THE UNIVERSITY OF ZAMBIA

SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTING AND INFORMATICS

FINAL YEAR PROJECT REPORT

ACADEMIC YEAR 2026

Implementation of a Web-Based Real-Time Public Bus Tracking System for Lusaka

A Final Year Project Report Submitted to the
Department of Computing and Informatics
in partial fulfilment of the requirements for the award of

Bachelor of Science in Computer Science

By

Kenneth Kaunda Musonda
Computer Number: 2020056283

Group members:
Madalitso Nyirenda (Computer Number: 2021502597)
Ngosa Muleba (Computer Number: 2022044371)

Supervisor: Dr. Lighton Phiri

Date of Submission: 23rd October, 2026.

DECLARATION

I declare that this final year project report is my own work and has been prepared in accordance with the academic requirements of the University of Zambia. All sources of information, ideas, data, software, images, and other material that are not my own have been appropriately acknowledged and referenced.

This project was undertaken as part of a group. I have clearly identified my individual contributions in Chapter 5, Chapter 6 and Appendix A, and I have not presented the work of other group members as my own.

Student Name:

_____

Signature:

_____

Computer Number:

_____

Date:

_____

APPROVAL

This final year project report has been submitted for examination with the approval of the project supervisor.

Supervisor Name:

_____

Signature:

_____

Date:

_____

Departmental/Examiner Approval (if applicable):

_____

ABSTRACT

Passengers who travel by bus in Lusaka often lack dependable, route-specific information about where a bus is and when it will reach their stop. This project, undertaken by a group of three students, set out to implement a web-based real-time public bus tracking system for Lusaka that gives passengers route-based journey information and an estimated arrival time, and gives drivers a view of the passengers waiting along their route. The project began from a proposal that assumed simulated GPS data. After supervisor feedback, the design was changed to use real coordinates and speed from a driver's GPS-enabled mobile device. Development was iterative and incremental, informed by requirements analysis, a study of the interaction concepts of the Yango application adapted to predefined bus routes, and a review of related systems.

The system was built as a web application with browser interfaces for passengers and drivers and a Django REST Framework server. It imports static GTFS route, stop, trip and stop-time data, authorises drivers to operate their assigned buses, maintains the trip, position, speed and status of each bus from its GPS updates, manages passengers' waiting requests, and estimates arrival times from road durations obtained from the OSRM routing service. It was verified functionally through automated tests of the server-side logic and manual checks of the deployed passenger and driver flows. The three intended features, Real-Time Bus Tracking, Journey Planning and Arrival Time Prediction, were implemented at prototype level. The evidence is limited. No operational bus fleet or operator data feed was available, so the GPS positions could not be verified as those of an operating bus, and no field trial, user study or performance benchmark was carried out. The system is therefore a verified prototype, not a proven reduction in waiting time.

Keywords: bus tracking; real-time GPS; Django REST Framework; GTFS; estimated time of arrival

ACKNOWLEDGEMENTS

I thank my supervisor, Dr. Lighton Phiri, for his guidance and for the regular project reviews, whose feedback shaped the direction of the project, most notably the decision to replace simulated GPS data with real GPS data. I acknowledge the University of Zambia and the Department of Computing and Informatics for the academic environment in which this work was carried out.

I also acknowledge my group members, Madalitso Nyirenda and Ngosa Muleba, with whom the project context, the requirements analysis and the integrated system were shared, and whose work on the browser interfaces the backend described in this report serves.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....13
1.1 Background and Context.....13
1.2 Problem Statement.....13
1.3 Project Aim.....14
1.4 Project Objectives.....14
1.5 Scope of the Project.....15
1.6 Significance of the Project.....16
1.7 Limitations.....16
1.8 Organisation of the Report.....17
CHAPTER 2: LITERATURE REVIEW.....17
2.1 Introduction.....17
2.2 Key Concepts and Theories.....18
2.3 Existing Systems / Related Work.....21
2.4 Comparative Analysis.....22
2.5 Research / Knowledge Gap.....23
2.6 Conceptual or Theoretical Framework.....23
2.7 Chapter Summary.....24
CHAPTER 3: METHODOLOGY.....24
3.1 Introduction.....24
3.2 Research / Development Approach.....25
3.3 Requirements / Data Collection.....28
3.4 System Architecture / Research Workflow.....30
3.5 Tools and Technologies.....30
3.6 Ethical and Legal Considerations.....31
3.7 Evaluation Method.....32
3.8 Chapter Summary.....32
CHAPTER 4: SYSTEM ANALYSIS AND DESIGN.....32
4.1 Introduction.....32
4.2 Stakeholder and User Analysis.....33
4.3 Functional Requirements.....33
4.4 Non-Functional Requirements.....34
4.5 Use Cases / User Stories.....35
4.6 System / Solution Architecture.....40
4.7 Database / Data Design.....42
4.8 Interface / Interaction Design.....45
4.9 Algorithms / Models / Technical Design.....47

