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

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.

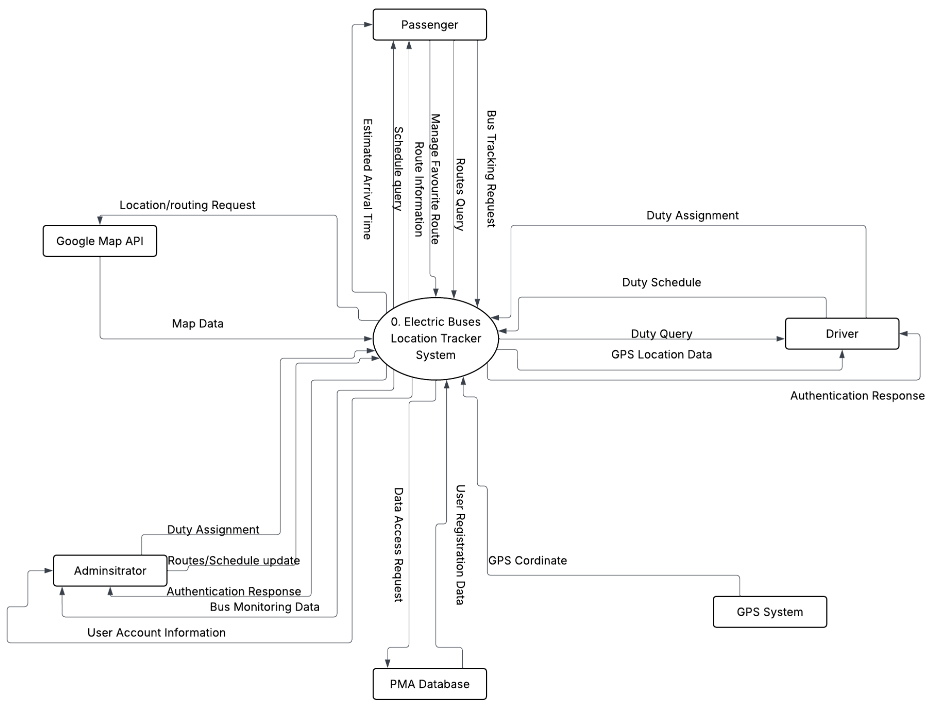


Figure 1: DFD Level 0 – Context Diagram

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.

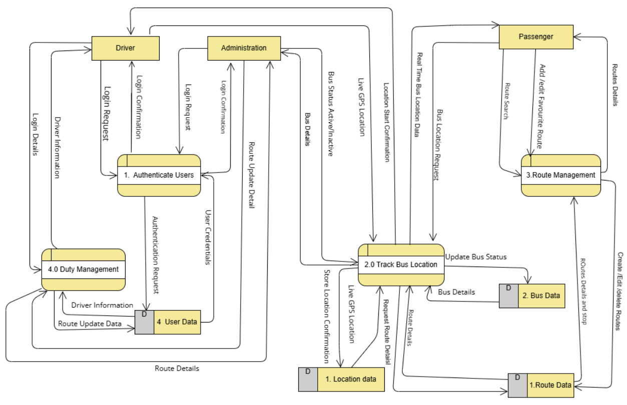


Figure 2: DFD Level 1 Diagram

4.1.2 DFD Level 2 (Tracking Decomposition)

DFD Level 2 further breaks down the tracking process into detailed sub-processes. These include location ingestion from drivers, real-time broadcasting using Redis and WebSockets, ETA computation, and map rendering using Google Maps API.

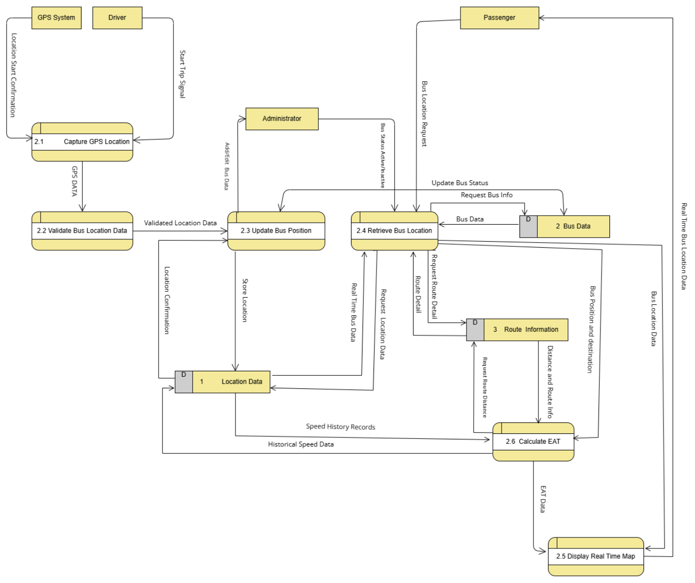


Figure 3: DFD Level 2 – Tracking Decomposition

4.2 Activity Diagrams

4.2.1 Passenger Activity Diagram

This activity diagram represents the workflow of a passenger using the application. The passenger logs in, views the live map, searches routes, checks ETAs, and manages favorite routes. Decision points handle network availability and user actions.

