Live Bus Tracking System for Sonargaon University: A Real-Time Solution for Students and Drivers

by

Zannatul Ferdous
ID: CSE2201025007

Zubayer Hossain
ID: CSE2201025075

Mustafizur Rahman
ID: CSE221025070

Md Nahid Islam
ID: CSE2201025036

Supervised by
Prof. Bulbul Ahamed
Head, Department of CSE

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SONARGAON UNIVERSITY (SU)

January 2026

APPROVAL

The project titled **"Live Bus Tracking System for Sonargaon University: A Real-Time Solution for Students and Drivers"** submitted by Zannatul Ferdous (CSE2201025007)**,** Zubayer Hossain (CSE2201025075), Mustafizur Rahman (CSE221025070), and Md Nahid Islam (CSE2201025036) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Bulbul Ahamed

Supervisor

Professor and Head,
Department of Computer Science and Engineering,
Sonargaon University(SU)

Examiner 1

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Prof. Bulbul Ahamed, Professor and Head, Department of Computer Science and Engineering, Sonargaon University**, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Bulbul Ahamed)
Supervisor

Signature

Zannatul Ferdous
ID: CSE2201025007

Zubayer Hossain
ID: CSE2201025075

Mustafizur Rahman
ID: CSE2201025070

Md Nahid Islam
ID: CSE2201025036

ABSTRACT

Modern technology has transformed how we manage daily activities, making mobile applications essential tools for improving efficiency and convenience. University transportation systems often suffer from a lack of real-time information, causing students to experience uncertainty about bus locations, arrival times, and routes. To solve these issues, Subahon has been developed as a real-time university bus tracking and student map application for Sonargaon University. The system enables students to view live bus positions, route paths, estimated arrival times (ETA), and driver details directly on an interactive map, while drivers can easily share their current location and update route status with a single tap. Built with Firebase Firestore, Kotlin, MVVM architecture, and Google Maps SDK, the application ensures fast and accurate synchronization of bus data between students and drivers. Subahon reduces unnecessary waiting time, improves communication, and enhances the overall transportation experience within the university. This paper presents the system's design, implementation, challenges, and impact, demonstrating how real-time tracking technology can significantly improve campus mobility and support smarter transportation management.

ACKNOWLEDGMENT

At the outset, we would like to express our sincere gratitude to the Almighty Allah for granting us the strength, patience, and wisdom to successfully complete this project within the scheduled time. Without His blessings, this accomplishment would not have been possible.

We feel deeply honored and fortunate to work under the kind guidance and supervision of Bulbul Ahamed, Professor and Head, Department of Computer Science and Engineering, Sonargaon University. His continuous support, valuable advice, constructive feedback, and constant encouragement played a crucial role in the successful completion of this project.

We are also sincerely thankful to all the respected teachers of the Department of Computer Science and Engineering, Sonargaon University, for sharing their knowledge, guidance, and inspiration throughout our academic journey, which greatly contributed to our learning and development.

Finally, we would like to express our deepest gratitude and heartfelt love to our parents for their unconditional support, encouragement, patience, and endless love. Their motivation and sacrifices inspired us to move forward with confidence and achieve our goals.

LIST OF ABBREVIATIONS

API	Application Programming Interface
DB	Extensible Markup Language
ETA	Estimated Time of Arrival
FCM	Firebase Cloud Messaging
Firestore	Firebase Firestore
GPS	Global Positioning System
MVVM	Model–View–ViewModel
SDK	Software Development Kit
UI	User Interface
UX	User Experience
XML	Extensible Markup Language

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO LIVE BUS TRACKING SYSTEM FOR SONARGAON UNIVERSITY

1.1 Introduction

Modern universities increasingly rely on smart mobility solutions to ensure efficient transportation for students and staff[1]. Every day, a large number of students travel to Sonargaon University from various parts of the city. However, due to the absence of an organized real-time bus monitoring system, students often face uncertainty regarding bus timings, estimated arrival time, and route details .

The Live Bus Tracking System for Sonargaon University is an Android-based mobile application that provides students with real-time bus locations, route paths, and driver details through Google Maps. Drivers can share their live location automatically using the app, which improves communication and reduces delays at bus stops [2]. Additionally, the system includes an admin panel that allows authorized administrators to approve or block user accounts and ensure that only verified students and drivers can access the application's features.

1.2 Problem Statement

The lack of a centralized real-time bus tracking system leads to the following issues :

- Students cannot know the real-time location of university buses[3].
- Estimated Time of Arrival (ETA) is unavailable or inaccurate.
- Miscommunication occurs frequently between students and drivers.
- Students often wait unnecessarily at bus stops, especially during peak hours.
- Drivers have no efficient system to share bus status or location.
- University authorities cannot monitor operations effectively.

Therefore, a real-time, mobile-based bus tracking system is required to streamline student transportation, reduce waiting times, and improve overall communication.

1.3 Objectives

The objectives of the project are:

- To provide real-time bus location tracking for students.
- To display route information, bus direction, driver details, and ETA[4].
- To enable bus drivers to share live GPS locations easily.
- To ensure instant communication between students and drivers.
- To reduce waiting time and improve transportation efficiency.
- To assist university authorities in monitoring and managing bus operations.
- To develop a lightweight, secure, and user-friendly Android application.

1.4 Scope of the Project

The scope of this project includes:

- Development of an Android application for students and drivers.
- Integration of Google Maps SDK for live route visualization.
- Implementation of Firebase Firestore for real-time location updates.
- Role-based user interface for students and drivers[5].
- Displaying active buses, route polylines, and driver information.
- Instant updates using Firestore listeners.
- Quick contact options and bus list functionality.

Out of Scope:

- Web-based admin panel (planned for future enhancement).
- Offline map navigation and caching.
- AI-based ETA prediction (future work).

1.5 Structure of the Report

- **Chapter 2:** Literature Review – Discusses existing bus tracking systems and related work.
- **Chapter 3:**: System Analysis and Design – Explores system requirements, use case analysis, and architectural design.
- **Chapter 4:** Implementation and Testing – Details technology stack, coding process, and evaluation through testing procedures.
- **Chapter 5 :** Results, Discussion, Conclusion and Future Work.

CHAPTER 2

LITERATURE REVIEW ON REAL-TIME BUS TRACKING SYSTEMS

2.1 Introduction

Real-time transportation tracking has become an essential component of modern smart mobility systems. Universities, public transport authorities, and private transport services widely use GPS-based tracking solutions to ensure an efficient and reliable commuting experience for passengers.

The purpose of this chapter is to review existing research, applications, and technologies related to real-time bus tracking, student transportation systems, and location-based mobile applications [6]. This review helps identify limitations in current solutions and establish the technological foundation for the Live Bus Tracking System for Sonargaon University.

2.2 Existing Bus Tracking Systems

Several bus tracking systems exist globally, implemented either by public transportation authorities or private companies[7]. Most of them rely on GPS, digital map services such as Google Maps, and cloud-based databases to collect and visualize real-time bus locations . Some key examples are outlined below.

2.2.1 Google Maps Live Transit Tracking

Google Maps offers real-time transit tracking in many countries. It provides live bus locations, estimated arrival times (ETA), and route visualizations for public transportation services[8]. However, it does not support university-specific custom routes, institution-managed fleets, or custom role-based access (such as student, driver, and admin). Moreover, the tracking and schedule data are controlled by public agencies, not by individual universities.

2.2.2 Dhaka City Bus Real-Time Apps

A few private applications and experimental services have attempted to provide real-time tracking for Dhaka city buses. Although they demonstrate the potential of GPS-based bus tracking in Bangladesh, they suffer from several limitations, such as:

- Lack of stable and continuous GPS updates
- Non-standardized or poorly documented bus routes
- Frequent delays or inconsistencies in data synchronization
- No dedicated support for university-operated buses or student-focused features

These limitations highlight the need for a more specialized, reliable, and university-centric real-time bus tracking solution, such as the proposed system for Sonargaon University.

2.2.3 University Bus Tracking Apps (International)

Many universities abroad use dedicated bus tracking platforms such as TransLoc, NextBus, and DoubleMap[9]. These services typically provide real-time GPS tracking, route prediction, stop-based estimated time of arrival (ETA), and web or mobile interfaces for students and staff.

However, these platforms are proprietary and usually require paid subscriptions, dedicated infrastructure, and technical integration support. As a result, they are often too costly and complex for smaller institutions or universities in developing countries like Bangladesh. Furthermore, they are not specifically tailored to the local transport context of Sonargaon University , which justifies the need for a customized, in-house solution.

2.3 Research on Real-Time Tracking Technologies

Several studies highlight the importance and feasibility of real-time location tracking in transportation systems. This section briefly reviews key enabling technologies such as GPS, cloud databases, and mobile applications[10].

2.3.1 GPS-Based Tracking

Research consistently confirms the Global Positioning System (GPS) as the most accurate and widely used tracking method for transportation systems. GPS enables:

- Continuous real-time location monitoring of moving vehicles
- Reliable measurement of distance, speed, and movement patterns
- Acceptable accuracy in urban environments when combined with map services
- Relatively low battery consumption on modern mobile devices when properly configured

Because of these advantages, GPS is the primary technology used for real-time bus tracking in this project .

2.3.2 Cloud Databases for Real-Time Sync

Cloud-based databases such as Firebase Firestore are widely used to support real-time applications[11]. Studies and practical implementations show that these technologies offer:

- Live synchronization of data across multiple clients
- Event-based updatesEvent-based updates using listeners or subscriptions
- Low-latency communication between mobile devices and the cloud
- Automatic scalability without dedicated server management

These features make Firebase Firestore highly suitable for a real-time university bus tracking system where student and driver devices must stay continuously updated.

2.3.3 Mobile Application-Based Navigation

Research on mobile navigation and location-based services indicates that smartphone applications provide:

- Better user accessibility, as most students already own Android smartphones
- Lower infrastructure cost, since no dedicated hardware devices are required on the user side
- High adoption rates among students, particularly when the interface is simple and intuitive[12]

These findings support the decision to implement the proposed system as an Android mobile application rather than as a web-only or hardware-based solution.

2.4 Limitations of Existing Systems

Despite the availability of various tracking solutions, several common limitations are observed, especially in the context of university bus services [13]:

- No dedicated systems for individual university buses; most solutions target city-wide public transport.
- Lack of truly real-time updates or slow refresh rates, leading to outdated bus positions.
- Absence of a proper role-based system that distinguishes between students, drivers, and administrators.
- Poor or incomplete route visualization, making it difficult for users to understand the exact bus path.
- Limited or no in-app communication features for contacting drivers or viewing their information.
- High implementation and subscription costs that are unsuitable for many local institutions in Bangladesh.

These limitations highlight the need for a cost-effective, real-time, and university-specific bus tracking solution with proper role-based access control.

2.5 SummaryExisting Bus Tracking Systems

This chapter reviewed existing tracking systems, related research, and modern technologies relevant to real-time bus tracking[14]. While several real-time bus tracking systems and platforms exist globally, none of them are specifically tailored to the operational context and requirements of Sonargaon University and its transportation network.The findings from this review justify the development of a dedicated, scalable, and mobile-based Live Bus Tracking System for Sonargaon University that leverages Google Maps, GPS, Firebase Firestore, and role-based access for students, drivers, and administrators.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN – EXPLORES SYSTEM REQUIREMENTS, USE CASE ANALYSIS, AND ARCHITECTURAL DESIGN.

3.1 Introduction

This chapter presents the system analysis and design of the Live University Bus Tracking and Student Map Application developed for Sonargaon University[15]. The main purpose of this chapter is to analyze user requirements, identify system functionalities, and design an efficient architecture for real-time bus tracking.

The chapter covers functional and non-functional requirements, use case analysis, system architecture, data flow, and database design to ensure that the system meets user needs and performs reliably. Special focus is given to role-based access for students, drivers, and administrators, and to real-time communication using GPS and cloud technologies[16].

3.2 System Analysis

System analysis focuses on understanding how the proposed system will operate and how different users will interact with it. The system is designed to solve common transportation problems such as uncertainty of bus arrival time, lack of real-time information, and poor communication between students and drivers[17].

The application provides three main user roles: Student, Driver, and Admin. Each role has different functionalities based on access permissions. Students mainly consume real-time information, drivers provide live location data, and admins manage user accounts and access control.