4.10 Chapter Summary....52
CHAPTER 5: IMPLEMENTATION / DEVELOPMENT....52
5.1 Introduction....52
5.2 Development Environment....52
5.3 System Components / Modules....53
5.4 Key Implementation Details....54
5.5 Integration....57
5.6 Challenges and Solutions....57
5.7 Individual Contribution (Group Projects)....58
5.8 Chapter Summary....59
CHAPTER 6: TESTING, RESULTS AND EVALUATION....59
6.1 Introduction....59
6.2 Testing Strategy....59
6.3 Test Cases and Results....60
6.4 Experimental Results / Performance Results....69
6.5 User Evaluation....69
6.6 Analysis of Results....69
6.7 Objective-by-Objective Evaluation....71
6.8 Individual Results....71
6.9 Chapter Summary....72
CHAPTER 7: DISCUSSION....72
7.1 Introduction....72
7.2 Interpretation of Findings....72
7.3 Comparison with Existing Work....73
7.4 Contributions of the Project....73
7.5 Limitations and Implications....73
7.6 Chapter Summary....74
CHAPTER 8: CONCLUSION AND RECOMMENDATIONS....74
8.1 Introduction....74
8.2 Summary of the Project....74
8.3 Conclusions....74
8.4 Achievement of Objectives....75
8.5 Recommendations....75
8.6 Future Work....75
REFERENCES....76
APPENDICES....77
APPENDIX A: INDIVIDUAL CONTRIBUTION STATEMENT (GROUP PROJECTS)....77

APPENDIX B: USER MANUAL / INSTALLATION GUIDE.....81
APPENDIX C: ADDITIONAL SYSTEM DESIGNS / DIAGRAMS.....82
APPENDIX D: ADDITIONAL TEST CASES AND RESULTS.....83
APPENDIX E: DATA COLLECTION INSTRUMENTS.....86
APPENDIX F: ADDITIONAL CODE / CONFIGURATION.....87
APPENDIX G: OTHER SUPPORTING MATERIAL.....89

LIST OF FIGURES

Figure 2.2.1.1 Intelligent Transportation Systems and bus tracking (conceptual).....18
Figure 2.2.2.1 Generic GPS-based bus tracking information flow (conceptual).....19
Figure 2.2.4.1 Real-time passenger information flow (conceptual).....20
Figure 2.2.7.1 Relationship between bus, route, stop and ETA (conceptual).....21
Figure 2.6.1 Conceptual framework derived from the reviewed literature.....24
Figure 3.2.1.1 Iterative and incremental development process of the Bus Tracking System.....26
Figure 4.5.1.1 UML use-case diagram of the Bus Tracking System.....36
Figure 4.5.3.1 System sequence diagram: authorised driver starts a trip and updates the bus GPS position.....38
Figure 4.5.3.2 System sequence diagram: passenger plans a journey, creates a waiting request and retrieves the ETA.....39
Figure 4.6.1 System architecture of the Bus Tracking System.....41
Figure 4.7.1.1 Domain model of the Bus Tracking System (conceptual).....43
Figure 4.7.2.1 Entity-relationship diagram of the persisted data.....44
Figure 4.8.1 Passenger interface screens for route search, boarding-stop selection and waiting-request creation (interface developed by other group members).46
Figure 4.8.2 A waiting request in the WAITING state (interface developed by other group members).....46
Figure 4.8.3 A waiting request in the ON_BOARD state (interface developed by other group members).....46
Figure 4.9.1.1 Design class diagram of the system modules and persistent classes.....48
Figure 4.9.1.2 Interaction diagram for the system operation updateBusLocation.....49
Figure 4.9.1.3 Interaction diagram for the system operation getWaitingRequestEta.....50
Figure 6.3.1 Console output of python manage.py test: 35 tests run, result OK.....62
Figure 6.3.2 Landing page of the deployed system (interface developed by other group members).....63
Figure 6.3.3 Driver sign-in (interface developed by other group members).....63
Figure 6.3.4 Driver dashboard with the assigned bus selected and its status IDLE (interface developed by other group members).....64
Figure 6.3.5 Bus activated and destination choice for the route Bauleni – Kulima Tower (interface developed by other group members).....64
Figure 6.3.6 Browser permission request and waiting for a device GPS fix (interface developed by other group members).....65
Figure 6.3.7 GPS update accepted, reported location, speed, status and live map (interface developed by other group members).....65

Figure 6.3.8 Passenger map showing the bus marker (interface developed by other group members)..... 66
Figure 6.3.9 Passenger sign-in (interface developed by other group members).....66
Figure 6.3.10 Route search and destinations offered (interface developed by other group members).....67
Figure 6.3.11 Boarding-stop selection (interface developed by other group members). 67
Figure 6.3.12 Waiting request created, with the arrival estimate and the options to board or cancel (interface developed by other group members).....68
Figure 6.3.13 Passenger waiting request (left) and the driver dashboard showing one passenger waiting at Civic Center (right) (interface developed by other group members)..... 68
Figure 6.3.14 Django administration site of the deployed system.....69

LIST OF TABLES

Table 1.4.1 Proposal objectives and their treatment in the final project.....14
Table 1.5.1 Scope of the project as implemented.....15
Table 2.4.1 Comparison of reviewed systems and standards with the Bus Tracking System.....22
Table 3.2.2.1 Stages of development and the work done in each stage.....26
Table 3.2.4.1 Proposal compared with the implemented system.....27
Table 3.3.3.1 Adaptation of ride-hailing interaction ideas to a public bus service.....29
Table 3.3.5.1 Contents of the GTFS files stored in the repository.....29
Table 3.5.1 Principal tools and technologies.....30
Table 4.1.1 Analysis and design artefacts used in this report.....32
Table 4.2.1 Stakeholders and their needs.....33
Table 4.3.1 Functional requirements implemented by the system.....34
Table 4.4.1 Non-functional requirements and design response.....35
Table 4.5.2.1 Brief descriptions of the principal use cases.....37
Table 4.5.3.1 System operations and the endpoints that implement them.....40
Table 4.7.2.1 Data dictionary of the principal entities.....45
Table 4.9.3.1 Rules for next-stop update and bus status.....51
Table 5.3.1 System components and their responsibilities.....53
Table 5.7.1 Contribution summary by activity.....58
Table 6.3.1 Behaviours verified by the automated tests.....60
Table 6.6.1 Defects and technical limitations identified in the system.....70
Table 6.7.1 Evaluation of the objectives.....71
Table A.1 Project and group information.....77
Table A.2 Nature of the individual contribution by activity.....79
Table D.1 Test methods.....83
Table F.1 API endpoints of the integration branch.....87
Table F.2 Environment variables read by the application.....88