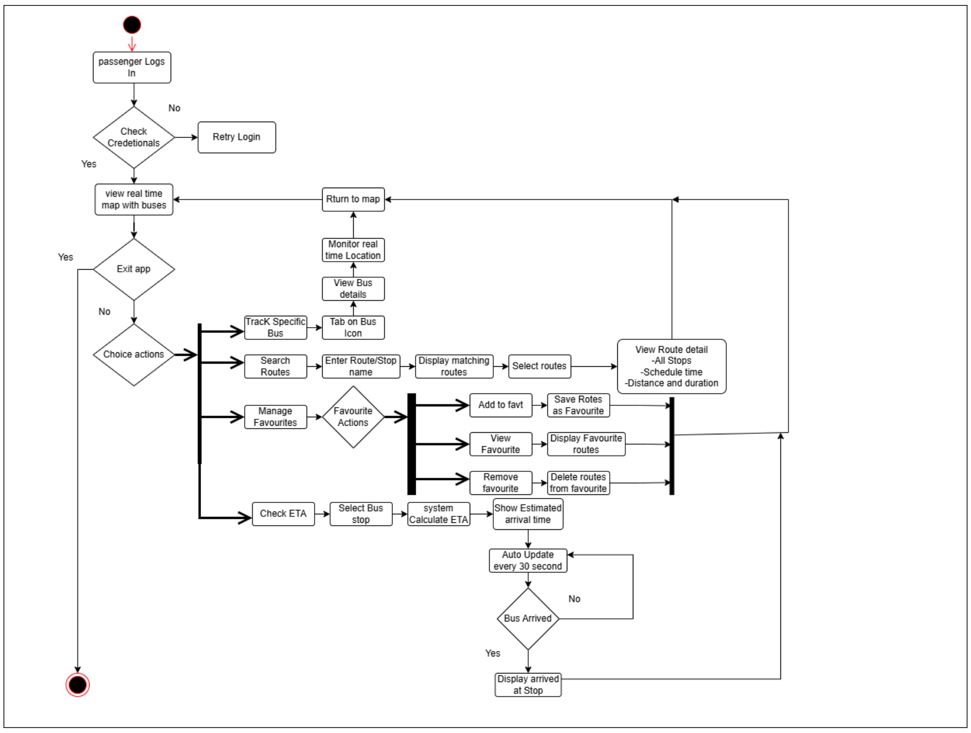


Figure 4: Passenger Activity Diagram

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.

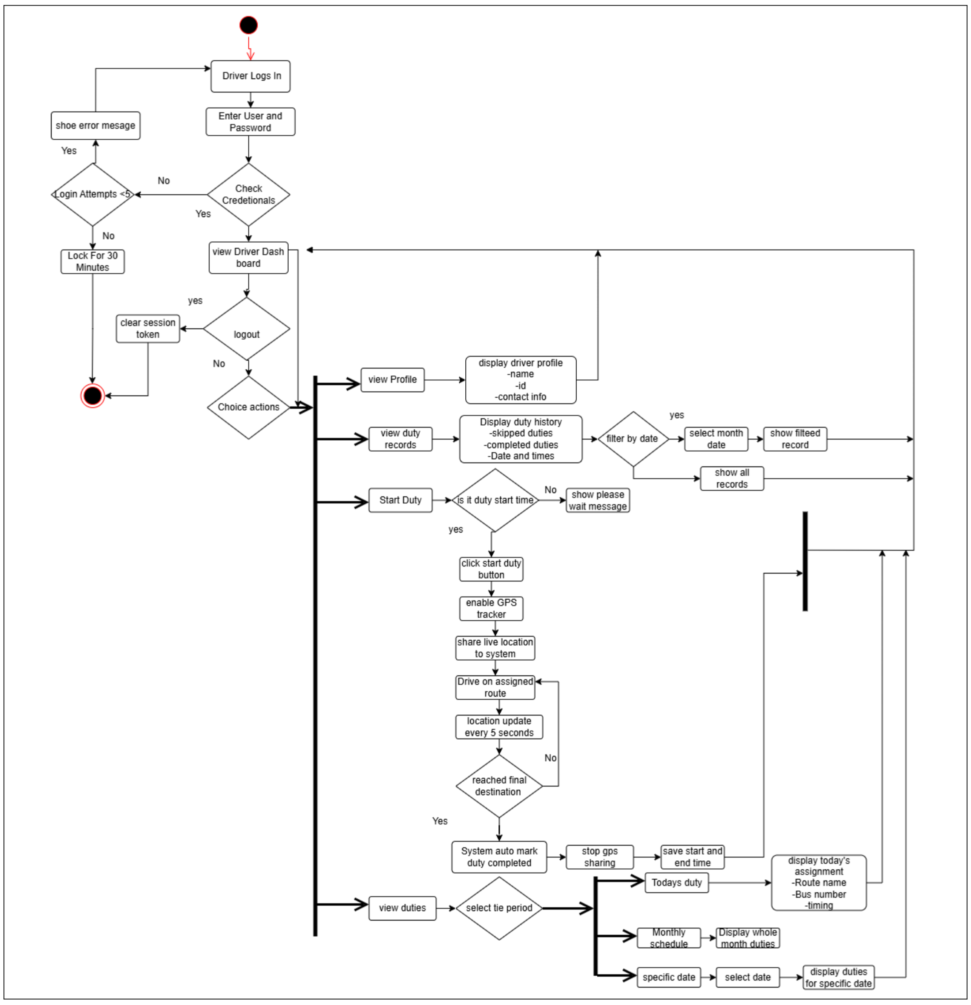


Figure 5: Driver Activity Diagram

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.