3.2.1 Stakeholders

- Students
  Students use the application to view real-time bus locations, routes, estimated time of arrival (ETA), distance from their current position, and driver contact information.
- Bus Drivers
  Drivers use the application to start and stop live location sharing, update the bus route and operational status, and ensure that students receive accurate real-time tracking information.
- University Authority / Admin Users
  Authorized members of the university (admins) use the admin panel to manage user

accounts. They can approve or reject newly registered users, assign roles (student or driver), and block or unblock accounts when necessary. In future, admins may also monitor detailed transportation operations and generate reports.

3.3 Functional Requirements

Functional requirements define what the system is expected to do for each type of user[16].

3.3.1 Student Requirements

- Students shall be able to view active university buses on Google Maps.
- Students shall receive real-time bus location updates.
- Students shall be able to view route information and bus direction .
- Students shall be able to see estimated arrival time (ETA) and distance from their current location.
- Students shall be able to call the driver directly from the application .
- Students shall be able to view a list of all active buses.
- Students shall only be able to use the system features after their account has been approved by an admin.

3.3.2 Driver Requirements

- Drivers shall be able to log in securely.
- Drivers shall be able to start and stop live bus tracking.
- Drivers shall be able to update bus route and operational status.
- The system shall automatically upload GPS location and heading at regular intervals.
- Drivers shall only be able to share location and update bus information after their account has been approved by an admin.

3.3.3 Admin Requirements

- Admins shall be able to log in securely with admin credentials.
- Admins shall be able to view a list of all registered users.
- Admins shall be able to view a list of users with “pending” status.
- Admins shall be able to approve or reject new user registrations.
- Admins shall be able to assign roles to users (student or driver).
- Admins shall be able to block or unblock user accounts.
- Admins shall be able to ensure that only approved and active users can access the core features of the application.

3.3.4 System Requirements

- The system shall support real-time data synchronization.
- The system shall manage multiple buses simultaneously.
- The system shall dynamically switch UI based on user role.
- The system shall store and retrieve data using Firebase Firestore.

3.4 Non-Functional Requirements

- Performance:
  The system shall provide near real-time location updates with minimal delay, under normal network conditions.
- Usability:
  The user interface shall be simple, responsive, and easy to use for students, drivers, and admins[18].
- Reliability:
  The system shall function continuously without data loss and should gracefully handle temporary network disconnections.
- Security:
  User authentication and authorization shall be handled securely using Firebase Authentication and role-based access control.
- Scalability:
  The system shall support future expansion in terms of number of users, number of buses, and additional features without major architectural changes.
- Maintainability:
  The system shall follow the MVVM architecture pattern with clear separation of concerns, making the codebase easier to maintain, test, and extend[17].

3.5 Use Case Analysis

Use case analysis describes how different users interact with the system and what services the system provides to each user[16]. It helps to understand the functional behavior of the system from the user's perspective. In this system, three primary actors interact with the application: Student, Driver, and Admin.

3.5.1 Use Case Diagram

Use Case Diagram showing Student and Driver actors

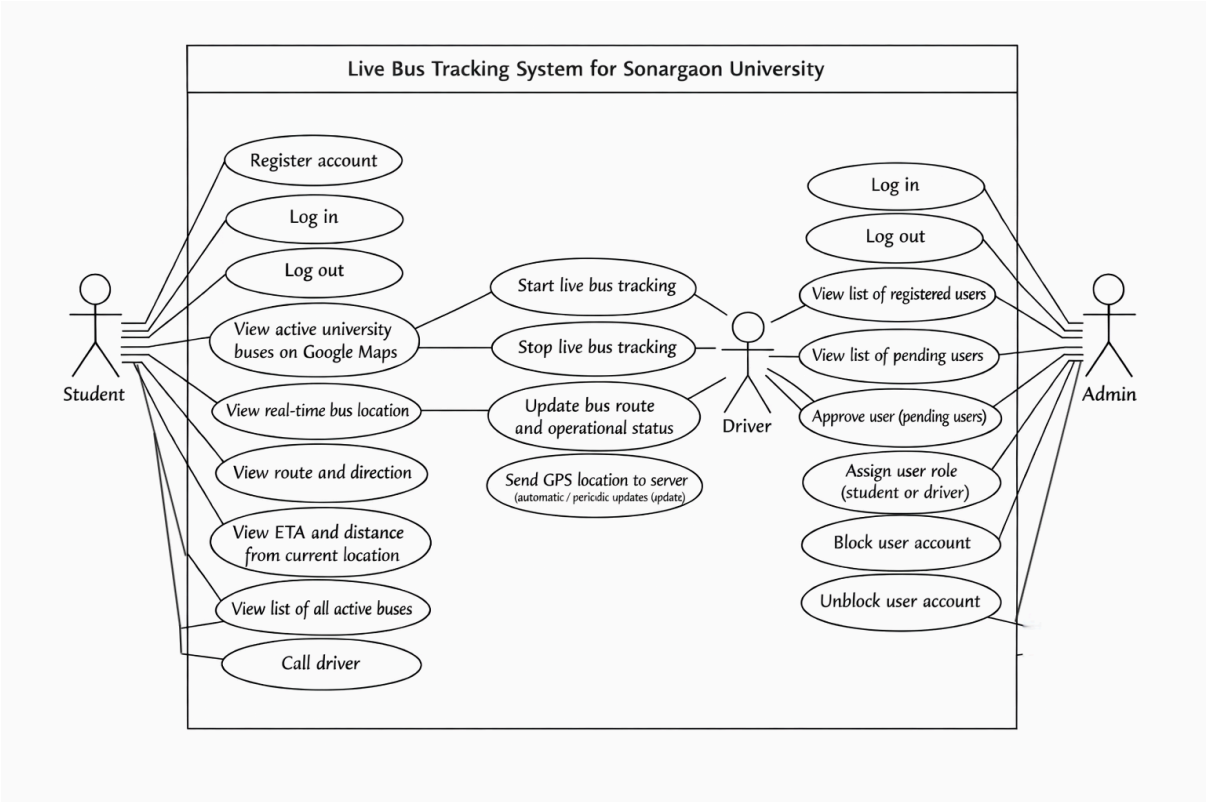


Figure 3.1: Use Case Diagram of the Live Bus Tracking System

3.5.2 Use Case Description – Student

A student is an end user of the system who uses the application to track university buses in real time. The student can view bus locations, routes, estimated time of arrival (ETA), and driver contact information.

Use Case Name: View Live Bus Location
Actor: Student

Preconditions:

- The student has a valid, admin-approved account.
- The student is logged in to the application.
- The device has an active internet connection and location service (GPS) enabled.

Main Flow:

1. The student opens the application and navigates to the bus tracking screen.

2. The system initializes Google Maps and centers the map based on the student's current location or a default campus location.
3. The system requests the list of active buses and their latest GPS coordinates from Firebase Firestore[13].
4. The system displays live bus locations on the map using markers, along with basic information such as route name and bus/driver identifier.
5. As Firestore data changes, the system continuously updates the bus markers in real time.

Postcondition:

- The student can view the current live positions of all active university buses on the map.

3.5.3 Use Case Description – Driver

A driver is responsible for sharing the live location of the bus with the system[2]. The driver controls the tracking status and updates the route and bus availability so that students can see accurate information.

Use Case Name: Start Bus Tracking
Actor: Driver

Preconditions:

- The driver has a valid, admin-approved account.
- The driver is authenticated and logged in to the application[14].
- The driver has selected/been assigned a bus and route.
- The device has GPS and an active internet connection enabled.

Main Flow:

1. The driver opens the application and navigates to the tracking screen.
2. The driver taps the "Start live bus tracking" button.
3. The system activates GPS and starts reading the current location of the device.
4. The system begins sending the bus's GPS coordinates, speed, and heading direction to Firebase Firestore at regular intervals.
5. The system updates the bus status to "Active" so that it is shown to students as an available bus on the map.

Postcondition:

- The bus becomes visible to students as an active bus, and its live location is updated on their map in real time.

3.6 System Design

3.6.1 Architectural Design

The proposed system is designed using the Model–View–ViewModel (MVVM) architectural pattern on top of a client–cloud infrastructure[16]. This approach clearly separates the user interface from the business logic and data layer, which improves maintainability, testability, and scalability of the application.

- Model Layer
  The Model layer represents the application data and the logic for accessing it. In this project, the Model includes:

  - Firebase Firestore collections such as users, routes, and bus_service, which store user profiles, bus routes, and live location information.

  - Repository classes that communicate with Firestore, Firebase Authentication, and the FusedLocationProviderClient. These repositories hide low-level implementation details from the rest of the application.

- View Layer
  The View layer consists of Android Activities and Fragments that display data on the screen and handle user interactions[18]. Examples include:

  - The Student Map Fragment, which shows Google Maps, active bus markers, and the list of available buses.

  - The Driver Tracking Screen, which provides Start/Stop tracking controls.

  - The Admin User Management Screen, where admins approve, block, or assign roles to users.
    Views do not contain business logic; they simply observe the data exposed by the ViewModel and update the UI accordingly[17].

- ViewModel Layer
  The ViewModel layer acts as an intermediary between the View and the Model. ViewModels[16]:

  - Contain the core business logic for loading and processing data.

  - Maintain observable data (e.g., LiveData lists of buses, routes, and user status).

- Attach real-time Firestore snapshot listeners to receive instant updates about active drivers and route changes.
  For example, StudentMapViewModel listens to driver documents with status "started", converts them into BusDisplay objects, and updates the map and bus list in real time. Similarly, a dedicated Admin ViewModel manages user approval and role changes.

By using MVVM, the same backend logic can support different roles (student, driver, admin) with minimal code duplication, while any future changes in the database or UI can be handled with limited impact on other layers.

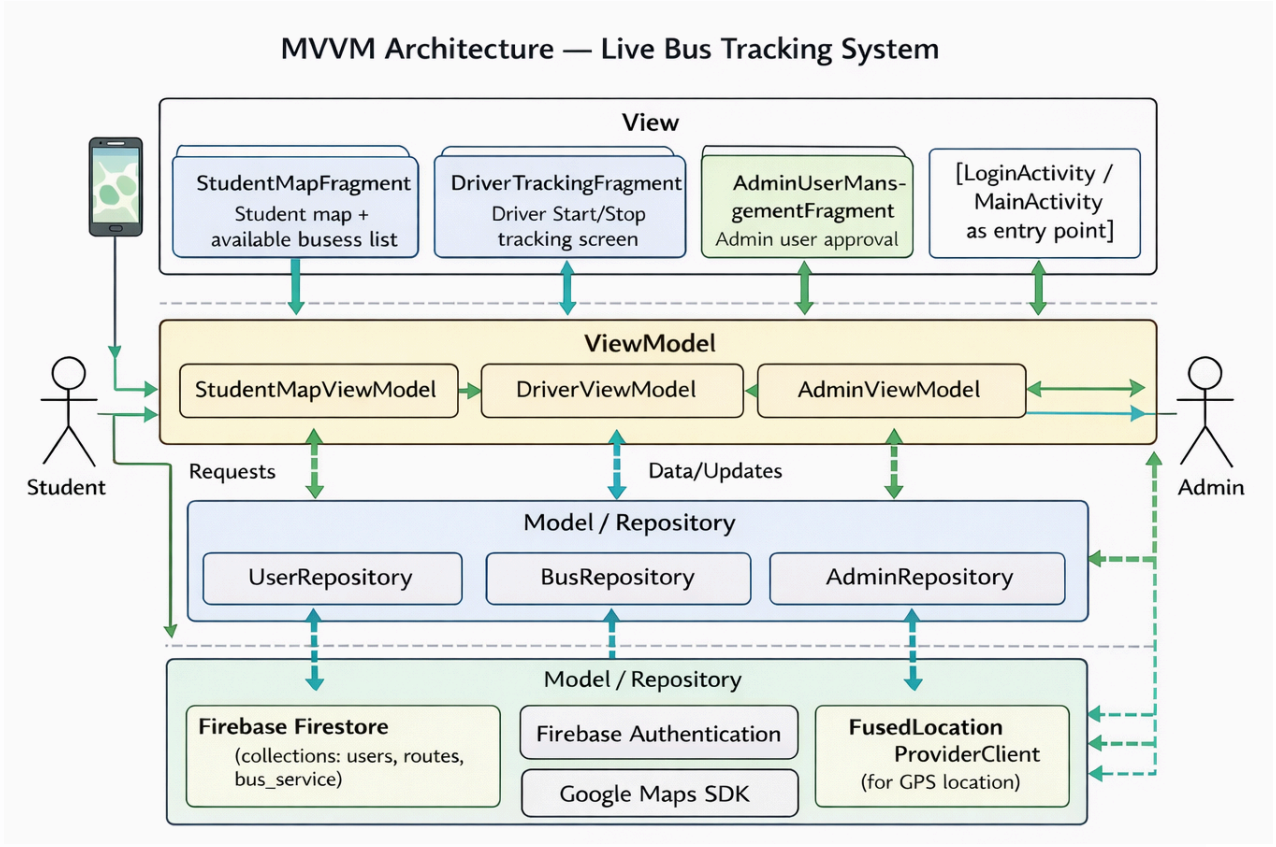


Figure 3.2: MVVM Architecture of the Live Bus Tracking System for Sonargaon University

3.6.2 Component Design

The proposed system is decomposed into several major software components. Each component has a specific responsibility and interacts with others through clearly defined interfaces[18].

