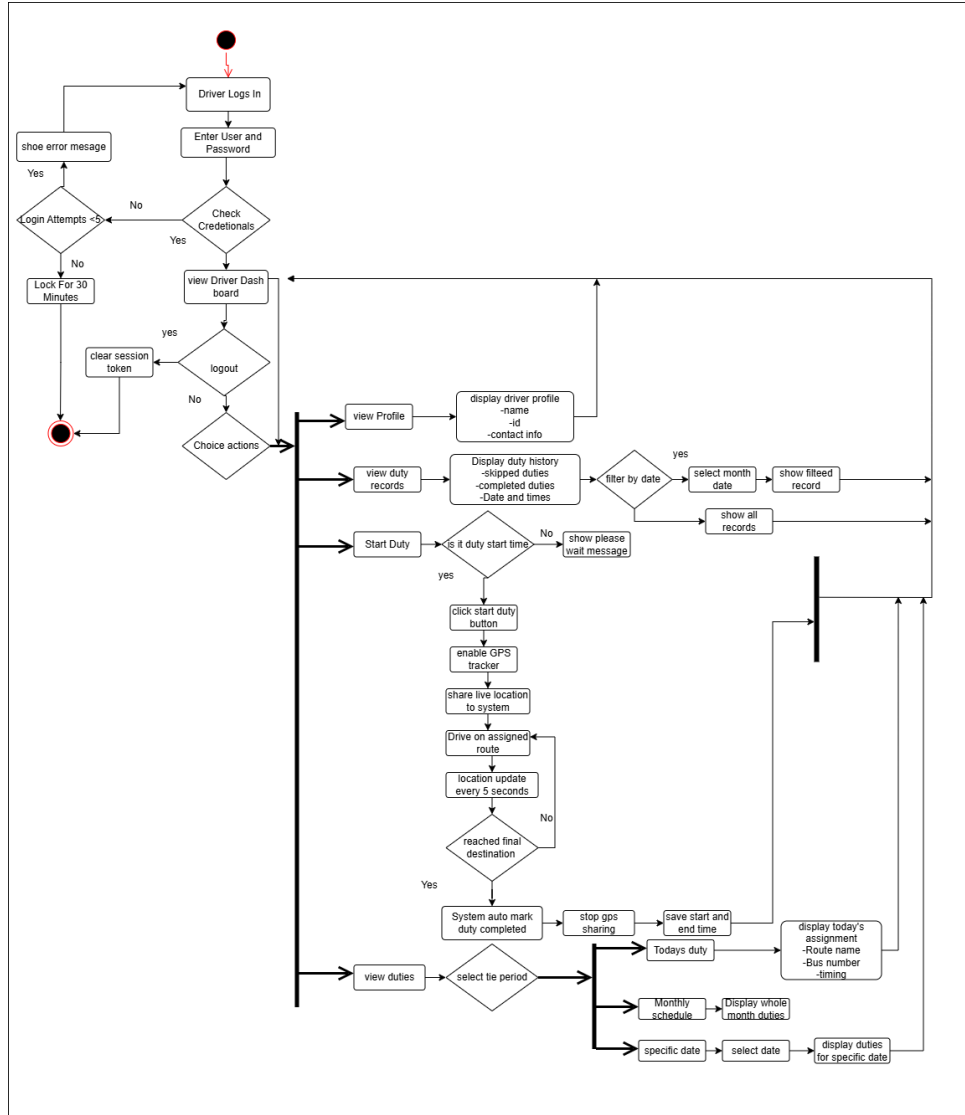


Figure 6: Admin Activity Diagram

4.3 Sequence Diagrams

4.3.1 Real-Time Tracking Sequence Diagram

This sequence diagram illustrates how real-time tracking works. The passenger requests live bus data, the backend retrieves cached locations from Redis, and updates are pushed to the client using WebSockets.