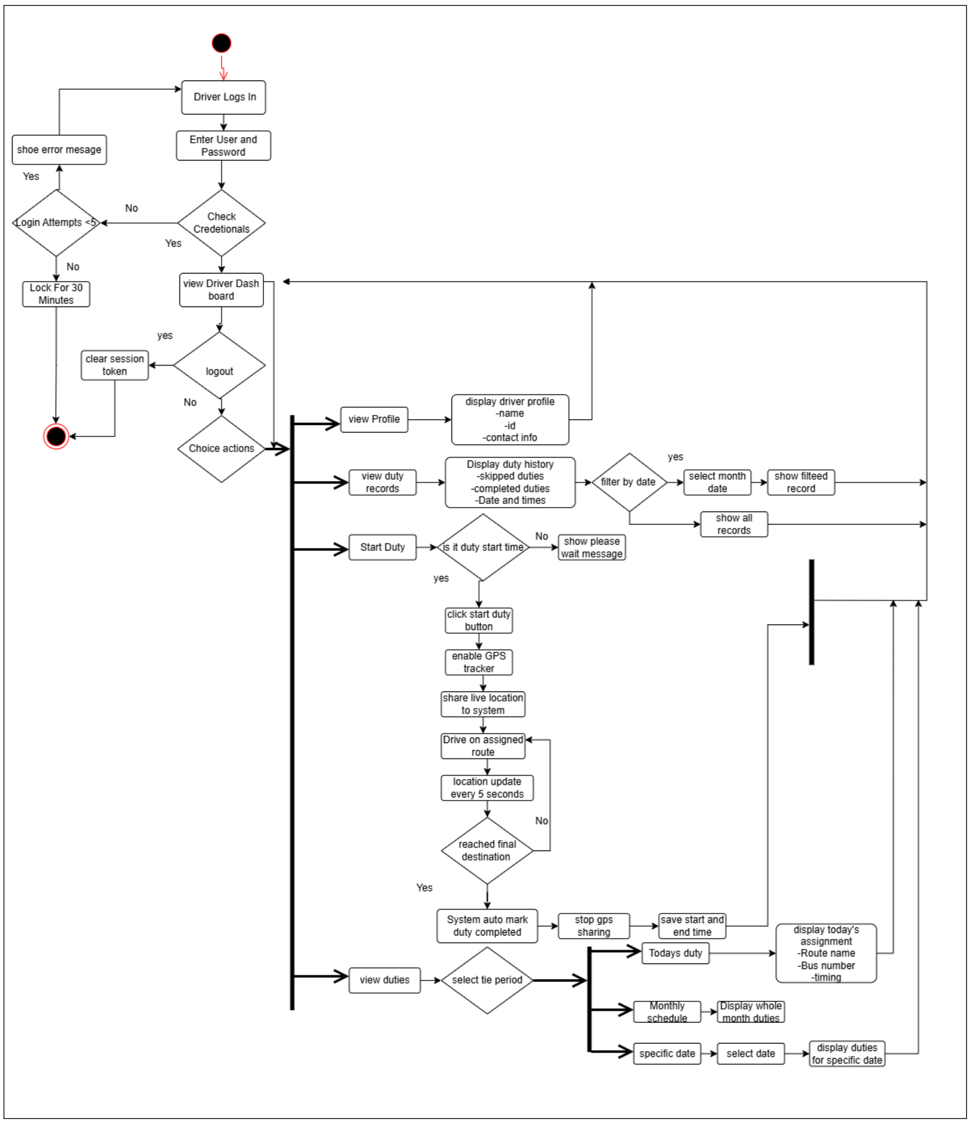


Figure 6: Admin Activity Diagram

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.