- StudentMapFragment
    - Initializes the Google Map and user interface for students.
    - Displays the map, active bus markers, route polylines, and the list of available buses.
    - Observes LiveData from StudentMapViewModel and updates the UI accordingly.
- DriverTrackingFragment
    - Provides the driver interface for starting and stopping live bus tracking.
    - Shows the current tracking status and route information assigned to the driver.
    - Calls the corresponding methods in DriverViewModel.
- AdminUserManagementFragment
    - Provides the admin interface for viewing registered and pending users.
    - Allows admins to approve, block, or assign roles (student/driver) to users.
    - Communicates with AdminViewModel for all admin operations.

- StudentMapViewModel
    - Manages Firestore snapshot listeners for active drivers (role = "driver", status = "started") [15].
    - Loads route polylines from the routes collection .
    - Exposes observable LiveData lists of buses and routes to StudentMapFragment
- DriverViewModel
    - Controls the start and stop of driver tracking.
    - Periodically sends the driver's GPS location, heading, and status to Firestore.
    - Updates the corresponding document in the users collection.
- AdminViewModel
    - Loads pending and approved user accounts from Firestore.
    - Implements business logic for approving, blocking, and updating user roles.
- BusListAdapter
    - A RecyclerView adapter used in the student UI to display the list of active buses under the map[17].
    - Binds each BusDisplay item (driver name, route name, heading, phone) to the list row.
    - Provides a callback to open the phone dialer when the call icon is tapped.

- UserRepository, BusRepository, RouteRepository
    - Encapsulate all Firestore read/write operations related to users, buses, and routes.
    - Provide a clean API for the ViewModels and hide low-level database details.
- Firebase Firestore
    - Cloud NoSQL database that stores user profiles, roles, approval status, live locations, and route definitions (polylines)[13].
    - Supports real-time listeners for instant update of active buses on student devices .
- Firebase Authentication
    - Manages user registration, login, and identity.
    - Provides the unique user ID used as the document ID in the `users` collection.
- FusedLocationProviderClient
    - Android location service used to obtain accurate GPS coordinates for drivers (and optionally students).
- Google Maps SDK for Android
    - Renders the map, bus markers, and route polylines in `StudentMapFragment`.
    - Supports marker click events to show bus details[11].

3.7 Data Flow Description

The live bus tracking feature is based on continuous data exchange between the driver app, Firebase Firestore, and the student app[15]. The main data flow is as follows:

1. Driver starts live tracking
    - The driver logs in and taps the “Start live bus tracking” button.
    - The application sets the driver’s document in the `users` collection to `status = "started"` and associates the corresponding `routeId` .
2. GPS location capture
    - The driver’s device uses the FusedLocationProviderClient to obtain high-accuracy GPS coordinates at regular intervals.
    - Each location result contains latitude, longitude, and (optionally) speed and heading.
3. Uploading data to Firebase Firestore
    - For every location update, the app writes the new `location` (GeoPoint), `heading`, `status`, and `timestamp` to the driver’s document in the `users` collection.
    - This turns the driver’s record into a live representation of an active bus
4. Real-time synchronization via Firestore listeners
    - On the student side, `StudentMapViewModel` registers a real-time snapshot listener on `users` with filters `role = "driver"` and `status = "started"`.
    - Whenever any driver’s document changes (new location, status change, etc.), Firestore immediately pushes an updated snapshot to the ViewModel.
5. Updating the student map

- The ViewModel converts the snapshot into a list of BusDisplay objects and corresponding route polylines (loaded from the routes collection).
- These lists are posted to LiveData, which the StudentMapFragment observes[17].
- The fragment then updates or animates the existing bus markers on Google Maps, removes markers for inactive buses, and refreshes the "Available buses" list.

Through this data flow, any change in the driver's GPS location is reflected on the students' devices almost instantly, providing a real-time bus tracking experience[15].

3.8 Data Flow Diagram of the System

The data flow diagram represents how information moves between external entities (Student, Driver, Admin), the mobile application, and the Firebase backend. It focuses on the main processes: user authentication, user management, live bus tracking, and viewing bus information[12].

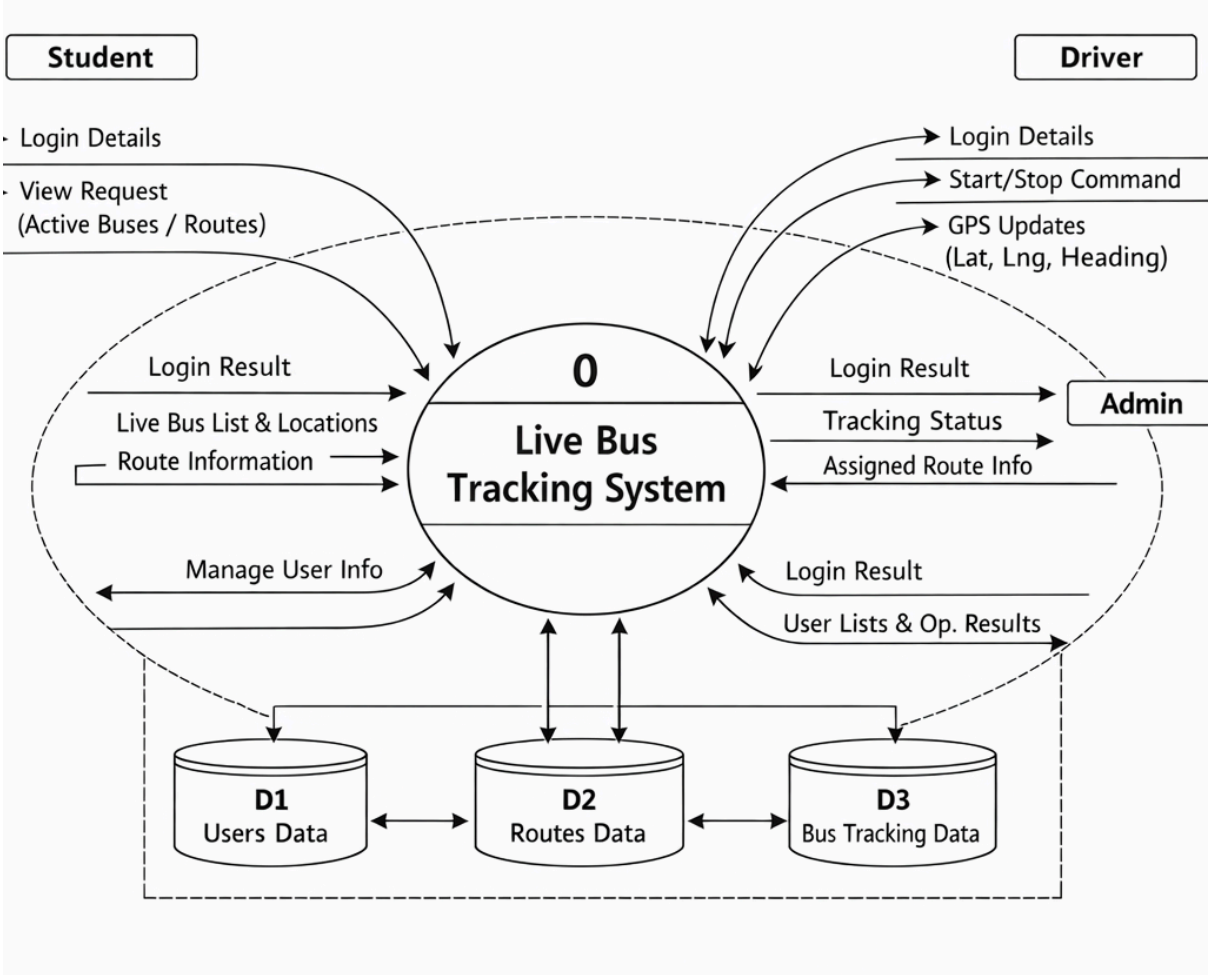


Figure 3.3: Data Flow Diagram of the Live Bus Tracking System.

3.8 Database Design

The system uses Firebase Firestore, a cloud-based NoSQL document database[15].
Data is organized into collections and documents instead of traditional relational tables.
Each document has a unique document ID (often the same as the user's Firebase UID).

The main collections used in this project are:

- users – stores all user accounts (students, drivers, admins) and, for drivers, the latest tracking data
- buses (or bus_service) – stores static information about each physical bus (optional, depending on implementation)
- routes – stores predefined route information and polyline points for drawing paths on Google Maps

3.8.1 users Collection

Each document in the users collection represents one user.

Key fields (besides the document ID itself):

- email : String – user's email address
- name : String – full name of the user
- phone : String – contact number (for drivers, used by students to call the driver)
- role : String – "student", "driver", or "admin"
- admin : Boolean – true for admin accounts, false otherwise
- studentId : String – university student ID / registration number (for verification)
- routeId : String – ID of the assigned or preferred route (e.g., "bus1", "bus2")
- location : GeoPoint – latest GPS position of the user's device (used mainly for drivers)
- heading : String – direction text such as "Up" or "Down"
- status : String – e.g., "pending", "approved", "blocked", "started", "stopped"
- timestamp : Timestamp – time of the last status/location update

3.8.2 buses (or bus_service) Collection *(optional / static bus info)*

If you are keeping a separate collection for bus metadata, describe it like this:
Each document in the buses (or bus_service) collection represents one physical bus.
Typical fields:

- busId : String – unique ID or bus code (document ID can also serve as busId)
- busName : String – display name of the bus (e.g., "Abdul Korim – Bus 1")
- routeId : String – default route assigned to this bus
- driverUserId : String – reference to the driver's document in users
- status : String – e.g., "active", "inactive", "under_maintenance"

If you are not actually using a separate buses collection and keeping everything in users for drivers, then either:

- remove this subsection from the book, or
- clearly write that it is a possible extension / logical view, but current prototype stores bus status with driver accounts[13]

3.8.3 routes Collection

Each document in the routes collection represents a predefined bus route[13].

Main fields:

- routeId : String – logical ID of the route (often equal to the document ID, e.g., "bus1", "bus2")
- routeName : String – human-readable name of the route
  - e.g., *"Green Road to Technical"*, *"Green Road to Mugrapara"*
- polylinePoints : Array of Maps – ordered list of points describing the path on the map
  - each element: { "lat": Double, "lng": Double }
- status : String – e.g., "Available", "Not Available" (whether this route is currently in use)

These polyline points are loaded by the Android app and used to draw the blue route line on Google Maps.