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
BRT	Bus Rapid Transit
CSRF	Cross-Site Request Forgery
DRF	Django REST Framework
ER	Entity-Relationship
ETA	Estimated Time of Arrival
GPS	Global Positioning System
GTFS	General Transit Feed Specification
HTTP	Hypertext Transfer Protocol
ITS	Intelligent Transportation Systems
JSON	JavaScript Object Notation
ORM	Object-Relational Mapping
OSM	OpenStreetMap
OSRM	Open Source Routing Machine
REST	Representational State Transfer
SSD	System Sequence Diagram
UML	Unified Modeling Language
URL	Uniform Resource Locator
WSGI	Web Server Gateway Interface

CHAPTER 1: INTRODUCTION

1.1 Background and Context

Public transport depends on information as much as on vehicles. A passenger who intends to travel by bus needs to know which route to use, where to board, whether a bus is operating on that route and roughly when it will arrive. The project proposal that started this work identified buses as the primary mode of transport in Lusaka and described passengers who rely on guesswork and informal information while waiting at stops.

Intelligent Transportation Systems (ITS) apply computing, positioning and communication technologies to transport, and recent reviews identify real-time monitoring and smart public transport as active areas [1]. GPS-based bus tracking is one application. Khalid, Salleh and Samsudin showed that consumer-grade GPS, mobile Internet and a central server can provide passengers with bus positions and arrival estimates [2], and Watkins et al. found that real-time information affects both perceived and actual waiting time for transit users [3]. Chapter 2 reviews this literature and its limits.

The project was carried out by a group of three students: Madalitso Nyirenda (2021502597), Ngosa Muleba (2022044371) and Kenneth Kaunda Musonda (2020056283), under the supervision of Dr. Lighton Phiri. The group built one integrated system, the *Bus Tracking System*, made of browser interfaces for passengers and drivers, a Django server with an API, a database and a road-routing service. This report describes the shared project, its requirements, architecture and overall system, and evaluates the system as a whole. The individual contribution which is the backend of the system and its deployment configuration, is identified in Section 5.7, Section 6.8 and Appendix A, and the work of the other members is acknowledged where it is discussed.

A significant change occurred during the project. The proposal planned to *simulate* bus movement with a GPS simulation tool. Following supervisor feedback, the design was revised so that the system accepts **real GPS coordinates and speed** from a GPS-enabled device used by the bus driver. The final system, and therefore this report, is based on real GPS updates. Simulation appears in this report only as an earlier prototype stage that was replaced.

1.2 Problem Statement

Passengers who wait for a bus in Lusaka generally have no system-provided, route-specific information on where a given bus is and when it will reach the stop at which they are waiting. A raw coordinate would not solve this on its own. A coordinate is useful to a passenger only when the system can relate it to a route, a direction of travel, an ordered list of stops and the stop at which that passenger intends to board.

The specific problem addressed by the project is therefore how to provide a web-based system that (a) receives real position updates only from an authorised driver for a bus that the driver is permitted to operate, (b) keeps the current operational state of each bus, (c) connects that state to structured route, trip and stop data, (d) lets passengers choose a route and boarding stop and record that they are waiting, (e) shows drivers where passengers are waiting, and (f) returns an estimated arrival time only when enough valid information exists to calculate one. The

reviewed literature does not allow a claim that no digital bus-tracking system exists anywhere in Zambia; the gap addressed is a localised implementation gap, developed in Section 2.5.

1.3 Project Aim

The aim of the group project is to implement a web-based real-time public bus tracking system for Lusaka that uses real driver-device GPS updates, structured transport data and a road-routing service to support three features: **1. Real-Time Bus Tracking, 2. Journey Planning** and **3. Arrival Time Prediction**.

1.4 Project Objectives

The proposal listed five objectives. They were reviewed against what was built, and Table 1.4.1 shows how each one was treated in the final project. The first objective was changed because of the move from simulated to real GPS.

Table 1.4.1: Proposal objectives and their treatment in the final project

Proposal objective	Objective as implemented
1. Implement real-time bus tracking using generated GPS data	Real-Time Bus Tracking from real GPS coordinates and speed sent by an authorised driver device (changed after supervisor feedback)
2. Build a passenger web UI to view bus locations and routes	A passenger web interface that shows active buses on a map and lets a passenger search a route, choose a destination and boarding stop, and place a waiting request (Journey Planning)
3. Provide estimated arrival times at specific stops	Arrival Time Prediction for a passenger's waiting request, from OSRM road-route duration
4. Enable operators to monitor and manage the fleet	Bus, driver, assignment and transport records can be managed through the Django administration site.
5. Allow drivers to see active passengers at pickup points	The driver dashboard shows the number of waiting passengers for each stop of the bus's current trip

For the individual work, the objectives are:

1. to design the persistent data model and implement GTFS import commands that load routes, stops, trips and stop times;
2. to authenticate drivers and authorise every driver operation through an active driver and an active driver-to-bus assignment;
3. to implement bus activation, deactivation, operational reset and trip start;
4. to process real GPS updates and maintain bus status and next-stop state;
5. to implement waiting-request services and OSRM-based arrival-time logic that declines to estimate when the data is invalid; and
6. to verify the arrival-time rules with automated tests and to prepare the deployment configuration.