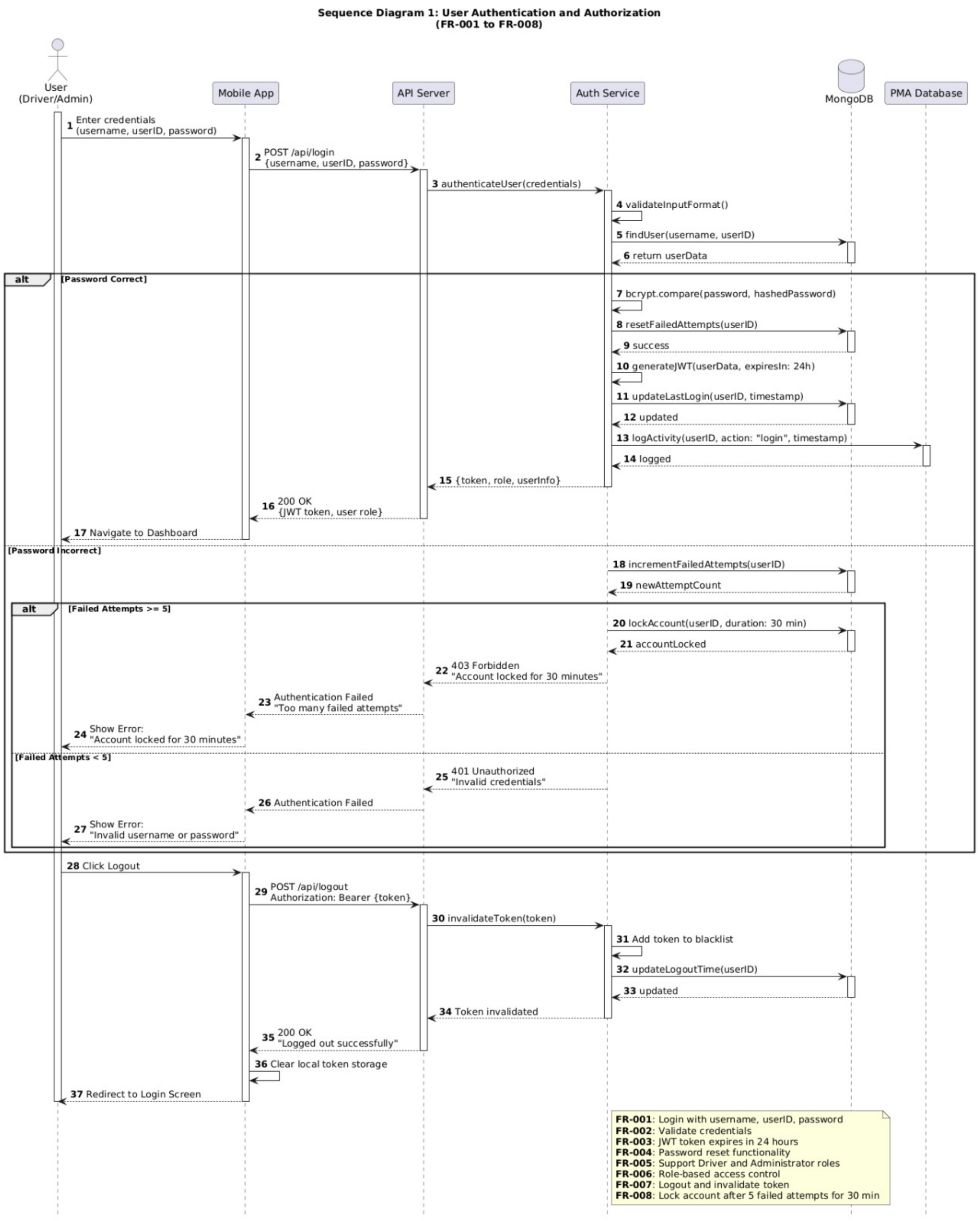

Sequence Diagram 1: User Authentication and Authorization
(FR-001 to FR-008)
User
(Driver/Admin)
Mobile App
API Server
Auth Service
MongoDB
PMA Database
1 Enter credentials
(username, userID, password)
2 POST /api/login
{username, userID, password}
3 authenticateUser(credentials)
4 validateInputFormat()
5 findUser(username, userID)
6 return userData
alt [Password Correct]
7 bcrypt.compare(password, hashedPassword)
8 resetFailedAttempts(userID)
9 success
10 generateJWT(userData, expiresIn: 24h)
11 updateLastLogin(userID, timestamp)
12 updated
13 logActivity(userID, action: "login", timestamp)
14 logged
15 {token, role, userInfo}
16 200 OK
{JWT token, user role}
17 Navigate to Dashboard
[Password Incorrect]
18 incrementFailedAttempts(userID)
19 newAttemptCount
alt [Failed Attempts >= 5]
20 lockAccount(userID, duration: 30 min)
21 accountLocked
22 403 Forbidden
"Account locked for 30 minutes"
23 Authentication Failed
"Too many failed attempts"
24 Show Error:
"Account locked for 30 minutes"
[Failed Attempts < 5]
25 401 Unauthorized
"Invalid credentials"
26 Authentication Failed
27 Show Error:
"Invalid username or password"
28 Click Logout
29 POST /api/logout
Authorization: Bearer {token}
30 invalidateToken(token)
31 Add token to blacklist
32 updateLogoutTime(userID)
33 updated
34 Token invalidated
35 200 OK
"Logged out successfully"
36 Clear local token storage
37 Redirect to Login Screen
FR-001: Login with username, userID, password
FR-002: Validate credentials
FR-003: JWT token expires in 24 hours
FR-004: Password reset functionality
FR-005: Support Driver and Administrator roles
FR-006: Role-based access control
FR-007: Logout and invalidate token
FR-008: Lock account after 5 failed attempts for 30 min

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.