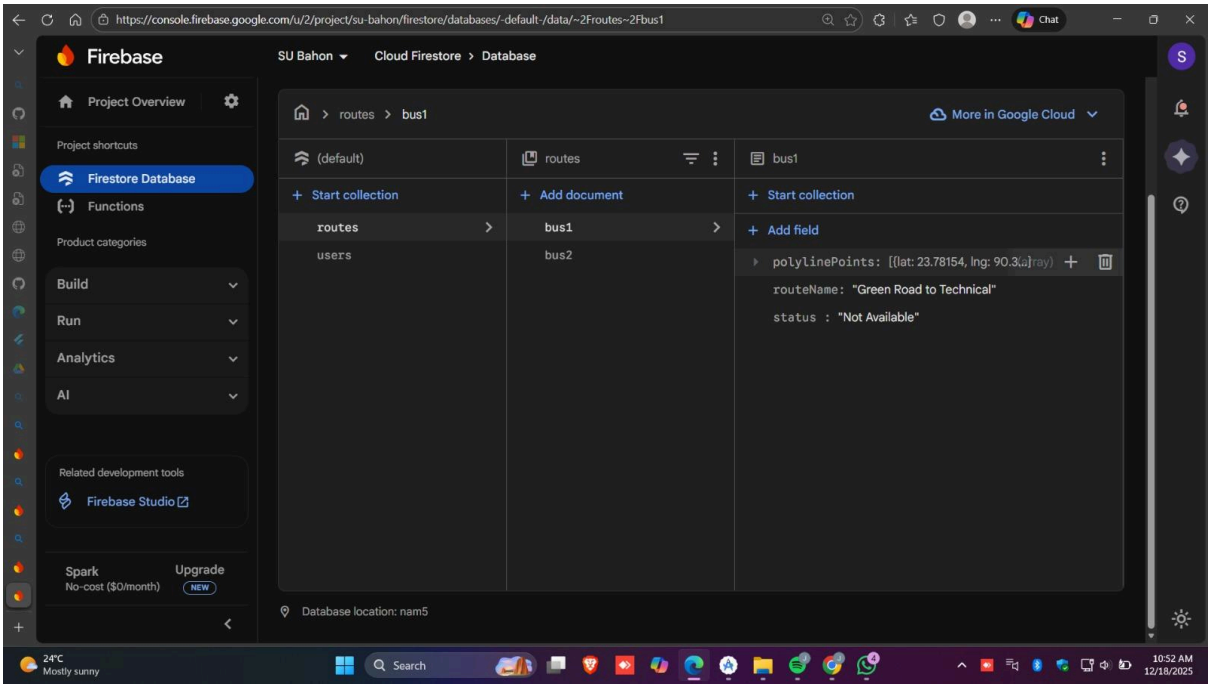


Figure 3.4: Firestore Routes Collection Showing Route Polylines

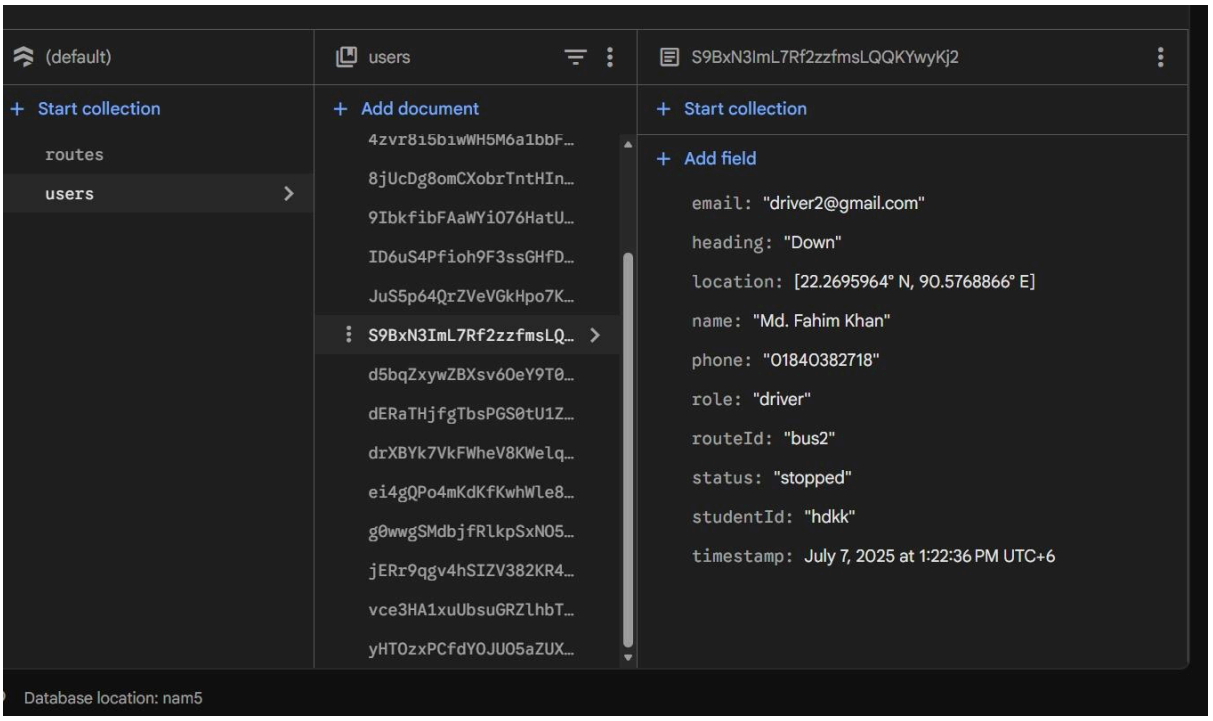


Figure 3.5: Firestore Users Collection Storing Driver and Tracking Information

3.9 Summary

This chapter presented a detailed system analysis and design of the Live Bus Tracking System and Student Map Application for Sonargaon University[18]. Functional and non-functional requirements were identified for all three user roles—student, driver, and admin—and expressed through use case descriptions and data flow diagrams (Level-0 and Level-1).

The overall architecture was designed using the MVVM pattern, supported by Firebase Firestore, Firebase Authentication, FusedLocationProviderClient, and Google Maps SDK. Component diagrams and database design (Firestore collections for users, routes, and related data) were described to clarify how the system is decomposed into independent modules. The proposed design supports real-time location updates, role-based access control, scalability, and maintainability, providing a solid foundation for an effective and extensible university transportation management system.

CHAPTER 4

SYSTEM ANALYSIS AND DESIGN – EXPLORES SYSTEM REQUIREMENTS, USE CASE ANALYSIS, AND ARCHITECTURAL DESIGN.

4.1 Introduction

This chapter describes the practical implementation and testing of the Live Bus Tracking System and Student Map Application for Sonargaon University. While Chapter 3 focused on system analysis and design, this chapter explains:

- the technology stack used to build the system,
- how the major modules were implemented in code, and
- how the application was evaluated through different testing procedures.

The system is implemented as an Android application using the MVVM architecture, Firebase Firestore and Firebase Authentication as the backend, FusedLocationProviderClient for GPS location, and Google Maps SDK for visualization of buses and routes[17].

4.2 Technology Stack and Tools

The main technologies and tools used in the implementation are:

- Programming Language
    - Kotlin for Android application development.
- Frameworks and Libraries
    - Android Jetpack: ViewModel, LiveData, RecyclerView, Fragment, etc.
    - Google Maps SDK for Android for map rendering and markers.
    - FusedLocationProviderClient (Google Play Services Location API) for GPS tracking.
    - Lottie animations for loading and transition effects (optional).
- Backend and Cloud Services
    - Firebase Firestore – NoSQL cloud database for storing users, routes, and live bus locations[7].
    - Firebase Authentication – For user registration, login, and identity management.
- Architecture Pattern
    - Model–View–ViewModel (MVVM) with a Repository layer.
- Development Environment
    - Android Studio Hedgehog / Jellyfish.

- Gradle as the build system.
- Minimum SDK level and target SDK level according to the latest Android version supported by the university lab.

The Android project is organized into several logical packages according to the MVVM architecture[16]:

- ui – Contains Activities and Fragments for all user interfaces (Student, Driver, Admin).
- viewmodel – Contains ViewModel classes (e.g., StudentMapViewModel, DriverViewModel, AdminViewModel).
- data / model – Contains data classes such as BusDisplay, User, and repositories for Firestore access.
- utils – Helper classes or objects for Firebase initialization, permission handling, and common utilities.

4.3 Firebase Project Configuration

Firebase was integrated into the application using the following steps:

1. Creating a Firebase project (e.g., "SU Bahon") in the Firebase console.
2. Registering the Android app with its package name and downloading the google-services.json file.
3. Enabling Cloud Firestore and Firebase Authentication.
4. Adding Gradle dependencies for Firestore, Authentication, Google Maps, and Location Services.
5. Implementing a small utility object (e.g., FirebaseUtil) to provide singleton access to FirebaseAuth and FirebaseFirestore.

4.4 Module Implementation

4.4.1 User Authentication and Role Management

User accounts are created and authenticated using Firebase Authentication (email/password)[14]. For every successfully registered user, a corresponding document is created in the users collection in Firestore. This document stores information such as:

- name, email, phone, studentId,
- role ("student", "driver", "admin"), and
- status ("pending", "approved", "blocked", "started", "stopped").

After a successful login, the application reads users/<uid> and loads the role and status fields. A helper function (e.g., checkUserRole()) decides which UI should be shown[16]:

- Student role → student map UI is displayed .
- Driver role → driver tracking UI is displayed .

- Admin role → admin panel is displayed .

Only users with status = "approved" are allowed to use the main features. Admin users can manage other users through the admin panel by changing their role and status .

The role-based user interface switching between student and driver views is implemented in the checkUserRole(), setupStudentUI() and setupDriverUI() functions, as shown in Fig 4.1.

```
private suspend fun checkUserRole() {
        val uid = FirebaseUtil.auth.currentUser?.uid ?: return

        val role = FirebaseUtil.firestore
            .collection("users")
            .document(uid)
            .get()
            .await()
            .getString("role")

        if (_binding == null) return

        if (role == "driver") setupDriverUI() else setupStudentUI()
    }

    private fun setupStudentUI() {
        if (_binding == null) return

        binding.studentBottomView.visibility = View.VISIBLE
        binding.driverBottomView.visibility = View.GONE

        adapter = BusListAdapter(emptyList()) { phone ->
            startActivity(
                Intent(
                    Intent.ACTION_DIAL,
                    Uri.parse("tel:${phone ?: ""}")
                )
            )
        }

        binding.rvRoutes.apply {
            layoutManager = LinearLayoutManager(requireContext())
            adapter = this@StudentMapFragment.adapter
            visibility = View.VISIBLE
        }
    }

    private fun setupDriverUI() {
        if (_binding == null) return

        binding.driverBottomView.visibility = View.VISIBLE
        binding.studentBottomView.visibility = View.GONE

        binding.btnStart.setOnClickListener {
            if (!isTracking) startDriverTracking() else stopDriverTracking()
        }
    }
```

Fig 4.1: checkUserRole(), setupStudentUI() and setupDriverUI() – role-based UI switching for student and driver views.

The checkUserRole() function reads the current user's UID from Firebase Authentication and fetches the corresponding document from the users collection in Firestore. Based on the role field, it calls either setupStudentUI() or setupDriverUI(). The student UI makes the student bottom navigation visible, initializes the BusListAdapter for the list of active buses, and enables the call action. The driver UI hides the student controls, shows the driver bottom view, and attaches a click listener to the Start/Stop button to control live bus tracking.

4.4.2 Student Map and Real-Time Bus Display

The StudentMapFragment is the main interface for students [17]. It is responsible for displaying the Google Map, showing active university buses in real time, and presenting the "Available buses" list under the map. The major tasks of this fragment are:
(1) Google Map initialization,
(2) obtaining the student's own location,
(3) observing LiveData from StudentMapViewModel, and
(4) updating the UI (markers, polylines, and list).

Google Map initialization:
The fragment uses a SupportMapFragment and the getMapAsync(this) callback to obtain a GoogleMap instance [11]. Once the map is ready, it is centered either on a default campus location or on the student's current GPS position . Basic map settings such as zoom level, map type, and UI controls are also configured in this stage.

Student location updates:
To track the student's own position, the fragment uses the FusedLocationProviderClient to request high-accuracy location updates at a fixed interval [12]. Each new location is stored as a LatLng value (e.g., in a variable named studentLocation), which can later be used to calculate distance and estimated time of arrival (ETA) to nearby buses .
The implementation of these student location requests is shown in Code Snippet 4.4, where startStudentLocationUpdates() registers a LocationCallback and stopLocationUpdates() cancels the updates when they are no longer needed.

Observing LiveData from StudentMapViewModel:
The fragment observes two key LiveData objects exposed by StudentMapViewModel:

- A list of BusDisplay objects containing information about active buses, including driver name, route name, heading, phone number, and the latest latitude/longitude values [15].
- A list of route polylines (List<List<LatLng>>) representing the path of each bus route on the map .