1.5 Scope of the Project

Table 1.5.1 states the scope of the project as it was implemented in the final system. It lists the five items that the proposal placed in scope, all of which were implemented at prototype level, and the four items that remained out of scope. The last column states what was delivered for each item.

Table 1.5.1: Scope of the project as implemented

Scope item	Scope	What was delivered
Real-time GPS tracking of buses	In scope	Implemented. The driver dashboard sends the device's coordinates and speed, and the server accepts them only from an authorised driver (Section 5.4.5). The proposal's simulated GPS data was replaced after supervisor feedback.
Display of bus routes and stops on a map	In scope	Implemented. The passenger interface shows active buses as markers on a map, with their status and speed, and refreshes them every 15 seconds. Routes, destinations and the stops of a trip are presented as lists in the trip-planning panel.
Web application for passengers (browser-based)	In scope	Implemented. A passenger can register, sign in, search a route, choose a destination and a boarding stop, place a waiting request, mark it as boarded, cancel or complete it, and see its arrival estimate.
Driver interface: active passengers at pickup	In scope	Implemented. The driver dashboard offers sign-in, bus selection, activation, trip start, GPS tracking and the number of passengers waiting at each stop of the trip.
Admin dashboard for transport operators	In scope	Implemented through the Django administration site, where operators manage buses, drivers, assignments and transport records. No separate custom operator dashboard was built.
Online ticket payment system	Out of scope	Remains out of scope.
Integration with government transport systems	Out of scope	Remains out of scope.
Physical GPS hardware installation in buses	Out of scope	Remains out of scope. Positions come from the driver's own device.
Native mobile app (iOS / Android)	Out of scope	Remains out of scope. The system is accessed through a web browser.

Within this scope, the author's individual work is the backend and the deployment configuration (Section 5.7); the interfaces were developed by the other group members and are credited where they are discussed. The statements about what was not done, such as the absence of city-wide deployment, user acceptance testing and performance testing, are limitations of the evidence and are given in Section 1.7.

1.6 Significance of the Project

The project shows how accessible components can be combined into one working pipeline: a GPS-enabled device used by the driver, structured GTFS-derived route data, a Django backend that keeps the state of each bus, and a road-routing service. It gives passengers a route-aware way to choose a bus, to register at a boarding stop and to obtain an arrival estimate, and it gives

drivers a controlled way to announce their position. For the university it provides a documented, tested base that later groups can extend. For the author it demonstrates the application of object-oriented analysis and design to a real system, using notation from Larman [4].

The significance should not be overstated. The project has not been evaluated by passengers or drivers, it has not run as a city-wide service, and its estimates have not been compared against observed arrival times. Its contribution is a verified prototype, not a proven reduction in waiting time.

1.7 Limitations

The limitations of the project are stated here as limitations of access, time, scope and evidence, data, technology, test coverage, security and deployment evidence. They affect how the results should be read and are discussed further in Sections 6.6 and 7.5.

- **Access to an operational fleet and operator data.** This was the principal limitation. The project had no access to an operational public-bus fleet or to an official bus-operator data feed. Although the system used real GPS coordinates from a GPS-enabled mobile device, the device location could not be independently verified as the location of an operating bus. Route, stop, trip and stop-time information was imported from static GTFS files, so live service changes could not be received automatically. The evaluation therefore focused on functional verification of the implemented workflows, not on a long-term field trial involving operational buses and passengers.
- **Scope and evidence.** No user study, user acceptance test, field trial or performance benchmark was carried out, although the proposal planned user testing and performance targets. The project therefore shows that the system works under test conditions. It does not show that the system improves passengers' waiting experience. The test evidence is the automated tests and manual functional and integration checks of the deployed flows (Section 6.3); neither is exhaustive acceptance testing.
- **Data.** No observed arrival times were collected, and the automated tests use synthetic fixtures with a mocked routing function, so the accuracy of the ETA could not be measured.
- **Technology.** Arrival times use OSRM road durations and do not model live traffic or dwell time at stops. OSRM is an external service. Only the current state of each bus is stored, so no location history is kept.
- **Test coverage.** The 35 automated tests exercise the server: the ETA, the waiting overview, access control, registration, login throttling and the debug default. There are no automated tests of the browser interfaces, which were checked manually, and the tests do not cover the GTFS import commands, the status and next-stop thresholds, validation of GPS values or a real OSRM instance.
- **Security.** Access control, password validation and login throttling were added in the final stage of development. Some weaknesses remain (Table 6.6.1), and no penetration test or independent security review was carried out.

1.8 Organisation of the Report

Chapter 2 reviews the literature, compares existing systems and states the research gap and conceptual framework. Chapter 3 presents the methodology, including the requirements analysis and the iterative development approach. Chapter 4 gives the system analysis and design: requirements, use cases, system sequence diagrams, architecture, domain model, entity-relationship design and design model. Chapter 5 documents the implementation and individual contribution. Chapter 6 reports testing and evaluation within the limits of the evidence. Chapter 7 discusses the findings, and Chapter 8 concludes and recommends future work. The references and appendices follow; Appendix A is the Individual Contribution Statement.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter reviews the concepts and related work that informed the Bus Tracking System. It covers Intelligent Transportation Systems, GPS-based vehicle tracking, estimated time of arrival, real-time passenger information, existing systems, public-transport data standards and the software-engineering approach. Existing systems are compared, a research gap is identified and a conceptual framework is derived.

A note on the figures in this chapter is necessary. Figures 2.2.1.1 to 2.6.1 are **conceptual diagrams** drawn from the literature to explain ideas. They are not diagrams of the deployed architecture of the Bus Tracking System; that architecture is presented in Chapter 4.