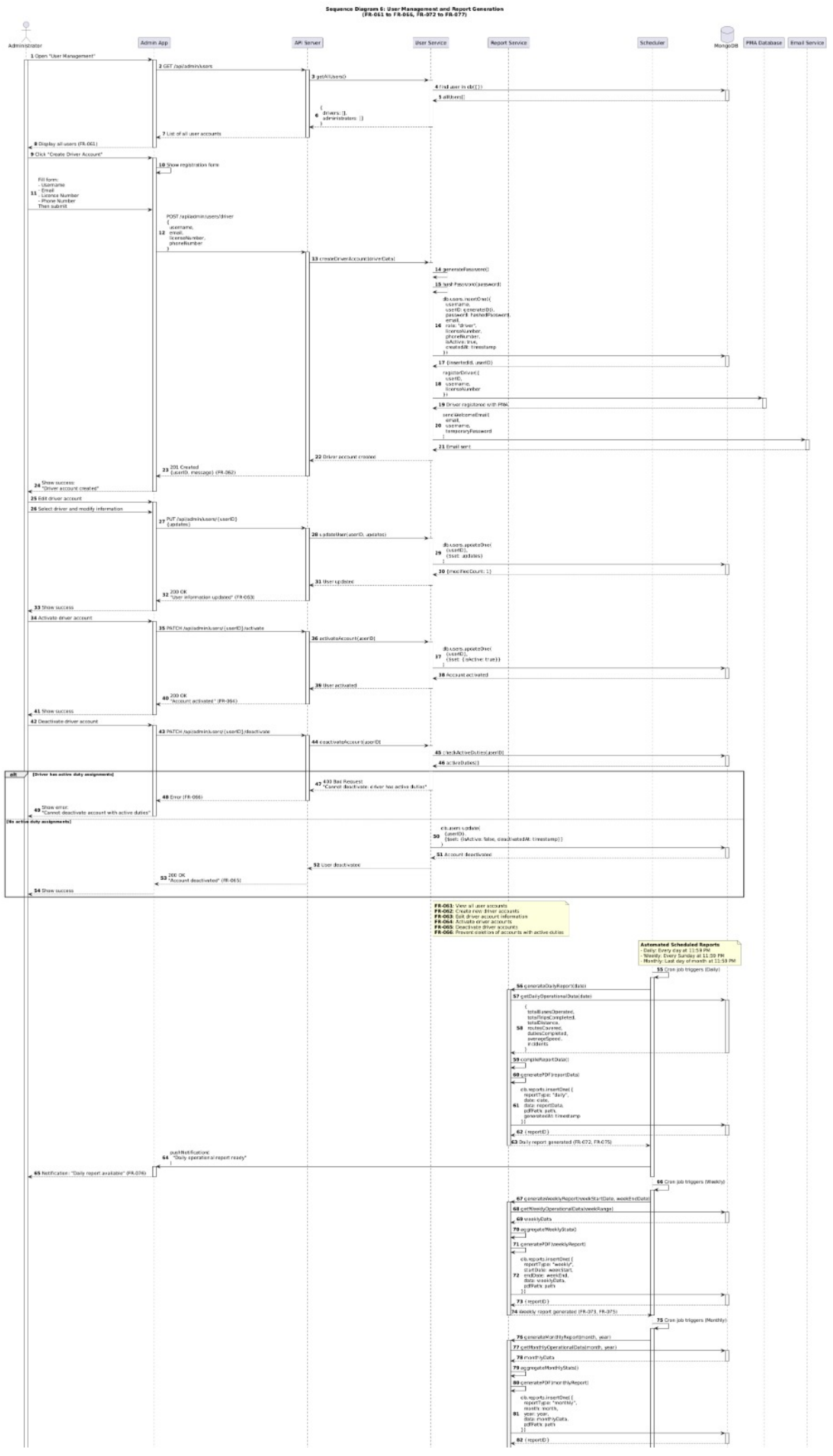
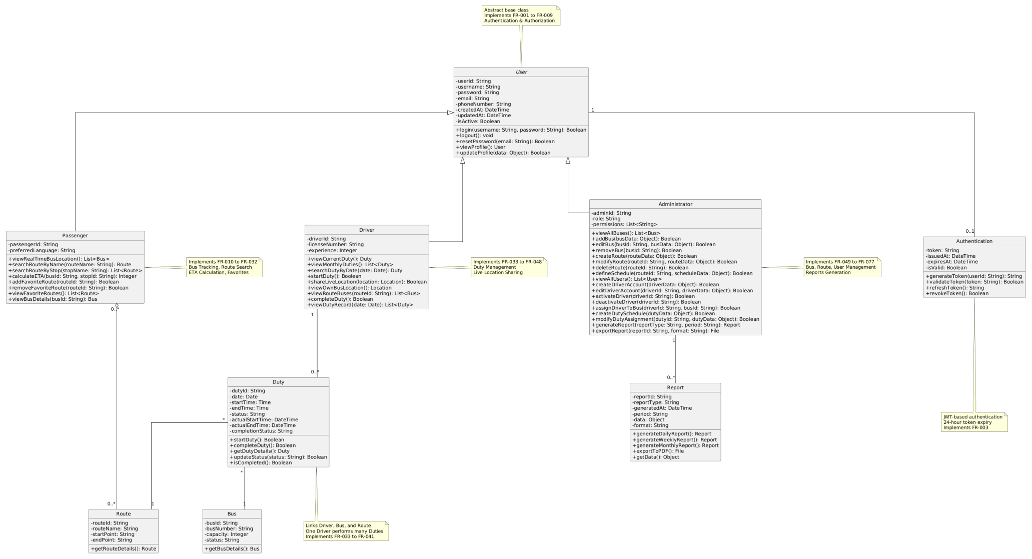

Sequence Diagram 6: User Management and Report Generation (FR-061 to FR-066, FR-072 to FR-077)
Administrator
Admin App
API Server
User Service
Report Service
Scheduler
MongoDB
PMA Database
Email Service
1 Open "User Management"
2 GET /api/admin/users
3 getAllUsers()
4 find user in db({})
5 allUsers[]
6 { drivers: [], administrators: [] }
7 List of all user accounts
8 Display all users (FR-061)
9 Click "Create Driver Account"
10 Show registration form
11 Fill form: - Username - Email - Licence Number - Phone Number Then submit
12 POST /api/admin/users/driver { username, email, licenseNumber, phoneNumber }
13 createDriverAccount(driverData)
14 generatePassword()
15 hashPassword(password)
16 db.users.insertOne({ username, userID: generateID(), password: hashedPassword, email, role: "driver", licenseNumber, phoneNumber, isActive: true, createdAt: timestamp })
17 {insertedId, userID}
18 registerDriver({ userID, username, licenseNumber })
19 Driver registered with PMA
20 sendWelcomeEmail(email, username, temporaryPassword)
21 Email sent
22 Driver account created
23 201 Created {userID, message} (FR-062)
24 Show success: "Driver account created"
25 Edit driver account
26 Select driver and modify information
27 PUT /api/admin/users/{userID} {updates}
28 updateUser(userID, updates)
29 db.users.updateOne({userID}, {$set: updates})
30 {modifiedCount: 1}
31 User updated
32 200 OK "User information updated" (FR-063)
33 Show success
34 Activate driver account
35 PATCH /api/admin/users/{userID}/activate
36 activateAccount(userID)
37 db.users.updateOne({userID}, {$set: {isActive: true}})
38 Account activated
39 User activated
40 200 OK "Account activated" (FR-064)
41 Show success
42 Deactivate driver account
43 PATCH /api/admin/users/{userID}/deactivate
44 deactivateAccount(userID)
45 checkActiveDuties(userID)
46 activeDuties[]
alt
[Driver has active duty assignments]
47 400 Bad Request "Cannot deactivate: driver has active duties"
48 Error (FR-066)
49 Show error: "Cannot deactivate account with active duties"
[No active duty assignments]
50 db.users.update({userID}, {$set: {isActive: false, deactivatedAt: timestamp}})
51 Account deactivated
52 User deactivated
53 200 OK "Account deactivated" (FR-065)
54 Show success
FR-061: View all user accounts
FR-062: Create new driver accounts
FR-063: Edit driver account information
FR-064: Activate driver accounts
FR-065: Deactivate driver accounts
FR-066: Prevent deletion of accounts with active duties
Automated Scheduled Reports
- Daily: Every day at 11:59 PM
- Weekly: Every Sunday at 11:59 PM
- Monthly: Last day of month at 11:59 PM
55 Cron job triggers (Daily)
56 generateDailyReport(date)
57 getDailyOperationalData(date)
58 { totalBusesOperated, totalTripsCompleted, totalDistance, routesCovered, dutiesCompleted, averageSpeed, incidents }
59 compileReportData()
60 generatePDF(reportData)
61 db.reports.insertOne({ reportType: "daily", date: date, data: reportData, pdfPath: path, generatedAt: timestamp })
62 {reportID}
63 Daily report generated (FR-072, FR-075)
64 pushNotification("Daily operational report ready")
65 Notification: "Daily report available" (FR-076)
66 Cron job triggers (Weekly)
67 generateWeeklyReport(weekStartDate, weekEndDate)
68 getWeeklyOperationalData(weekRange)
69 weeklyData
70 aggregateWeeklyStats()
71 generatePDF(weeklyReport)
72 db.reports.insertOne({ reportType: "weekly", startDate: weekStart, endDate: weekEnd, data: weeklyData, pdfPath: path })
73 {reportID}
74 Weekly report generated (FR-073, FR-075)
75 Cron job triggers (Monthly)
76 generateMonthlyReport(month, year)
77 getMonthlyOperationalData(month, year)
78 monthlyData
79 aggregateMonthlyStats()
80 generatePDF(monthlyReport)
81 db.reports.insertOne({ reportType: "monthly", month: month, year: year, data: monthlyData, pdfPath: path })
82 {reportID}