Whenever the ViewModel (fed by the Firestore listener described in Section 4.5 and Code Snippet 4.1) updates these LiveData objects, the observers in StudentMapFragment are triggered automatically.

Updating the UI:
When new data arrives, the fragment refreshes both the map and the list:

- It calls updateMarkers() to add new markers for active buses or move existing markers to their updated positions .
- It calls drawPolylines() to draw or refresh the blue route lines on the map using the polyline point lists [17].

- It uses BusListAdapter together with a RecyclerView to display the list of active buses beneath the map, including a call icon that lets the student dial the driver's phone number directly from the application .

The detailed implementation of updateMarkers() and drawPolylines() is provided in Code Snippet 4.3, which shows how markers are reused from a markerMap, how markers for inactive buses are removed, and how route polylines are rendered on the Google Map.

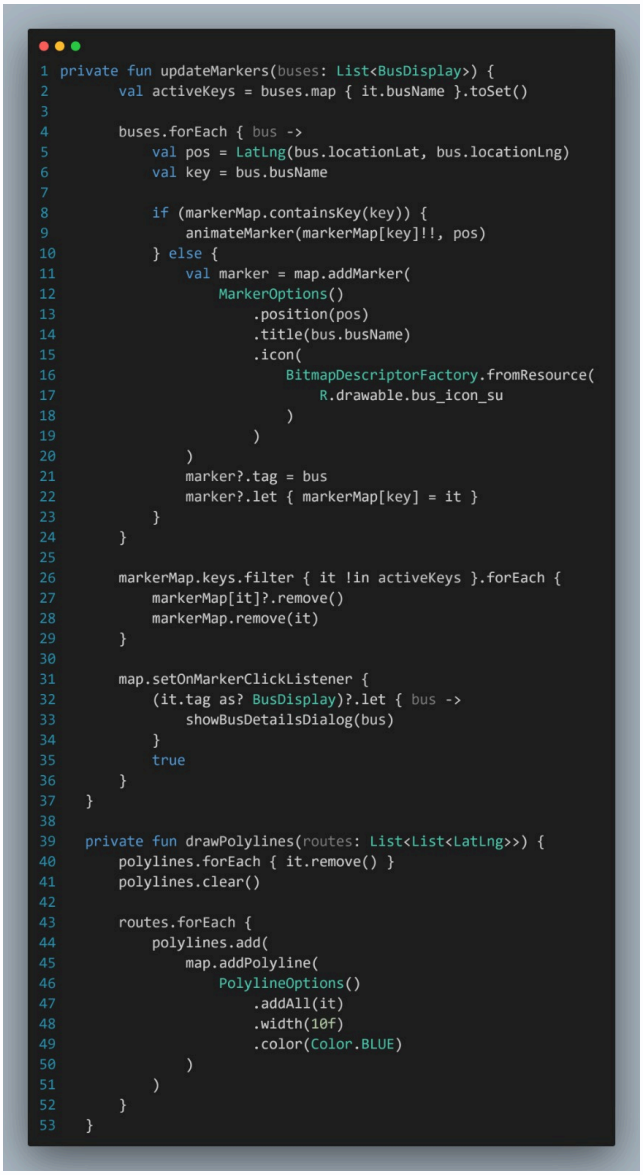
The Google Map is initialized using a SupportMapFragment and the getMapAsync(this) callback, as shown in Fig 4.2.

```
1 val mapFragment =
2             childFragmentManager.findFragmentById(R.id.map) as SupportMapFragment
3         mapFragment.getMapAsync(this)
```

Fig 4.2: Initializing Google Map in StudentMapFragment using SupportMapFragment.

This code retrieves the embedded SupportMapFragment from the layout using its ID and registers the fragment as a callback through getMapAsync(this). When the map is ready, the onMapReady() method of StudentMapFragment is invoked, where markers and polylines are drawn.

The detailed implementation of marker and route updates is shown in Fig 4.3, where the updateMarkers() and drawPolylines() functions manage bus markers and route polylines on the map.

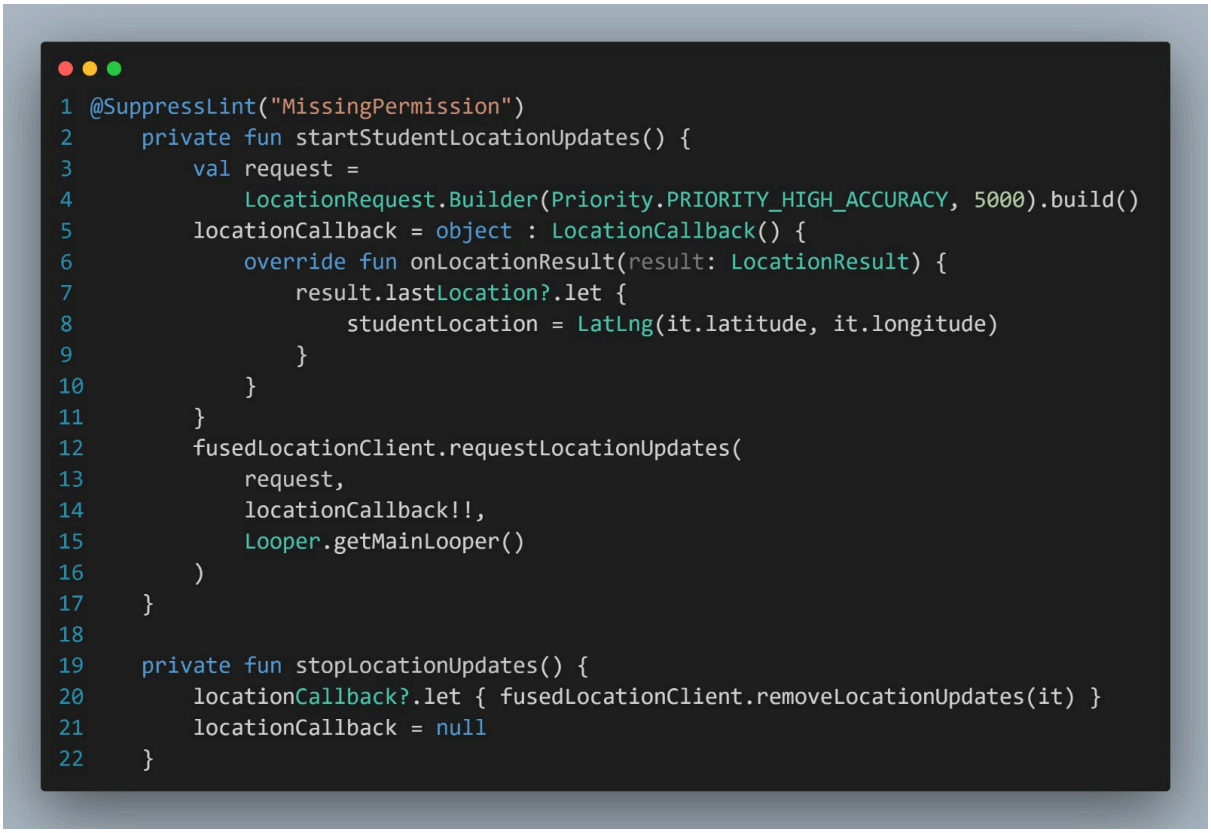


```
private fun updateMarkers(buses: List<BusDisplay>) {
        val activeKeys = buses.map { it.busName }.toSet()

        buses.forEach { bus ->
            val pos = LatLng(bus.locationLat, bus.locationLng)
            val key = bus.busName

            if (markerMap.containsKey(key)) {
                animateMarker(markerMap[key]!!, pos)
            } else {
                val marker = map.addMarker(
                    MarkerOptions()
                        .position(pos)
                        .title(bus.busName)
                        .icon(
                            BitmapDescriptorFactory.fromResource(
                                R.drawable.bus_icon_su
                            )
                        )
                )
                marker?.tag = bus
                marker?.let { markerMap[key] = it }
            }
        }

        markerMap.keys.filter { it !in activeKeys }.forEach {
            markerMap[it]?.remove()
            markerMap.remove(it)
        }

        map.setOnMarkerClickListener {
            (it.tag as? BusDisplay)?.let { bus ->
                showBusDetailsDialog(bus)
            }
            true
        }
    }

    private fun drawPolylines(routes: List<List<LatLng>>) {
        polylines.forEach { it.remove() }
        polylines.clear()

        routes.forEach {
            polylines.add(
                map.addPolyline(
                    PolylineOptions()
                        .addAll(it)
                        .width(10f)
                        .color(Color.BLUE)
                )
            )
        }
    }
```

Fig 4.3: updateMarkers() and drawPolylines() – updating bus markers and drawing route polylines.

The updateMarkers() function first builds a set of active bus keys and then updates the map accordingly. If a marker for a bus already exists in markerMap, it is moved to the new LatLng position; otherwise a new marker with a custom bus icon is created and stored in markerMap. Markers for buses that are no longer active are removed. The drawPolylines() function clears any existing polylines and then draws new blue polylines for each route using the list of LatLng points received from the ViewModel.

The implementation of student location updates using the FusedLocationProviderClient is shown in Fig 4.4.



```
@SuppressLint("MissingPermission")
    private fun startStudentLocationUpdates() {
        val request =
            LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY, 5000).build()
        locationCallback = object : LocationCallback() {
            override fun onLocationResult(result: LocationResult) {
                result.lastLocation?.let {
                    studentLocation = LatLng(it.latitude, it.longitude)
                }
            }
        }
        fusedLocationClient.requestLocationUpdates(
            request,
            locationCallback!!,
            Looper.getMainLooper()
        )
    }

    private fun stopLocationUpdates() {
        locationCallback?.let { fusedLocationClient.removeLocationUpdates(it) }
        locationCallback = null
    }
```

Fig 4.4: startStudentLocationUpdates() and stopLocationUpdates() – requesting GPS location using FusedLocationProviderClient.

This function configures a high-accuracy LocationRequest with a 5-second interval and registers a LocationCallback with the FusedLocationProviderClient. Whenever a new location result is received, the callback updates the studentLocation variable with the latest latitude and longitude. The stopLocationUpdates() function safely removes the callback and stops further GPS updates to save battery when location is no longer needed.

4.4.3 Driver Tracking Module

Drivers use a dedicated UI (or the same map fragment with a different bottom panel) to start and stop tracking [12]:

1. When role == "driver", the fragment calls setupDriverUI(), which:
    - Hides the student bus list,
    - Shows a Start/Stop button for tracking.

2. Starting live tracking
    - On tapping Start, the app:
        - Sets the driver's status field in users to "started".
        - Ensures that a routeId is associated with the driver.
        - Requests high-accuracy location updates from FusedLocationProviderClient.
        - On each update, writes the new location (GeoPoint), heading, and timestamp to Firestore.
3. Stopping live tracking
    - On tapping Stop, the app:
        - Stops the location updates.
        - Sets status to "stopped".
        - As a result, student listeners no longer treat this bus as active.

4.4.4 Admin Module Implementation

The AdminUserManagementFragment (or equivalent Activity) provides the admin interface[13]. Its main screens and functions include:

- Viewing lists of:
    - pending users (status = "pending"),
    - approved users (status = "approved"),
    - blocked users (status = "blocked").
- Approving or rejecting new accounts:
    - Admin selects a user and sets status from "pending" to "approved" or "blocked".
- Assigning roles:
    - Admin can update the role field (student/driver).

These operations are implemented using Firestore queries and update operations on the users collection.

4.5 Real-Time Location Update Logic

Real-time synchronization is implemented mainly inside StudentMapViewModel using a Firestore snapshot listener[15].

1. Attaching the listener
   A query filters documents from the users collection:

Kotlin

```
firestore.collection("users")
    .whereEqualTo("role", "driver")
    .whereEqualTo("status", "started")
    .addSnapshotListener { snapshot, error -> ... }
```

This listener is invoked whenever an active driver starts or stops tracking or moves to a new location.

2. Building the bus list
   Inside the callback:
   - The ViewModel iterates over all driver documents.
   - Extracts fields: name, phone, routeId, heading, location.
   - Uses routeId to fetch the route document from the routes collection when needed.
   - Converts route polylinePoints to List<LatLng> for drawing polylines.
   - Stores everything in a list of BusDisplay objects and a list of route polylines.