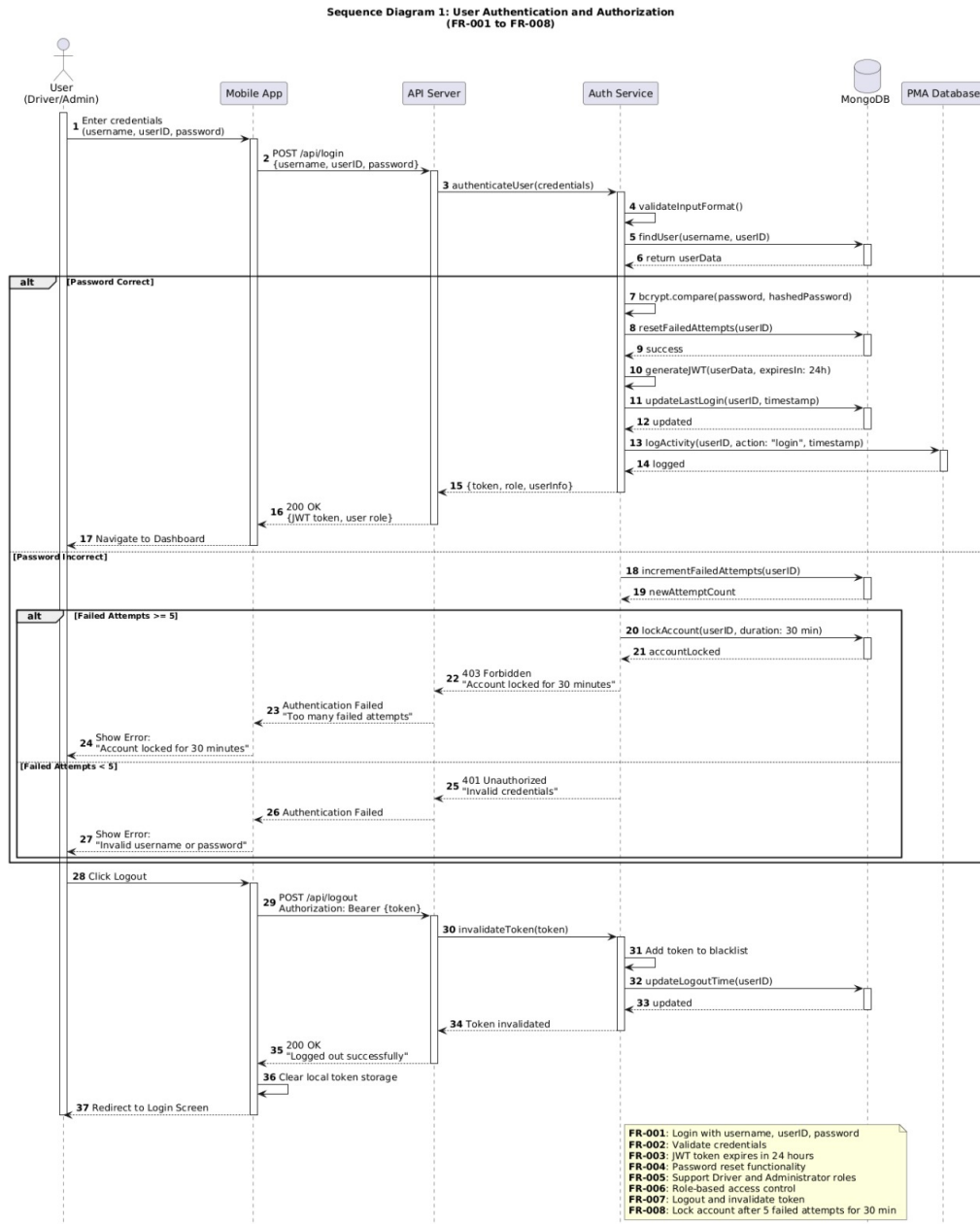


Figure 7: Real-Time Tracking Sequence Diagram

4.3.2 Authentication Sequence Diagram

The authentication sequence diagram shows the login process. User credentials are validated, passwords are verified using bcrypt, JWT tokens are generated, and role-based access is granted.