2.2 Key Concepts and Theories

2.2.1 Intelligent Transportation Systems

Intelligent Transportation Systems (ITS) apply computing, sensing, communication and information technologies to transport. In public transport they support vehicle monitoring, passenger information, fleet management and route planning. A recent systematic review of ITS in public and shared urban transport identifies smart infrastructure, data-driven mobility and smart public transportation among the prominent research areas [1]. The review treats ITS as an integrated technological environment rather than a single technology. A bus-tracking application is one component of that environment: positioning supplies the location of a vehicle, communication carries it to a server, and software processes it and presents it to passengers (Figure 2.2.1.1).

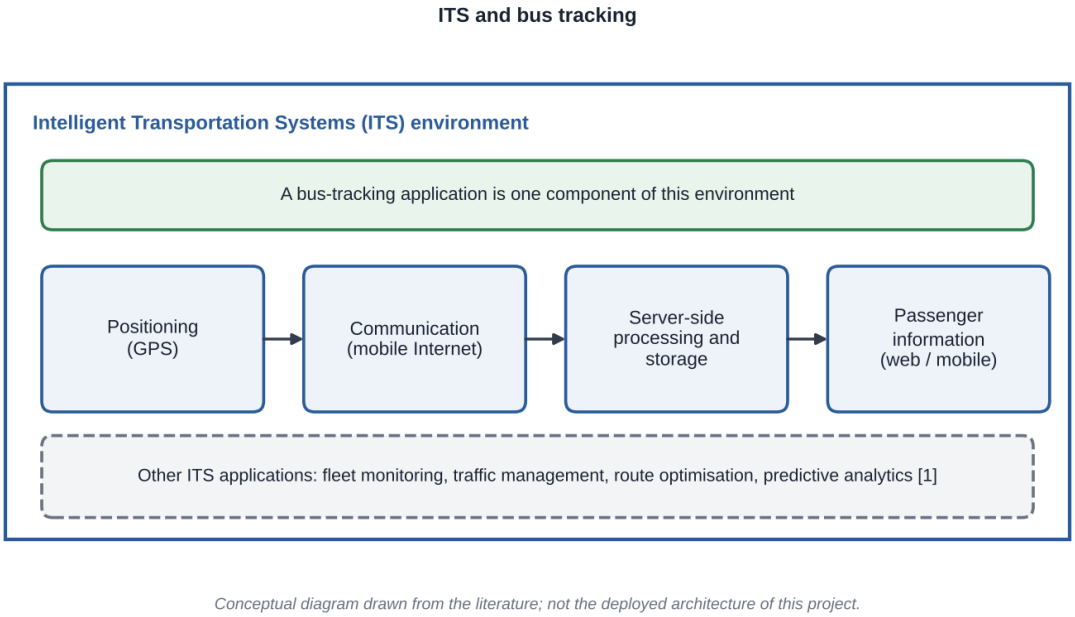


Figure 2.2.1.1: Intelligent Transportation Systems and bus tracking (conceptual)

2.2.2 GPS-Based Vehicle Tracking

GPS-based tracking determines the geographical position of a vehicle and communicates it to a central system. A typical process determines the position of the vehicle, transmits it, stores and processes it on a server, and presents the result to passengers (Figure 2.2.2.1). Khalid, Salleh and Samsudin built a Bus Tracking Information System on this pattern using consumer-grade GPS devices on buses, mobile Internet and a central server [2].

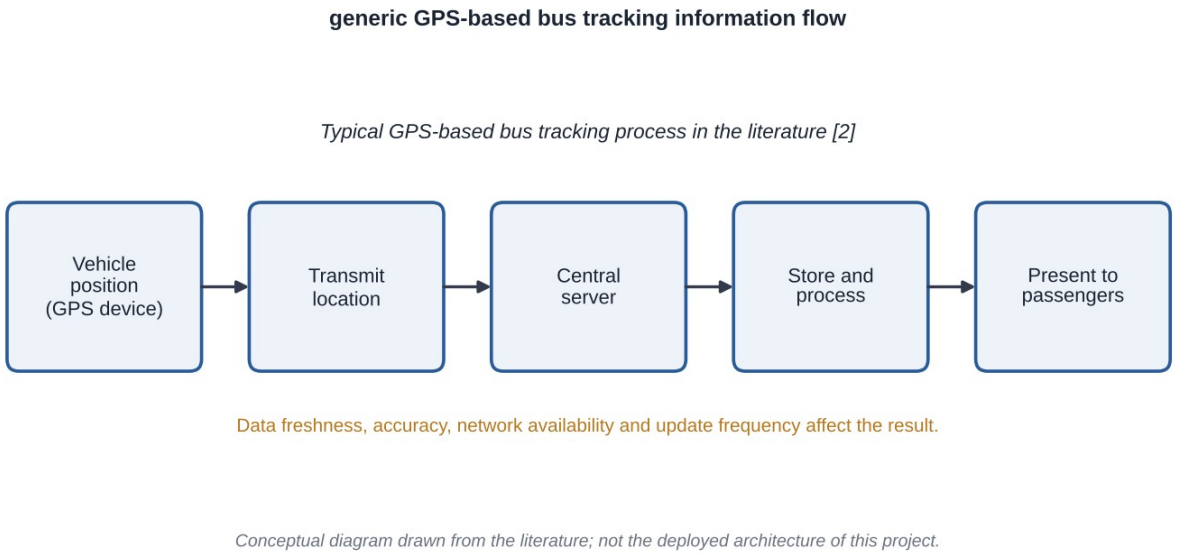


Figure 2.2.2.1: Generic GPS-based bus tracking information flow (conceptual)

The usefulness of tracking depends on the freshness and accuracy of the data. Communication delays, outdated coordinates, positioning error, network availability and update frequency all

influence what a passenger sees. A coordinate also needs context. It becomes useful to a passenger when it is associated with a route, a direction, a stop and ideally an arrival estimate. This is the reason the Bus Tracking System links bus position to structured route and stop data.