3. Publishing LiveData

Kotlin

```
_buses.postValue(busList)
_routePolylines.postValue(polylineList)
```

The fragment, which observes these LiveData objects, automatically updates the map and list.

4. Stopping the listener
   When the fragment or ViewModel is destroyed, the listener registration is removed to stop receiving updates and free resources[15].

```
fun startListening() {
        if (listenerRegistration != null) return  // prevent duplicate listeners

        listenerRegistration = firestore.collection("users")
            .whereEqualTo("status", "started")
            .whereEqualTo("role", "driver")
            .addSnapshotListener { snapshot, error ->
                if (error != null || snapshot == null) return@addSnapshotListener

                viewModelScope.launch(Dispatchers.IO) {
                    val busList = mutableListOf<BusDisplay>()
                    val polylineList = mutableListOf<List<LatLng>>()

                    snapshot.documents.forEach { doc ->
                        val busName = doc.getString("name") ?: "Unknown"
                        val phone = doc.getString("phone") ?: "N/A"
                        val routeId = doc.getString("routeId") ?: ""
                        val heading = doc.getString("heading") ?: "N/A"
                        val location = doc.getGeoPoint("location")

                        val lat = location?.latitude ?: return@forEach
                        val lng = location.longitude

                        var routeName = "Unknown Route"
                        var points: List<LatLng> = emptyList()

                        if (routeId.isNotEmpty()) {
                            try {
                                val routeDoc = firestore.collection("routes")
                                    .document(routeId)
                                    .get()
                                    .await()

                                routeName = routeDoc.getString("routeName") ?: routeName
                                val polylinePoints =
                                    routeDoc["polylinePoints"] as? List<Map<String, Double>>

                                points = polylinePoints?.map {
                                    LatLng(it["lat"]!!, it["lng"]!!)
                                } ?: emptyList()
                            } catch (_: Exception) {}
                        }

                        busList.add(
                            BusDisplay(busName, routeName, heading, phone, lat, lng)
                        )

                        if (points.isNotEmpty()) polylineList.add(points)
                    }

                    _buses.postValue(busList)
                    _routePolylines.postValue(polylineList)
                }
            }
    }

    fun stopListening() {
        listenerRegistration?.remove()
        listenerRegistration = null
    }

    override fun onCleared() {
        super.onCleared()
        stopListening()
    }
```

Fig 4.5: startListening() function – Firestore listener for active buses

This function attaches a real-time snapshot listener to the users collection, filtered by status = "started" and role = "driver". Whenever an active driver document changes, the ViewModel builds a list of BusDisplay objects and loads the corresponding route polylines from the routes collection. The results are posted to the _buses and _routePolylines LiveData objects so that the student map can update automatically. The stopListening() method removes the listener, and onCleared() ensures it is called when the ViewModel is destroyed to prevent memory leaks.

4.6 Error Handling and Security Considerations

To make the application more robust and secure, the following measures were taken:

- Runtime permissions and network checks
    - Location access is requested at runtime; if the user denies permission, the app shows a message and does not start tracking.
    - Before sending or reading data from Firestore, the app checks for network connectivity where possible[15].
- Authentication and Authorization
    - Only logged-in users can reach the main screens.
    - Role and status are checked before enabling student, driver, or admin features.
    - Admin operations are restricted to users with `role = "admin"` / `admin = true`[13].
- Firestore Security Rules (conceptual)
    - Students have read-only access to necessary fields in `users` and `routes`.
    - Drivers can update only their own document (`location`, `heading`, `status`).
    - Admins can change roles and statuses of any user.
- Exception Handling
    - Try–catch blocks around Firestore calls and polyline parsing prevent the application from crashing if a document is missing or malformed.

4.7 Testing Strategy

A combination of manual and black-box testing techniques was used to verify that the system meets its functional and non-functional requirements. The testing strategy includes[13]:

- Unit-level testing (informal) for key ViewModel and repository functions.
- Functional testing for each primary use case:
    - student login and map display,
    - driver tracking,
    - admin approval and blocking.
- Integration testing to check correct interaction between the Android app, Firestore, Authentication, and Google Maps.
- Performance and usability testing to evaluate responsiveness and real-time update delay in real devices on mobile data/Wi-Fi.

4.8 Test Cases and Results

Table 4.1: Test Cases and Results of the Live Bus Tracking System

Test Case ID	Scenario	Input / Steps	Expected Result	Actual Result	Status
TC-01	Student login with valid credentials	1) Open app 2) Enter registered student email & correct password 3) Tap “Log in”	Student is authenticated; role = student; student home/map screen is displayed	As expected	Passed
TC-02	Student login with invalid password	1) Open app 2) Enter valid email but wrong password 3) Tap “Log in”	Login fails; error message “Invalid email or password” shown; user remains on login screen	As expected	Passed
TC-03	Driver login and role-based UI	1) Approved driver enters email & password 2) Tap “Log in”	Driver is authenticated; role = driver; driver tracking UI (Start/Stop button) is displayed instead of student UI	As expected	Passed
TC-04	Driver starts live bus tracking	1) Driver logs in 2) Selects assigned route (if needed) 3) Tap “Start live bus tracking”	Driver document in users collection: status = “started”; location & heading fields begin updating in Firestore	As expected (verified in Firestore)	Passed
TC-05	Student views active buses on map	1) While driver tracking is started 2) Student logs in 3) Opens map screen	Driver status in users collection changes to “stopped”; bus marker is removed/hidden from student map	As expected	Passed

TC-06	Driver stops live bus tracking	1) Driver taps “Stop live bus tracking”	Selected user’s status changes from “pending” to “approved”; user can log in and access features according to role	As expected	Passed
TC-07	Admin approves pending user account	1) Admin logs in 2) Opens pending users list 3) Selects a user 4) Taps “Approve”	Selected user’s status changes from “pending” to “approved”; user can log in and access features according to role	As expected	Passed
TC-08	Admin blocks user and access is denied	1) Admin selects an existing user 2) Sets status = “blocked”	Blocked user cannot access main features; on next login sees error / is redirected back from main screen	As expected	Passed
TC-09	Network loss during driver tracking	1) Driver starts tracking 2) Turn off mobile data/Wi-Fi	App temporarily stops sending updates; no new location appears in Firestore; when network returns, updates resume	As expected (observed on real device)	Passed

To illustrate the actual coding, a few important parts of the implementation are highlighted below[13]-[17]:

4.10 Screenshots of Application Output

To demonstrate the working system, several screenshots are included.
Figure 4.6 shows the login and registration screens of the application.

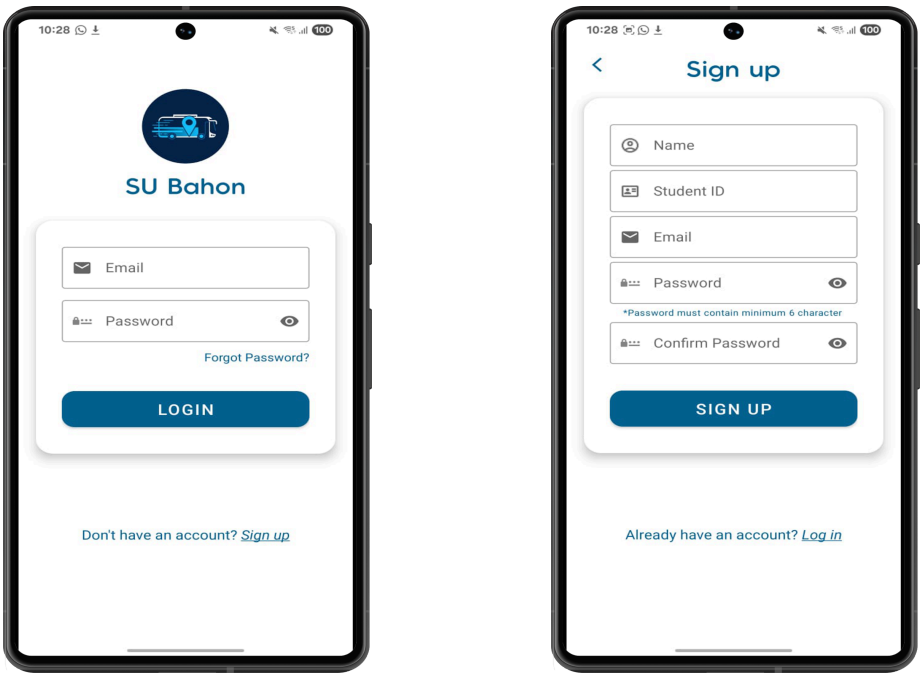


Fig 4.6: Login and registration screens of the SU Bahon application.

Figure 4.7 shows the student home screen, where active university buses are displayed on Google Maps along with the “Available buses” list at the bottom of the screen.

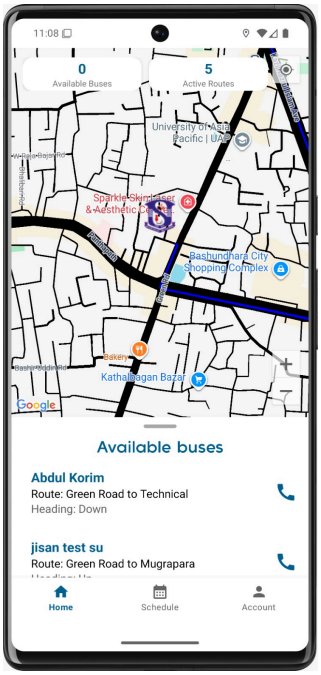


Fig 4.7: Student map screen with active bus markers and the “Available buses” list.

Figure 4.8 shows the driver tracking interface. In this screen an approved driver can start or stop live bus tracking. When the driver taps the START button, the app begins sending GPS updates to Firestore; when the STOP button is pressed, tracking is stopped.

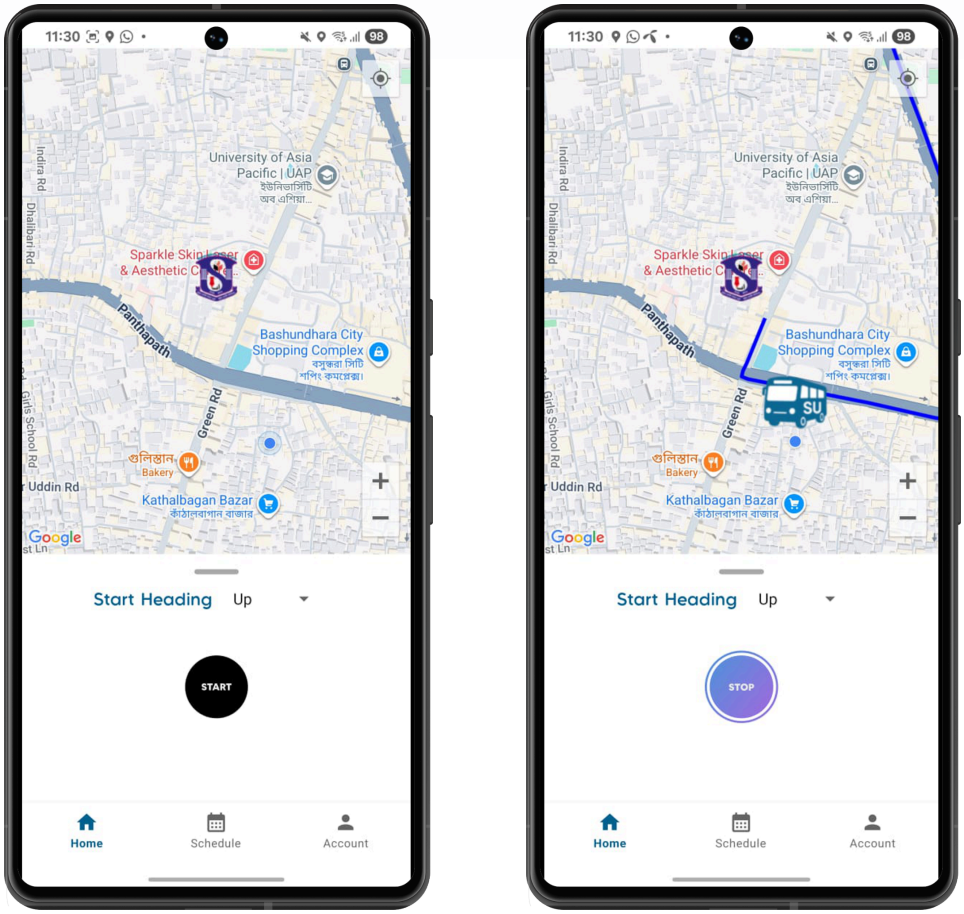


Fig 4.8: Driver tracking screen with Start/Stop button for live bus tracking.