4.4.1 Management Class Diagram

```

classDiagram
    class User {
        +name: String
        +password: String
        +email: String
        +phone: String
        +vehicle: Vehicle
        +license: License
        +loginName: String
        +password: String
        +isAdmin: Boolean
        +isAdmin: Boolean
        +isAdmin: Boolean
    }
    class Passenger {
        +name: String
        +email: String
        +phone: String
        +license: License
        +license: License
        +license: License
        +license: License
        +license: License
        +license: License
        +license: License
    }
    class Driver {
        +name: String
        +password: String
        +email: String
        +phone: String
        +license: License
        +license: License
        +license: License
        +license: License
        +license: License
        +license: License
    }
    class Duty {
        +date: String
        +date: String
        +date: String
        +date: String
        +date: String
        +date: String
        +date: String
        +date: String
        +date: String
        +date: String
    }
    class Rule {
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
    }
    class Bus {
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
    }
    class Report {
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
    }
    class Authentication {
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
        +name: String
    }

    User "0" -- "1" Passenger
    User "0" -- "1" Driver
    User "0..1" -- "1..*" Report
    User "0..1" -- "1" Authentication
    User "1" -- "1" Duty
    User "1" -- "1" Rule
    User "1" -- "1" Bus
    User "1" -- "1" Report
    User "1" -- "1" Authentication
    
```

UML Class Diagram for a Taxi System:

- User** (Base Class):
 - Attributes: `name: String`, `password: String`, `email: String`, `phone: String`, `vehicle: Vehicle`, `license: License`, `loginName: String`, `password: String`, `isAdmin: Boolean`, `isAdmin: Boolean`, `isAdmin: Boolean`.
- Passenger** (Generalization):
 - Attributes: `name: String`, `email: String`, `phone: String`, `license: License`, `license: License`, `license: License`, `license: License`, `license: License`, `license: License`, `license: License`.
- Driver** (Generalization):
 - Attributes: `name: String`, `password: String`, `email: String`, `phone: String`, `license: License`, `license: License`, `license: License`, `license: License`, `license: License`, `license: License`.
- Duty** (Generalization):
 - Attributes: `date: String`, `date: String`, `date: String`, `date: String`, `date: String`, `date: String`, `date: String`, `date: String`, `date: String`, `date: String`.
- Rule** (Generalization):
 - Attributes: `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`.
- Bus** (Generalization):
 - Attributes: `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`.
- Report** (Generalization):
 - Attributes: `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`.
- Authentication** (Generalization):
 - Attributes: `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`, `name: String`.

Relationships:

- User** (0) is associated with **Passenger** (1).
- User** (0) is associated with **Driver** (1).
- User** (0..1) is associated with **Report** (1..*).
- User** (0..1) is associated with **Authentication** (1).
- User** (1) is associated with **Duty** (1).
- User** (1) is associated with **Rule** (1).
- User** (1) is associated with **Bus** (1).
- User** (1) is associated with **Report** (1).
- User** (1) is associated with **Authentication** (1).

Notes:

- Passenger: `license: License` is a license.
- Driver: `license: License` is a license.
- Duty: `date: String` is a date.
- Rule: `name: String` is a name.
- Bus: `name: String` is a name.
- Report: `name: String` is a name.
- Authentication: `name: String` is a name.