Figure 8: Authentication Sequence Diagram

4.4 Class Diagrams

4.4.1 Management Class Diagram

This diagram represents the main domain entities of the system such as User, Bus, Administrator, Driver, Duty,Passenger and Route. It shows attributes and relationships required to manage users and track buses effectively.



Abstract base class
Implements FR-001 to FR-009
Authentication & Authorization
User
-userId: String
-username: String
-password: String
-email: String
-phoneNumber: String
-createdAt: DateTime
-updatedAt: DateTime
-isActive: Boolean
+login(username: String, password: String): Boolean
+logout(): void
+resetPassword(email: String): Boolean
+viewProfile(): User
+updateProfile(data: Object): Boolean
1
0..1
Passenger
-passengerId: String
-preferredLanguage: String
+viewRealTimeBusLocation(): List<Bus>
+searchRouteByName(routeName: String): Route
+searchRouteByStop(stopName: String): List<Route>
+calculateETA(busId: String, stopId: String): Integer
+addFavoriteRoute(routeId: String): Boolean
+removeFavoriteRoute(routeId: String): Boolean
+viewFavoriteRoutes(): List<Route>
+viewBusDetails(busId: String): Bus
Implements FR-010 to FR-032
Bus Tracking, Route Search
ETA Calculation, Favorites
Driver
-driverId: String
-licenseNumber: String
-experience: Integer
+viewCurrentDuty(): Duty
+viewMonthlyDuties(): List<Duty>
+searchDutyByDate(date: Date): Duty
+startDuty(): Boolean
+shareLiveLocation(location: Location): Boolean
+viewOwnBusLocation(): Location
+viewRouteBuses(routeId: String): List<Bus>
+completeDuty(): Boolean
+viewDutyRecord(date: Date): List<Duty>
Implements FR-033 to FR-048
Duty Management
Live Location Sharing
Administrator
-adminId: String
-role: String
-permissions: List<String>
+viewAllBuses(): List<Bus>
+addBus(busData: Object): Boolean
+editBus(busId: String, busData: Object): Boolean
+removeBus(busId: String): Boolean
+createRoute(routeData: Object): Boolean
+modifyRoute(routeId: String, routeData: Object): Boolean
+deleteRoute(routeId: String): Boolean
+defineSchedule(routeId: String, scheduleData: Object): Boolean
+viewAllUsers(): List<User>
+createDriverAccount(driverData: Object): Boolean
+editDriverAccount(driverId: String, driverData: Object): Boolean
+activateDriver(driverId: String): Boolean
+deactivateDriver(driverId: String): Boolean
+assignDriverToBus(driverId: String, busId: String): Boolean
+createDutySchedule(dutyData: Object): Boolean
+modifyDutyAssignment(dutyId: String, dutyData: Object): Boolean
+generateReport(reportType: String, period: String): Report
+exportReport(reportId: String, format: String): File
Implements FR-049 to FR-077
Bus, Route, User Management
Reports Generation
Authentication
-token: String
-issuedAt: DateTime
-expiresAt: DateTime
-isValid: Boolean
+generateToken(userId: String): String
+validateToken(token: String): Boolean
+refreshToken(): String
+revokeToken(): Boolean
JWT-based authentication
24-hour token expiry
Implements FR-003
0..*
1
0..*
Duty
-dutyId: String
-date: Date
-startTime: Time
-endTime: Time
-status: String
-actualStartTime: DateTime
-actualEndTime: DateTime
-completionStatus: String
+startDuty(): Boolean
+completeDuty(): Boolean
+getDutyDetails(): Duty
+updateStatus(status: String): Boolean
+isCompleted(): Boolean
*
*
1
0..*
1
0..*
Report
-reportId: String
-reportType: String
-generatedAt: DateTime
-period: String
-data: Object
-format: String
+generateDailyReport(): Report
+generateWeeklyReport(): Report
+generateMonthlyReport(): Report
+exportToPDF(): File
+getData(): Object
0..*
1
1
Route
-routeId: String
-routeName: String
-startPoint: String
-endPoint: String
+getRouteDetails(): Route
Bus
-busId: String
-busNumber: String
-capacity: Integer
-status: String
+getBusDetails(): Bus
Links Driver, Bus, and Route
One Driver performs many Duties
Implements FR-033 to FR-041

Figure 9: Management Class Diagram

4.4.2 Information Providing Class Diagram

The information providing class diagram illustrates the core data structures used to support real-time tracking and information services within the system. It includes classes such as Tracking, Location, ETA, Route, Stop, and Schedule, which collectively manage GPS-based location updates, route definitions, stop sequencing, scheduling information, and estimated arrival time calculations. These classes are interconnected through associations and composition relationships to ensure accurate monitoring of bus movement and timely information delivery to administrative users, drivers, and passengers.