2.2.3 Estimated Time of Arrival

Estimated Time of Arrival (ETA) is the expected time remaining before a vehicle reaches a stop. It tells a passenger more than a map position. An ETA can be derived in several ways: from a published schedule, from the vehicle position and an assumed or observed speed, or from a routing service that estimates travel time along a road. Khalid et al. update the arrival information using the bus location relative to the next stop [2]. Whatever the method, an ETA is an estimate rather than a guarantee. Liu and Miller found that real-time information does not always outperform simpler schedule-based strategies, partly because of delays and differences between reported and actual conditions [5]. The Bus Tracking System therefore treats its ETA as a conditional estimate and returns no value when its inputs are missing or invalid.

2.2.4 Real-Time Passenger Information

Real-time passenger information describes the current or expected state of a service. Unlike a static timetable it changes as vehicles move. The GTFS Realtime specification is an established example of how such dynamic information (vehicle positions, trip updates and service alerts) can be represented [6]. Passengers who do not know when the next bus will arrive experience uncertainty, and real-time information aims to reduce it. Watkins et al. studied the OneBusAway system and reported effects on both perceived and actual waiting time [3]. Liu and Miller later showed that the effect depends on how the information is delivered and used [5]. A cautious position follows: real-time information can reduce uncertainty, but a tracking system should not assume that it reduces actual waiting time in every setting (Figure 2.2.4.1).

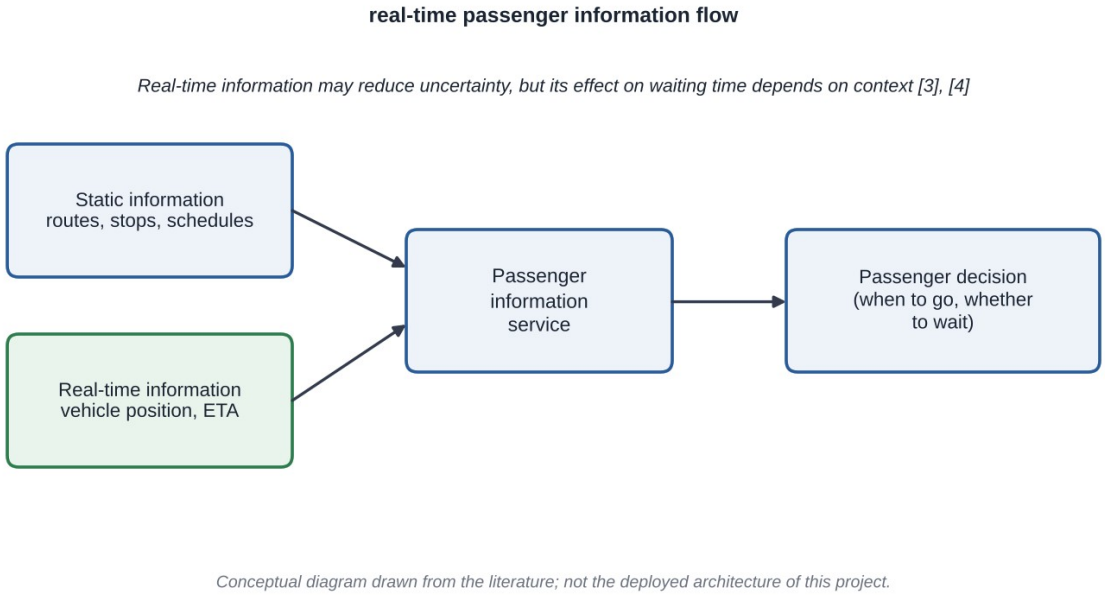


Figure 2.2.4.1: Real-time passenger information flow (conceptual)

2.2.5 Public Transport Data Standards

The General Transit Feed Specification (GTFS) is an open format for public transport information. Its static part describes agencies, routes, stops, trips and stop times in text files. The Bus Tracking System imports four of these files: routes.txt, stops.txt, trips.txt and stop_times.txt, and it also persists the shape points from shapes.txt. GTFS Realtime extends the static model with trip updates, vehicle positions and service alerts [6]. The Bus Tracking System does not produce or consume a GTFS Realtime feed; bus positions arrive directly from the driver's device through the project's own API.

2.2.6 Maps and Road Routing

Bus tracking is geographical, so a passenger benefits from seeing buses, routes and stops on a map. The browser interfaces of the project use Leaflet with OpenStreetMap for this presentation. The system's use of mapping technology is different. It asks the Open Source Routing Machine (OSRM) for the road route between two coordinates. The OSRM route service returns a distance in metres and a duration in seconds for a request made with longitude and latitude pairs. The system uses these results to decide whether a bus is at a stop, whether it has left a stop and how long a bus will take to reach a stop.

2.2.7 Iterative Development and Object-Oriented Analysis

Transport information systems have several interacting parts and their requirements change as prototypes are tested. Iterative and incremental development supports gradual refinement of requirements, design and implementation.