Figure 4.9 shows the admin user management interface. In this screen,an admin can view all registered users, filter them by role or status, and perform actions such as approving pending users, blocking users, or changing their roles between student and driver.

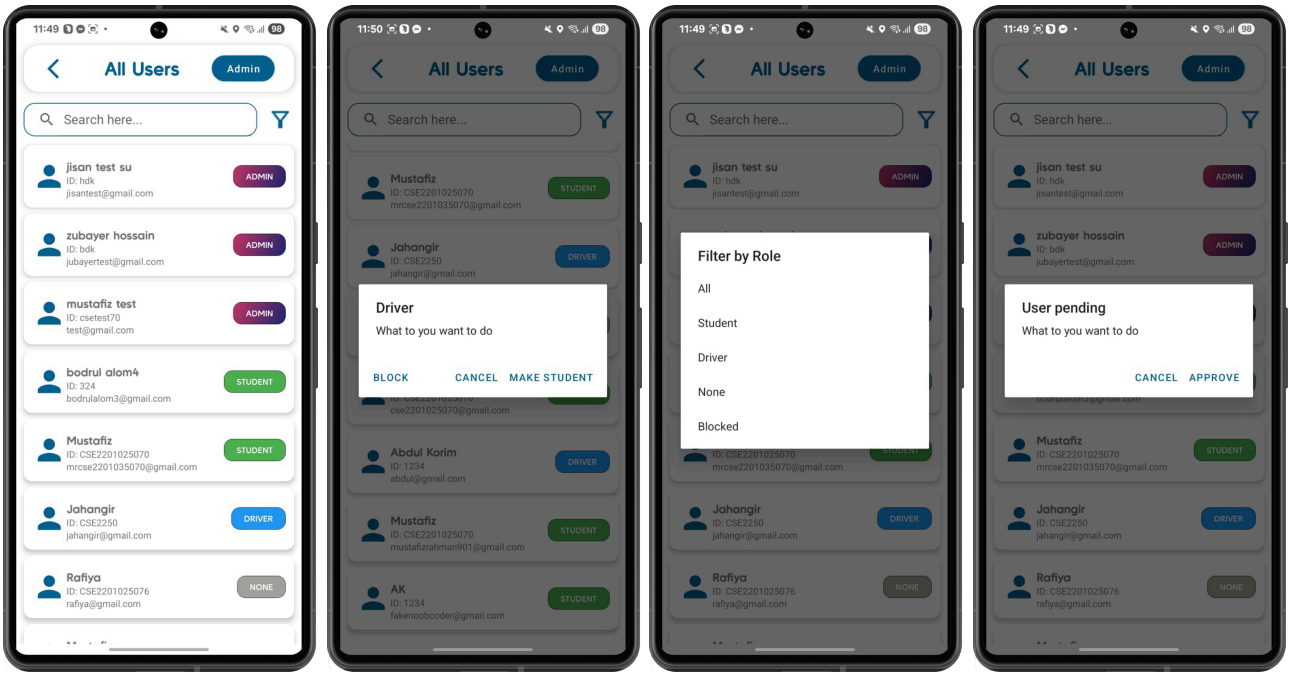


Fig 4.9 Admin user management screens for viewing, filtering, approving and blocking users.

4.10.1 Student Interface Screens

Figure 4.1 and Figure 4.2 illustrate the main student interfaces of the application. Figure 4.1 shows the login and registration screens, while Figure 4.2 presents the student home screen with the Google Map and the “Available buses/routes” list.

Figure 4.5 shows the detailed bus schedule screen, where a student can view the list of buses for a selected day, along with route, driver name and stop-wise timing. Figure 4.6 shows the bus details dialog that appears when the student taps on a bus marker or list item; the dialog displays route name, driver information, heading, ETA/distance and a direct “Call Driver” action.

1) Bus schedule + day filter

Figure 4.10 shows the Bus Schedule screen from the student's point of view. On this screen the student can switch to the Schedule tab and see a list of all university buses for a selected day of the week. For each bus, the app displays the trip type (incoming or outgoing), the driver's name and phone number, and the list of major stops with their approximate times. The day-selection dialog allows the student to choose Saturday, Sunday, Monday, etc., so that the corresponding schedule is loaded dynamically.

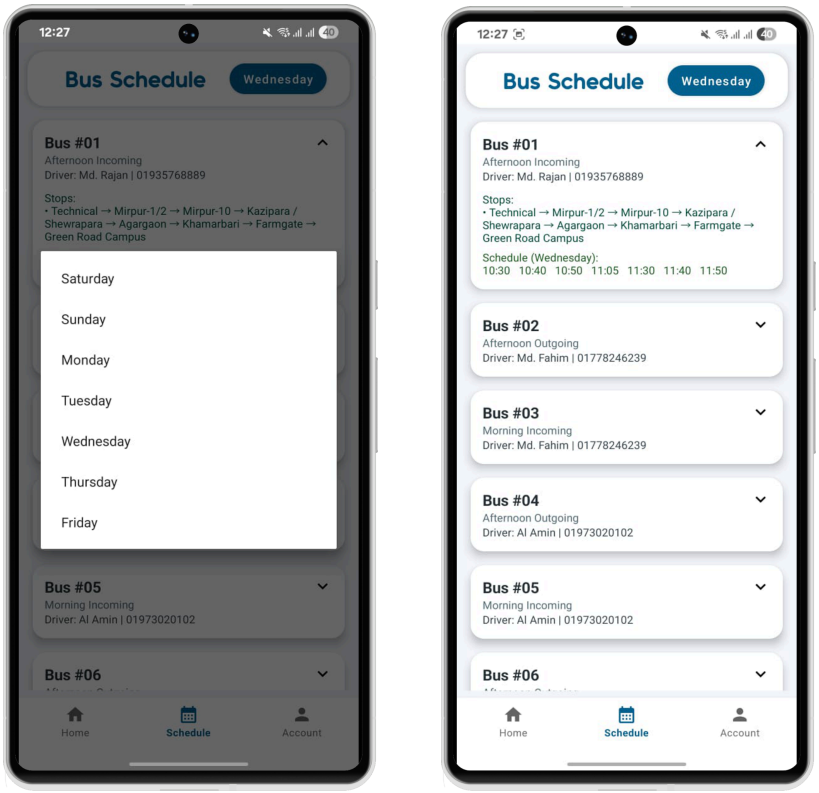


Fig 4.10: Student bus schedule screen and day selection dialog.

– first screenshot = full "Bus Schedule" list
– second screenshot = weekday select dialog

2) Bus details dialog with “Call Driver”

Figure 4.11 illustrates the bus details dialog that appears when a student taps on a bus marker on the map or selects a bus from the available routes list. The dialog summarizes the key information for the selected bus, including the route name, start and end locations, driver name, heading direction, and current status (for example, “Arriving now”). It also shows the approximate distance from the student’s current location and provides a prominent “Call Driver” button so that the student can directly contact the driver if necessary.

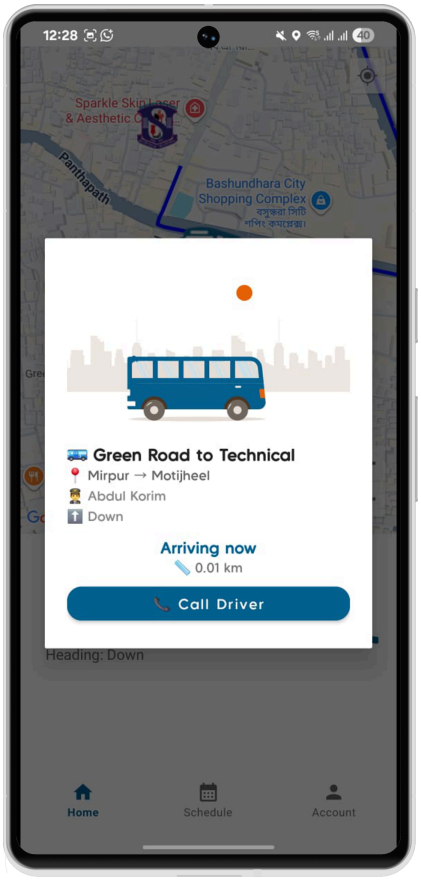


Fig 4.11: Bus details dialog showing route, driver, heading, ETA and “Call Driver” button.

3) First extra map screenshot

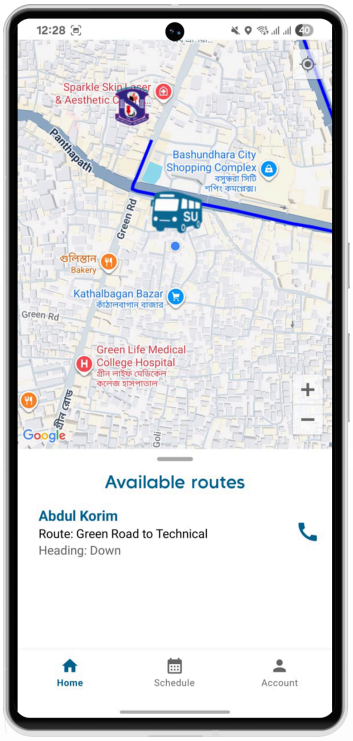


Fig 4.12: Student map screen with active bus marker and the “Available buses/routes” list.

4.11 Source Code

The complete Android Studio project, including all Kotlin source files, XML layouts, and Firebase configuration files, is provided in digital form (CD/USB drive or Git repository link) along with this report for further review and evaluation.

4.12 Summary

This chapter discussed the implementation and testing of the Live Bus Tracking System for Sonargaon University[18]. The system was developed using Kotlin and the MVVM architecture, with Firebase Firestore and Firebase Authentication as the backend, and Google Maps SDK and FusedLocationProviderClient for location and visualization.

Implementation details for user authentication, student map display, driver tracking, and admin management were described, along with the real-time data flow between drivers, Firestore, and student devices. A set of functional and integration tests confirmed that the system meets the key requirements of real-time tracking, role-based access control, and usability for university transportation management.

CHAPTER 5

Results, Discussion, Conclusion and Future Work

5.1 Introduction

The previous chapters described the analysis, design, and implementation of the Live Bus Tracking System and Student Map Application for Sonargaon University. This chapter presents the outcomes of that implementation. It evaluates how far the system has achieved the initial objectives, how it performs in practice, and what benefits it can offer to the different stakeholders of the university.

The chapter is organized into four main parts. Section 5.2 describes the results obtained from implementing the system, including the functioning of the student, driver and admin modules, as well as observations about performance and usability. Section 5.3 discusses these results in the broader context of the problems identified earlier and the related work reviewed in Chapter 2. Section 5.4 concludes the entire project by summarizing the key contributions. Finally, Section 5.5 suggests several directions for future work and possible improvements[3].

5.2 Results

5.2.1 Achievement of Project Objectives

In Chapter 1, several objectives were defined for this project, such as providing real-time bus location to students, enabling drivers to share their GPS position, and giving administrators control over user access. Based on the implementation and testing described in Chapter 4, these objectives have largely been achieved[6]:

- A fully functional Android application was developed using Kotlin and MVVM architecture.
- Students can view the live locations of university buses, along with route information and driver contact details.
- Drivers can start and stop live tracking, and their current locations are automatically sent to Firebase Firestore at regular intervals.
- Admins can approve or block user accounts and assign roles, thereby ensuring that only verified drivers and students use the system.
- Real-time synchronization is implemented using Firestore snapshot listeners, so that map markers on the student side update automatically when a driver moves.

Thus, the core technical and functional goals of the project have been fulfilled.

5.2.2 Functional Results by User Role

The system was tested with different user roles to verify that each group can perform its expected tasks.

Student Role

- Approved students can log in with their credentials and are automatically directed to the student interface.
- The map screen shows the campus and surrounding areas, overlaid with markers representing all active buses.
- The “Available buses” list displays, for each active bus:
    - driver name,
    - assigned route name,
    - heading (e.g., “Up” or “Down”), and
    - a call icon.
- When any active bus moves, its marker on the student’s map interface is updated in real time[6].
- Students can tap the call icon to open the phone dialer with the driver’s number, enabling direct communication when needed.