4.4.2 Information Providing Class Diagram

11

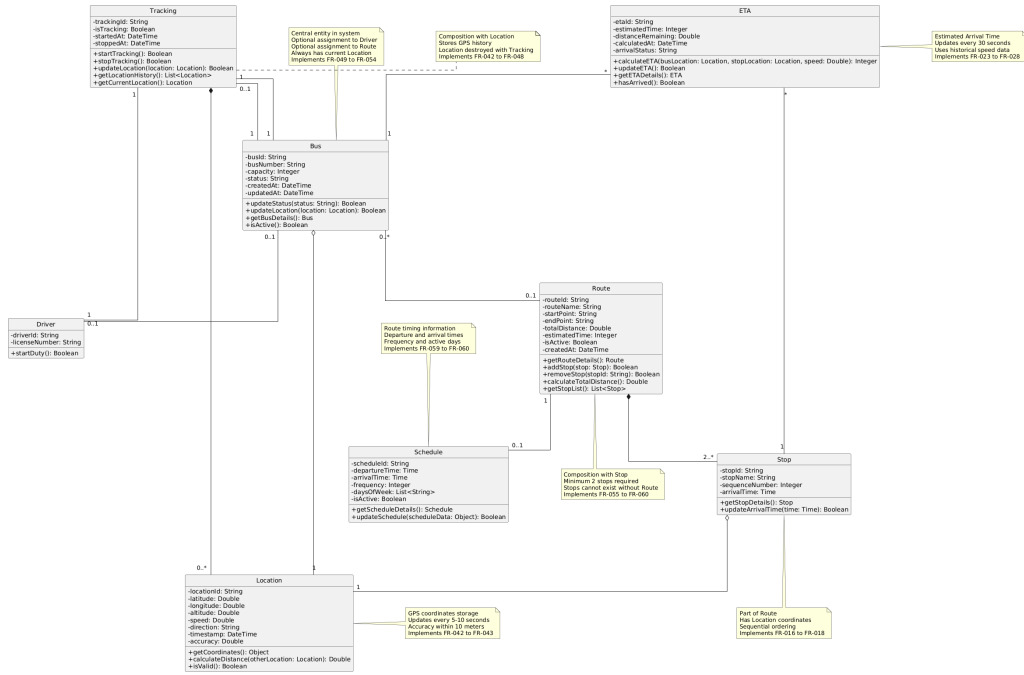


Figure 10: Information Providing Class Diagram

4.5 Component Diagrams

4.5.1 Client Component Diagram

This component diagram represents the internal architecture of the client-side mobile application. The application is composed of multiple UI components that allow users to interact with the system. It integrates the Google Maps SDK to display maps, routes, and real-time location information. The Socket.io client is used to enable real-time communication with the backend server for live updates such as driver location and trip status. Additionally, a local cache module is included to temporarily store data, improving performance and reducing network dependency.