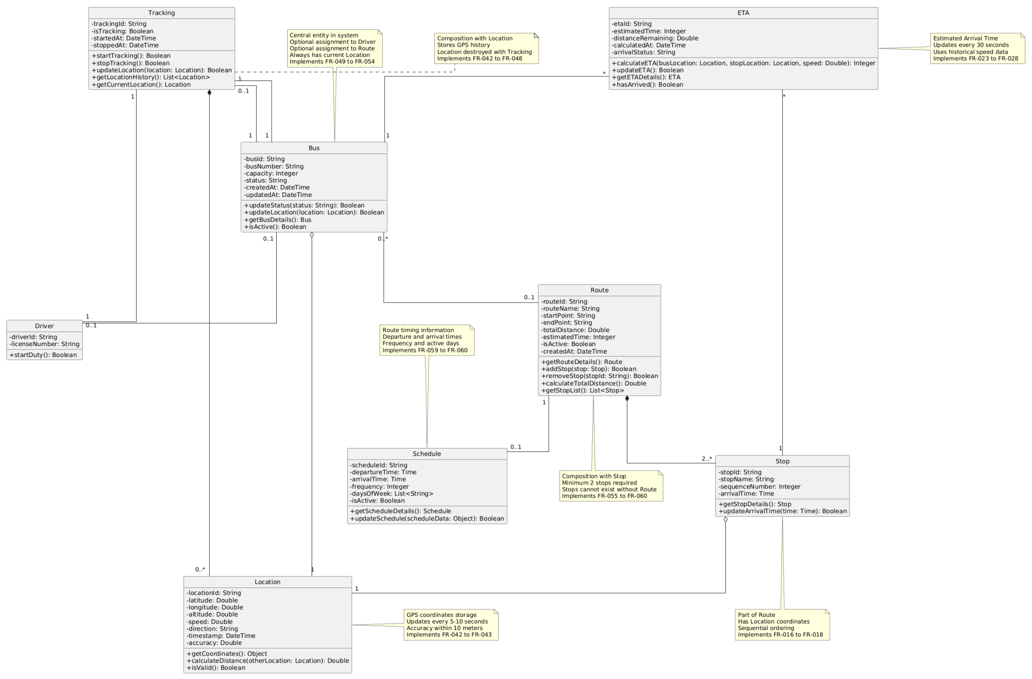


Figure 10: Information Providing Class Diagram

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.

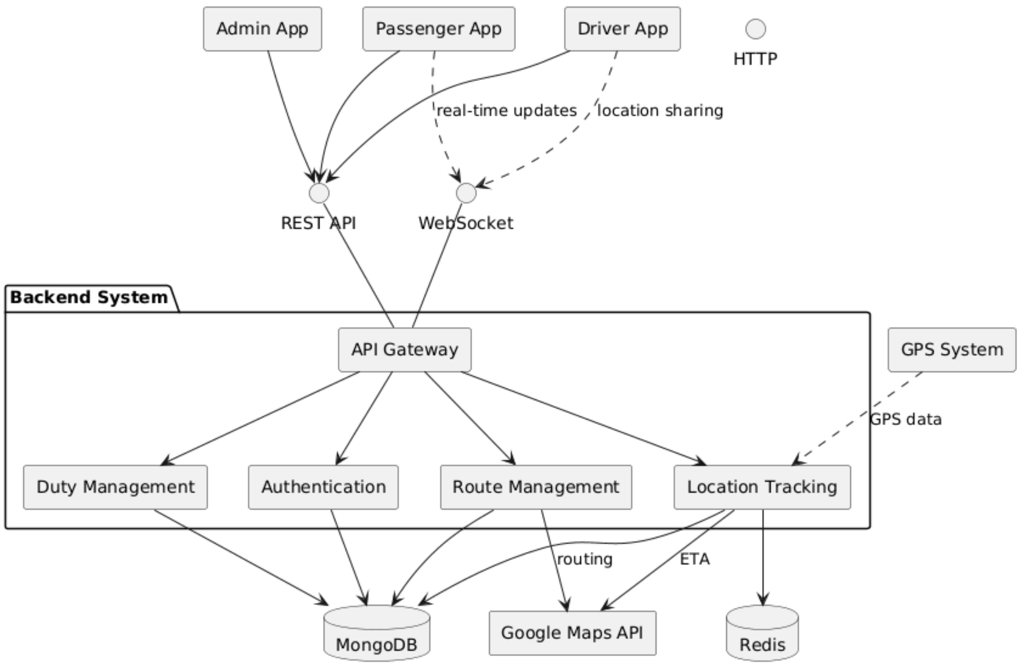


Figure 11: Client Component Diagram

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 backend 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