Driver Role

- Approved drivers log in and are shown the driver tracking interface with a clear Start/Stop tracking button.
- When a driver taps Start live bus tracking, the system:
    - sets their `status` field in the `users` collection to `"started"`,
    - associates the selected `routeId`, and
    - begins uploading GPS coordinates, heading and timestamp to Firestore[13].
- When the driver taps Stop, location updates stop and the `status` field changes to `"stopped"`.
- While tracking is active, students can see the driver’s bus as an active marker on the map; when tracking stops, the marker disappears from the student view[5].

Admin Role

- Admin users log in and access the admin panel, where they can:
    - view lists of pending, approved and blocked users;
    - approve or reject new registrations;
    - change user status (approved/blocked);
    - assign or change user roles (student or driver).
- Any change performed by an admin is immediately reflected in Firestore and therefore affects what that user can or cannot do inside the app. For example, when a driver is blocked, their tracking functions are automatically disabled[14],[15].

These results confirm that the role-based behaviour of the system is working as intended.

5.2.3 User Interface and Usability Results

The user interface was designed to be simple and intuitive. During informal user evaluation[16]:

- Students reported that the home screen showing the map and available bus list was easy to understand, even without written instructions.
- Icons and labels (Home, Schedule, Account, Start, Stop, etc.) were found to be self-explanatory.
- The use of a bottom navigation bar and clear section titles helped users quickly switch between different parts of the application.
- The call icon placed beside each bus entry reduced confusion about how to contact the driver.

Although the usability study was small-scale and informal, the overall feedback suggests that the interface is user-friendly for basic tasks such as checking active buses and starting/stopping tracking.

5.2.4 Performance and Reliability Results

The performance of the system depends mainly on two factors: the network connectivity of driver and student devices, and the response time of Firebase services.

From the tests carried out:

- When both devices had stable mobile data or Wi-Fi connections, the delay between driver movement and marker update on the student's map was short and acceptable for practical use in a university context.
- Firestore snapshot listeners correctly detected changes in the `users` collection and updated the LiveData in the ViewModel without the need for manual refresh.
- The application remained responsive while handling multiple active drivers and routes, thanks to the use of background threads (coroutines) in ViewModels.
- No major crashes were observed during repeated cycles of login, tracking start/stop, and switching roles.

These results indicate that the system is sufficiently reliable and performant for day-to-day bus tracking, as long as internet connectivity is available.

5.2.5 Database and Scalability Results

The use of Firebase Firestore as a NoSQL database brings several advantages:

- The structure of collections (`users`, `routes`, `bus_service`) is flexible and can be extended with new fields without schema migration[13].

- Firestore's automatic scaling means that the same implementation can support more buses or more students without significant architectural changes.
- Adding a new route involves only creating a new document in the routes collection with appropriate polyline points; no application update is necessary.
- Storing driver tracking data directly within the users collection simplifies queries and reduces the need for joinsv[13].

Although the system has not yet been tested with a very large number of concurrent users, the underlying technology is capable of supporting growth as the university transportation network expands.

5.3 Discussion

This section interprets the results in relation to the initial problem statement and the literature review presented in Chapter 2.

5.3.1 Comparison with Existing Solutions

In Chapter 2, several categories of existing systems were reviewed:

Google Maps live transit tracking,

Dhaka city bus tracking attempts, and

international university bus tracking platforms such as TransLoc, NextBus and DoubleMap.

Compared with these, the proposed system has the following characteristics:

It is custom-built for Sonargaon University, using its own routes and buses instead of generic city-wide transports.

It provides a dedicated admin panel for university authorities, which many public apps do not include.

It is implemented using free or low-cost cloud services (Firebase), making it more feasible for a private university in Bangladesh than expensive commercial platforms[13].

The application is tightly integrated with student and driver roles, rather than being a generic passenger information app.

Therefore, the system fills an important gap between high-end commercial tracking platforms and simple, city-scale tracking attempts that are not focused on university buses[9].

5.3.2 Benefits to Stakeholders

The results can also be discussed from the perspective of each stakeholder group.

- Benefits for Students
    - Reduced uncertainty and waiting time at bus stops, as students can see the real-time position of buses and their routes[3].
    - Easier communication with drivers through in-app call functionality.
    - A more organized and official source of information than informal social media groups or phone calls.
- Benefits for Drivers
    - A clear and simple tool for sharing their locations with all students at once, instead of responding to individual calls or messages[5].
    - Integration with route information helps drivers maintain consistent service according to assigned routes.
- Benefits for University Administration
    - Digital record of all registered students and drivers, including their roles and status.
    - Ability to control who can operate as a driver in the system, improving security and trust.
    - Potential to analyse usage patterns in the future (e.g., which routes are most active, at what times)[1].
- Educational and Technical Benefits
    - The project demonstrates practical use of modern technologies (Kotlin, MVVM, Firebase, Google Maps) in a real-world scenario.
    - It can serve as a reference or base project for future student batches interested in mobile development and smart transportation.

5.3.3 Limitations and Challenges

Despite its strengths, the system also has several limitations:

- Dependence on Internet Connectivity
    - Both driver and student apps require active internet connections. In areas with poor coverage, location updates may be delayed or temporarily unavailable[13].
- GPS Accuracy and Battery Usage
    - GPS accuracy can be affected by tall buildings, bad weather, or device limitations.
    - Continuous tracking consumes battery power on the driver's device, which may require careful configuration of update intervals.
- Limited Feature Set in Prototype
    - The current version focuses on core tracking and role management; features such as push notifications, offline maps, detailed schedules, or seat availability have not yet been implemented[4].

- Data Security and Privacy
    - While Firebase Authentication and Firestore security rules provide a strong base, more advanced privacy controls (e.g., data retention policies, encryption of sensitive fields) could be considered if the system is deployed at scale.

Recognizing these limitations is important for planning realistic future improvements.

5.4 Conclusion

The Live Bus Tracking System and Student Map Application for Sonargaon University demonstrates that it is feasible to build a real-time, role-based transportation support system using widely available mobile and cloud technologies. By integrating GPS, Google Maps, Firebase Firestore, and MVVM architecture[13], the project successfully:

- provides students with real-time visibility of active university buses and their routes,
- enables drivers to share their location in a controlled and secure manner, and
- offers administrators the tools to manage user access and maintain the integrity of the system.

From an academic perspective, the project also served as a valuable learning experience in requirements analysis, system design, mobile application development, cloud database design, and software testing[16]. The outcomes align well with the initial objectives and show that such a system could meaningfully improve the daily commuting experience of Sonargaon University students.

5.5 Future Work

Although the current implementation is functional and usable, there are many opportunities to enhance and extend it. Some promising directions for future work are outlined below.

1. Push Notifications and Alerts
    - Integrate Firebase Cloud Messaging (FCM) to send notifications when:
        - a bus on a selected route becomes active,
        - a bus is approaching a particular stop, or
        - there is a sudden change in schedule or service disruption[13].
    - This would further reduce the need for students to constantly open the app to check bus status.
2. More Accurate ETA and Analytics
    - Combine current location, historical speed data, and route distance to calculate accurate Estimated Time of Arrival (ETA).
    - Store historical tracking data in a separate collection and analyse it to understand peak times, delays, and route performance[1].

3. Bus Capacity and Seat Availability
    - Allow drivers to indicate the approximate occupancy level (e.g., low/medium/full) or use sensors in future hardware-integrated versions.
    - Display this information in the student interface so that students can choose less crowded buses.
4. Schedule and Timetable Management
    - Add a scheduling module where the university administration can publish regular timetables and special services (exam days, events, etc.).
    - Combine schedule data with real-time tracking to show whether a particular bus is on time or delayed[2].
5. Multi-Platform and Web Dashboard
    - Develop an iOS version of the mobile application to support students with iPhones.
    - Implement a web-based admin dashboard for transportation staff to monitor buses from desktop computers.
6. Offline Support and Data Caching
    - Cache last-known locations and route information locally so that basic information remains visible even when a device temporarily loses connectivity.
    - Automatically synchronize pending data once the network is restored.
7. Enhanced Security and Privacy
    - Design and enforce more detailed Firestore security rules with role-based access at field level[15].
    - Introduce options for students to control how their personal data (such as phone numbers or student IDs) is stored and displayed.
8. Integration with Other University Systems
    - In the long term, the bus tracking system could be integrated with the university's student portal, ID card system, or attendance system, providing richer services such as automatic attendance logging when students board a bus.Implementing these improvements would transform the current prototype into a comprehensive, production-ready platform capable of supporting the long-term transportation needs of Sonargaon University and potentially other institutions as well.

References

[1] M. H. Abid, A. Islam, A. D. Biswas, and I. A. Talin, “IoT-based vehicle tracking system for Khulna University,” *Khulna University Studies*, pp. 925–935, 2022.

[2] K. Ammar, M. Jalmoud, A. Boushehri, and K. Fakhro, “A real-time school bus tracking and monitoring system,” in *Proc. 2019 IEEE 10th Annu. Inf. Technol., Electron. and Mobile Commun. Conf. (IEMCON)*, Vancouver, BC, Canada, Oct. 2019, pp. 654–660.

[3] S. A. Sharif, M. S. Suhaimi, N. N. Jamal, I. K. Riadz, I. F. Amran, and D. N. Jawawi, “Real-time campus university bus tracking mobile application,” in *Proc. 2018 7th ICT Int. Student Project Conf. (ICT-ISPC)*, Jul. 2018, pp. 1–6.

[4] M. S. Sulaiman, S. Syamsul, F. H. Yusoff, Z. Derasit, A. Ismail, and N. H. I. Teo, “OnBoard: A real-time bus tracking mobile application for university campus,” in *Proc. 2025 IEEE 7th Symp. Comput. and Informat. (ISCI)*, Aug. 2025, pp. 93–98.

[5] M. N. Hasan and M. S. Hossen, “Development of an Android based real time bus tracking system,” in *Proc. 2019 1st Int. Conf. Adv. Sci., Eng. and Robotics Technol. (ICASERT)*, May 2019, pp. 1–5.

[6] M. S. Alamgir, I. Jahan, N. Aktar, and A. N. Ramisa, “Chittagong University teachers’ bus tracking system using smartphone application,” in *Proc. 2018 4th Int. Conf. Electr. Eng. and Inf. & Commun. Technol. (iCEEiCT)*, Sep. 2018, pp. 199–203.

[7] TransLoc, “TransLoc real-time bus tracking for universities and transit agencies.” Accessed: Jan. 5, 2026. [Online]. Available: https://transloc.com

[8] NextBus, “NextBus real-time passenger information system.” Accessed: Jan. 5, 2026. [Online]. Available: https://www.nextbus.com

[9] DoubleMap, “DoubleMap bus tracking solutions.” Accessed: Jan. 5, 2026. [Online]. Available: https://doublemap.com

[10] Google, “Google Maps Platform documentation.” Accessed: Jan. 5, 2026. [Online]. Available: https://developers.google.com/maps/documentation

[11] Google, “Google Maps SDK for Android – overview.” Accessed: Jan. 5, 2026. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk

[12] Google, “Fused Location Provider API.” Accessed: Jan. 5, 2026. [Online]. Available: https://developers.google.com/location-context/fused-location-provider

[13] Google Firebase, “Cloud Firestore documentation.” Accessed: Jan. 5, 2026. [Online]. Available: https://firebase.google.com/docs/firestore

[14] Google Firebase, “Firebase Authentication documentation.” Accessed: Jan. 5, 2026. [Online]. Available: https://firebase.google.com/docs/auth

[15] Google Firebase, “Cloud Firestore security rules.” Accessed: Jan. 5, 2026. [Online]. Available: https://firebase.google.com/docs/firestore/security/get-started

[16] Android Developers, “Guide to app architecture.” Accessed: Jan. 5, 2026. [Online]. Available: https://developer.android.com/topic/architecture

[17] Android Developers, “RecyclerView.” Accessed: Jan. 5, 2026. [Online]. Available: https://developer.android.com/guide/topics/ui/layout/recyclerview

[18] Android Developers, “SupportMapFragment.” Accessed: Jan. 5, 2026. [Online]. Available: https://developer.android.com/reference/com/google/android/gms/maps/SupportMapFragment