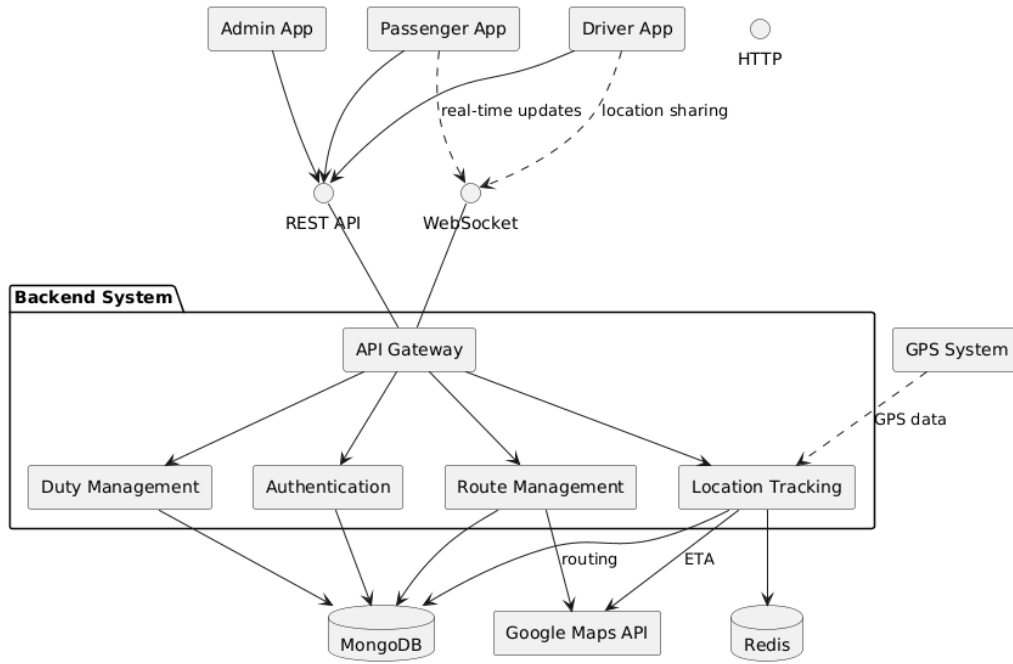


Figure 11: Client Component Diagram

4.5.2 Server Component Diagram

The server component diagram represents the layered architecture of the backend system. The Presentation Tier consists of the mobile application, which communicates with back-end services through well-defined interfaces. The Business Logic Tier includes core services such as Authentication, Bus Tracking, Route Management, and Duty Management, which handle the main system operations. The Data Access Tier provides a centralized data repository interface to manage data communication between business services and storage systems. The Persistence Tier includes MongoDB for persistent data storage and Redis for caching and fast data access. Additionally, the backend system integrates with external systems such as Google Maps for routing and map services and GPS for real-time location data

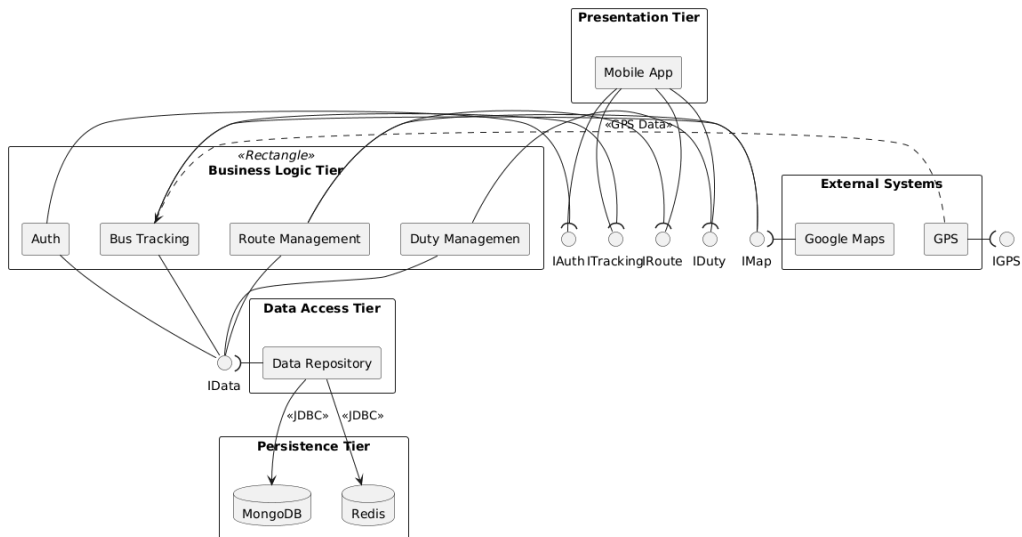


Figure 12: Server Component Diagram

4.6 Use Case Diagram

The use case diagram presents interactions between system actors and the system. Passengers track buses and view ETAs, drivers manage duties and share location, and admins monitor operations and generate reports.