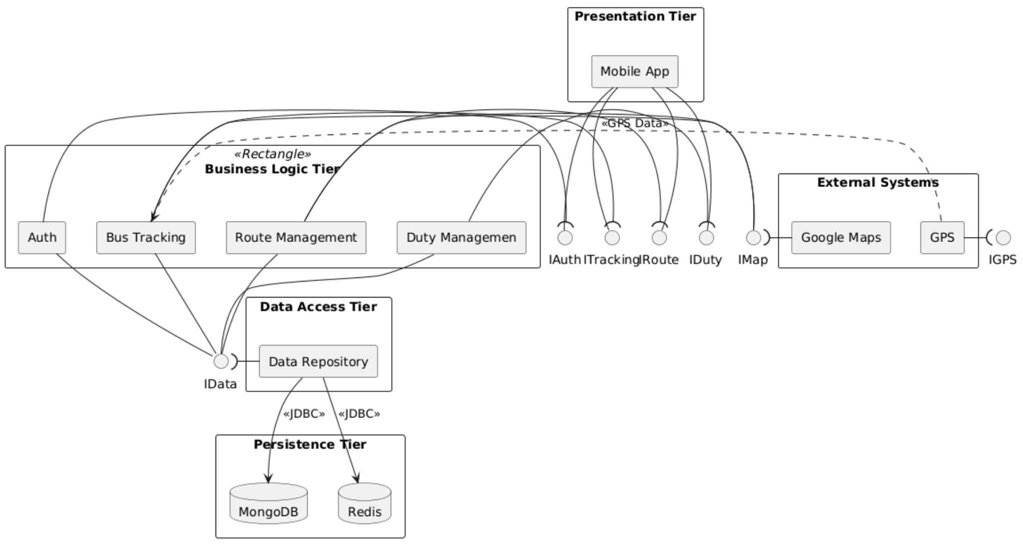


Figure 12: Server Component Diagram

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.

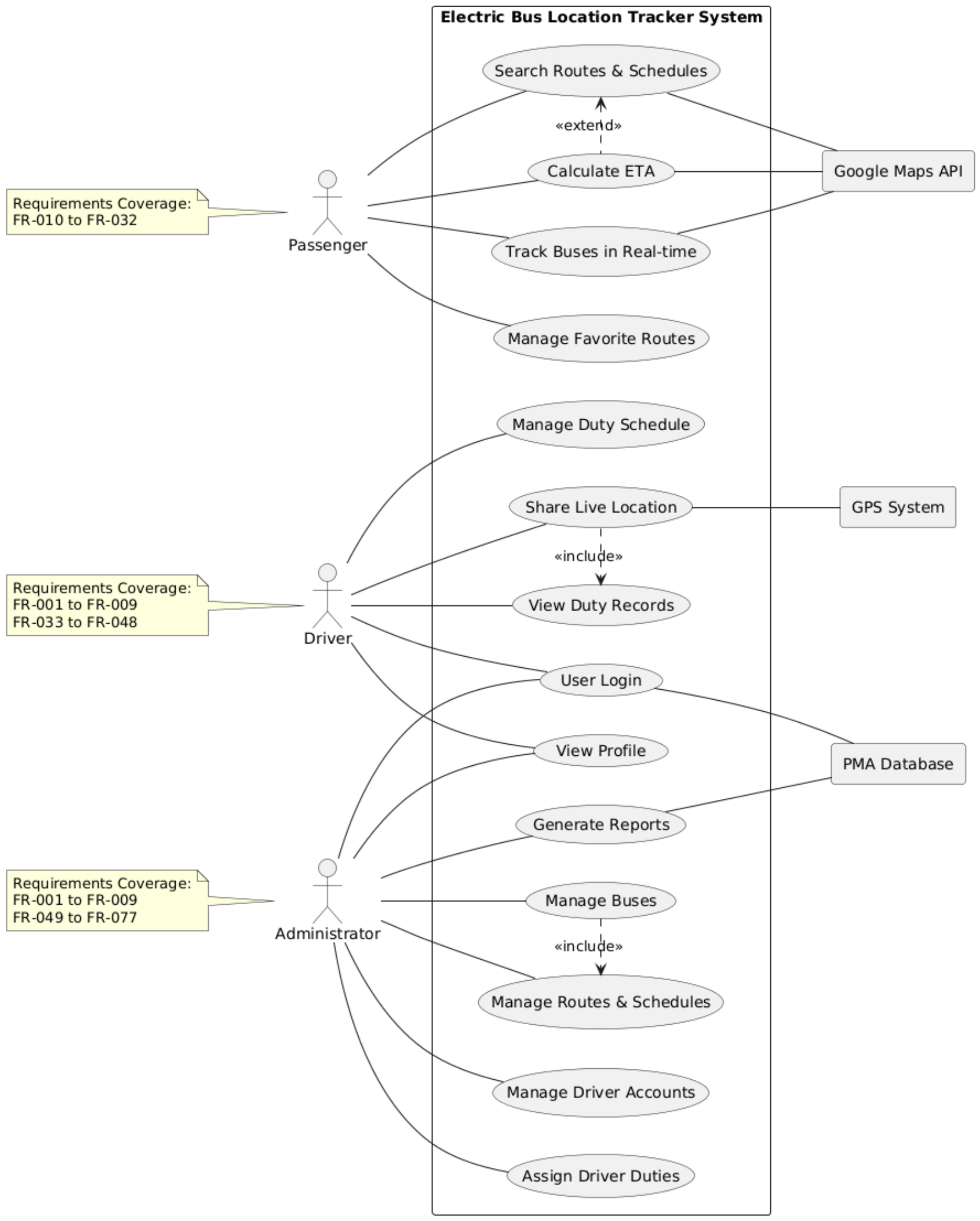


Figure 13: Use Case Diagram

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, Inforamation 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.

6 Github Link

Github link : `https://github.com/Mehrozalikhan01/Electric-buses-location-tracker.git`

7 Figma Link

Figma link : `https://www.figma.com/design/BoCShru8p7isvqPn427vHa/Untitled?node-id=13-6881&t=kgQ1WCa5wS2Oaihv-1`