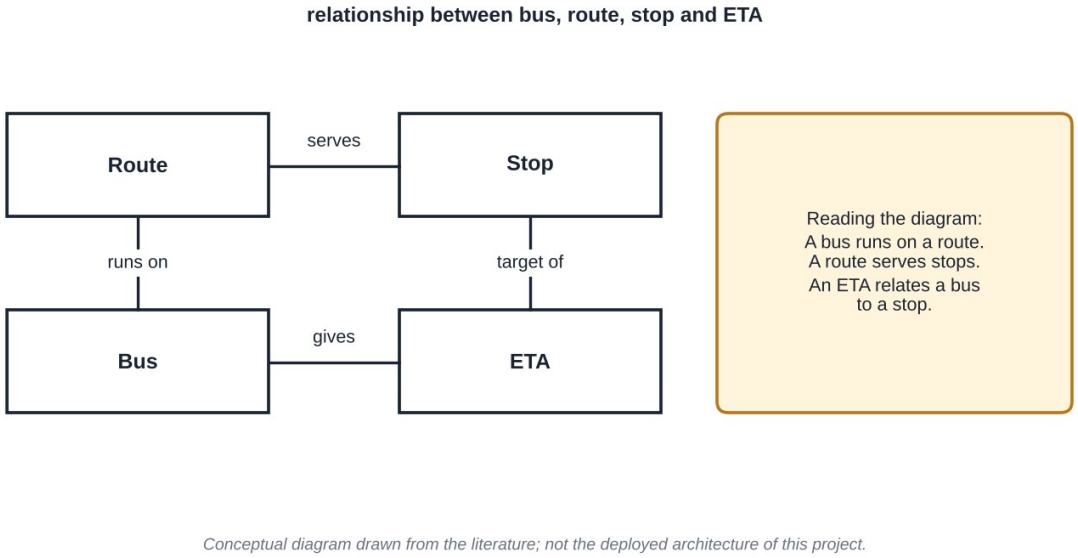


Figure 2.2.7.1: Relationship between bus, route, stop and ETA (conceptual)

2.3 Existing Systems / Related Work

2.3.1 Rea Vaya

Rea Vaya in Johannesburg is an African bus rapid transit (BRT) system that uses structured infrastructure and technology in an urban setting [9]. It is useful as African context. It is not a technical model for this project, because a BRT system has dedicated infrastructure, stations and institutional arrangements that a student prototype does not.

2.3.2 OneBusAway

OneBusAway is a passenger-facing real-time transit information system. Watkins et al. studied its effect on transit riders in the King County Metro service in the United States. It provided next-bus information through several channels, including a website, telephone, text messaging and smartphone applications. Its relevance to this project is conceptual: it shows that tracking becomes more useful when location data is turned into passenger-oriented information. Its findings come from an established U.S. transit environment and should not be assumed to transfer to Lusaka.

2.3.3 A University Bus Tracking Information System

Khalid, Salleh and Samsudin developed a bus tracking information system for a university transport service using consumer-grade GPS and mobile Internet. Android devices on buses transmitted positions to a central server, and passenger clients displayed bus positions and arrival information. The study shows that tracking can be built from accessible technology. Its limitation for the present work is that it addresses a university service, not a public transport network with many routes.

2.3.4 Yango as an Interface Reference

Yango is a ride-hailing application. It was not a published source in this review. It was examined through the prior use of the application by the team members, as described in Section 3.3, and served as a reference for interaction concepts rather than for the transport data model.

2.3.5 Mobile and Browser Access

GSMA Intelligence describes the growth of mobile connectivity and mobile Internet in Sub-Saharan Africa. This supports delivering transport information through the browser of a mobile device, which needs no application installation.

2.4 Comparative Analysis

Table 2.4.1 compares the reviewed systems and standards with the Bus Tracking System on the criteria that matter to this project. The comparison is based on the cited descriptions. It is not an evaluation of the systems.

Table 2.4.1: Comparison of reviewed systems and standards with the Bus Tracking System

System	Position source	Passenger information	Context	Limitation for this project
Rea Vaya	Formal BRT infrastructure	Structured service information	Johannesburg, South Africa	BRT scale and infrastructure
OneBusAway	Transit agency real-time data	Next-bus information by web, phone, text and apps	King County Metro, United States	Established agency environment
University bus tracking system	Consumer-grade GPS on buses, mobile Internet	Bus positions and arrival information	University transport service	Not a multi-route public network
GTFS and GTFS Realtime	Data standards (feeds)	Static schedules; real-time positions, updates and alerts	International	A standard, not a tracking system; needs data feeds
Bus Tracking System (this project)	Real GPS from the driver's device, validated against driver–bus assignment	Routes, destinations, boarding stops, waiting requests, OSRM-based ETA	Prototype for representative Lusaka routes	Prototype; not evaluated in the field

The comparison shows that the basic idea of GPS bus tracking is established. The contribution of this project is not the invention of tracking. It is the integration, in one working system, of driver-authorised real GPS updates, imported GTFS route data, a waiting-request model and a conditional OSRM-based ETA for Lusaka routes.

2.5 Research / Knowledge Gap

- **Geographic.** Much of the cited empirical work originates outside Africa: Watkins et al. and Liu and Miller studied U.S. transit and Khalid et al. developed their system in Malaysia. The ITS review also notes geographical concentration in the field.
- **Contextual.** Reported systems are often tied to a specific organisation, for example a university service. This project considers representative public transport routes in Lusaka.
- **Implementation.** Many deployed systems rely on dedicated GPS equipment and agency data feeds. A student prototype can demonstrate the whole information flow with a GPS-enabled device used by the driver, without claiming an operational deployment.
- **Passenger information.** The effect of real-time information on waiting varies with context. The project therefore concentrates on providing information, not on claiming a behavioural outcome.
- **Interoperability.** GTFS and GTFS Realtime provide standard ways to exchange transport data. The project uses static GTFS for its data and leaves GTFS Realtime as future work.