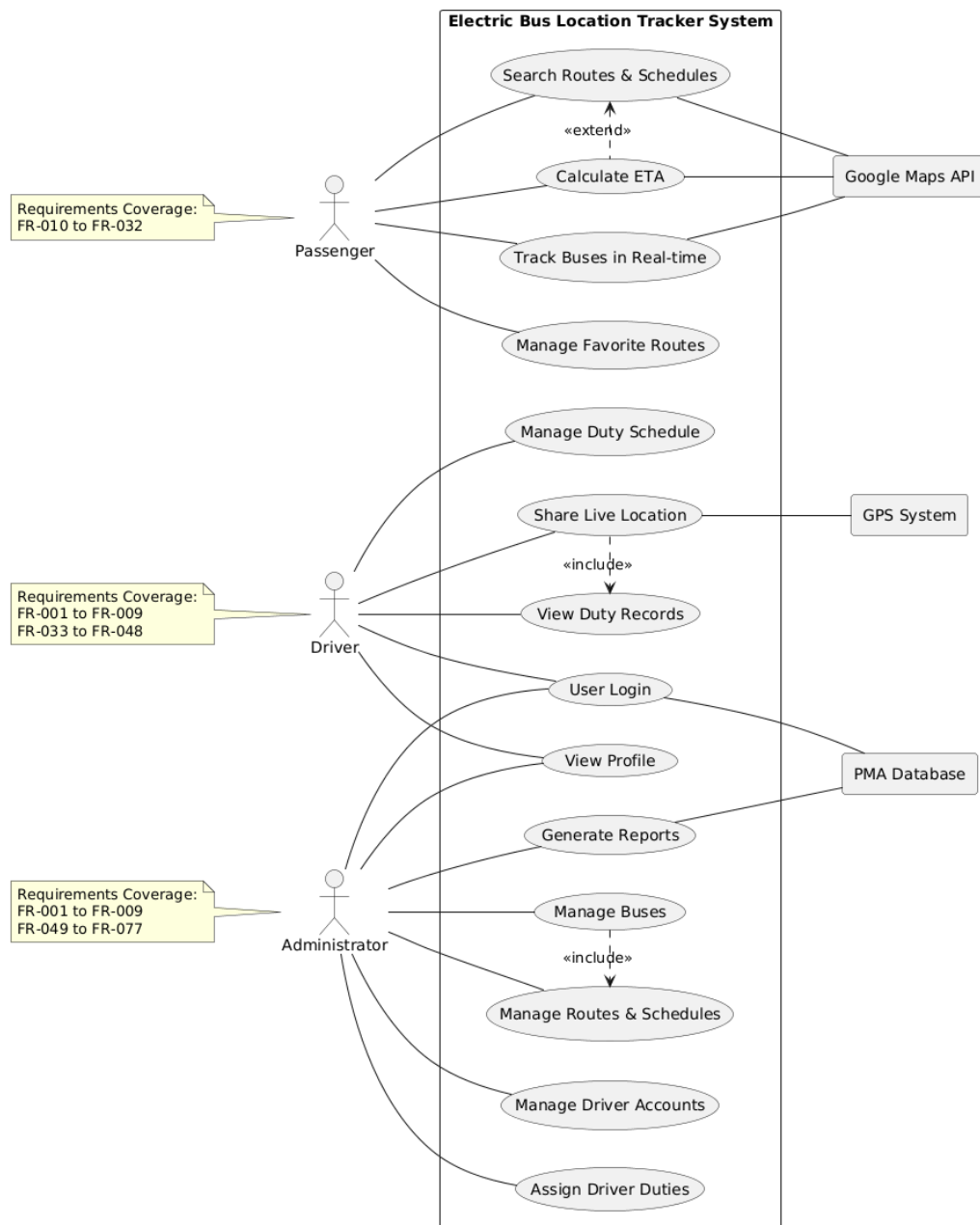


Figure 13: Use Case Diagram

Requirements–Design Traceability Matrix

Requirement ID	Description	Design Artifacts
FR-001–FR-008	Authentication and RBAC	Sequence Diagram, User Class
FR-010–FR-048	Real-time Tracking	DFD L1/L2, WebSockets
FR-014–FR-032	Route Search and Favorites	Activity Diagram, Information Providing class

FR-023–FR-028	ETA Calculation	ETA Engine, Sequence Diagram
FR-033–FR-071	Duty Management	Driver Activity, Duty Class
FR-049–FR-077	Admin Operations	Admin Activity Diagrams
NFR-001–NFR-069	Performance and Security	Entire System Design

5 Conclusion

The proposed system design successfully meets all functional and non-functional requirements specified in the SRS. The architecture ensures real-time performance, scalability, security, and future extensibility for electric bus tracking in Mianwali.