The gap addressed by this project is therefore a localised implementation gap: a working, documented system that connects real driver-device GPS updates, GTFS-derived route and stop data, current bus state, waiting requests and a conditional ETA for Lusaka routes.

2.6 Conceptual or Theoretical Framework

The conceptual framework in Figure 2.6.1 links the inputs, processing and outputs of the system as suggested by the literature. Transport data (routes, stops and trips) and real GPS location updates enter the system. The system validates who is allowed to act, maintains the state of the bus, stores data, and calls a road-routing service to estimate arrival. The results are presented to passengers by the browser interfaces. The three project features map onto this flow: Real-Time Bus Tracking relies on the GPS updates and stored bus state, Journey Planning relies on the transport data, and Arrival Time Prediction relies on the road-based ETA.

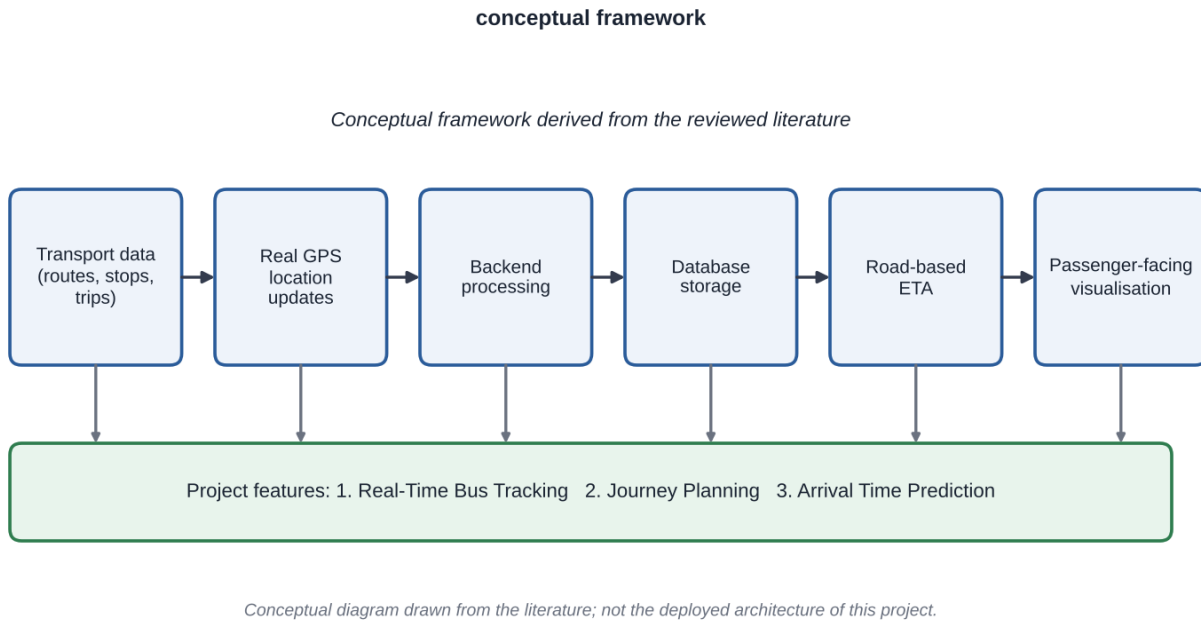


Figure 2.6.1: Conceptual framework derived from the reviewed literature

2.7 Chapter Summary

The literature establishes that GPS can supply vehicle position, that real-time passenger information can influence waiting experience but not in every setting, and that ITS is an established area of work. African examples such as Rea Vaya show that technology is used in formal urban transport. GTFS provides a standard model for transport data. The review supports building a localised prototype, treating ETA as a conditional estimate, and being cautious about claims on waiting time.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This chapter describes the methodology used to develop the Bus Tracking System for public transport in Lusaka. It explains the project's research design, the iterative and incremental approach used to design and implement the system, the architecture and principal components of the implemented solution, the database design, the user-interface design, and the software tools and technologies used.

The development process combined findings from the literature review with practical analysis of existing public-transport applications. The Rea Vaya bus system was considered during the literature review because it demonstrates the use of structured public-transport services in an urban context. The Yango application was also analysed through the team members' prior use of the application. Its interface and interaction patterns informed notes on potentially relevant functions, including location-based information, route and destination selection, vehicle information and passenger interaction.

The implemented system developed beyond the original proposal. Early work included simulated vehicle movement, but supervisor feedback led the team to replace simulation with real GPS coordinates obtained from a driver's GPS-enabled device. The final backend uses Django and Django REST Framework, GTFS-derived transport data, a PostgreSQL database, browser-based interfaces, OpenStreetMap and OSRM-supported road-distance processing.

3.2 Research / Development Approach

3.2.1 Iterative and Incremental Development

The project used an iterative and incremental approach. In an iterative process a limited subset of the requirements is designed, built and tested in each cycle, and early feedback is used to adapt later cycles. The approach suited this project because the system has several connected parts (data model, driver operations, GPS handling, waiting requests and arrival estimation) whose requirements became clearer as earlier parts were exercised.

The proposal had planned Agile Scrum with two-week sprints. In practice the group worked in iterations and reviewed progress with the supervisor roughly every two weeks, in person and online. The project did not keep formal sprint artefacts, so this report does not describe the process as Scrum. It describes it as iterative and incremental development with regular supervisor review.

Figure 3.2.1.1 shows the sequence of stages. The stages were revisited as new information arrived. The prototype stage that used simulated GPS data is marked, because supervisor feedback led the group to replace it with real GPS updates (Stage 5). The simulation is not part of the final system.

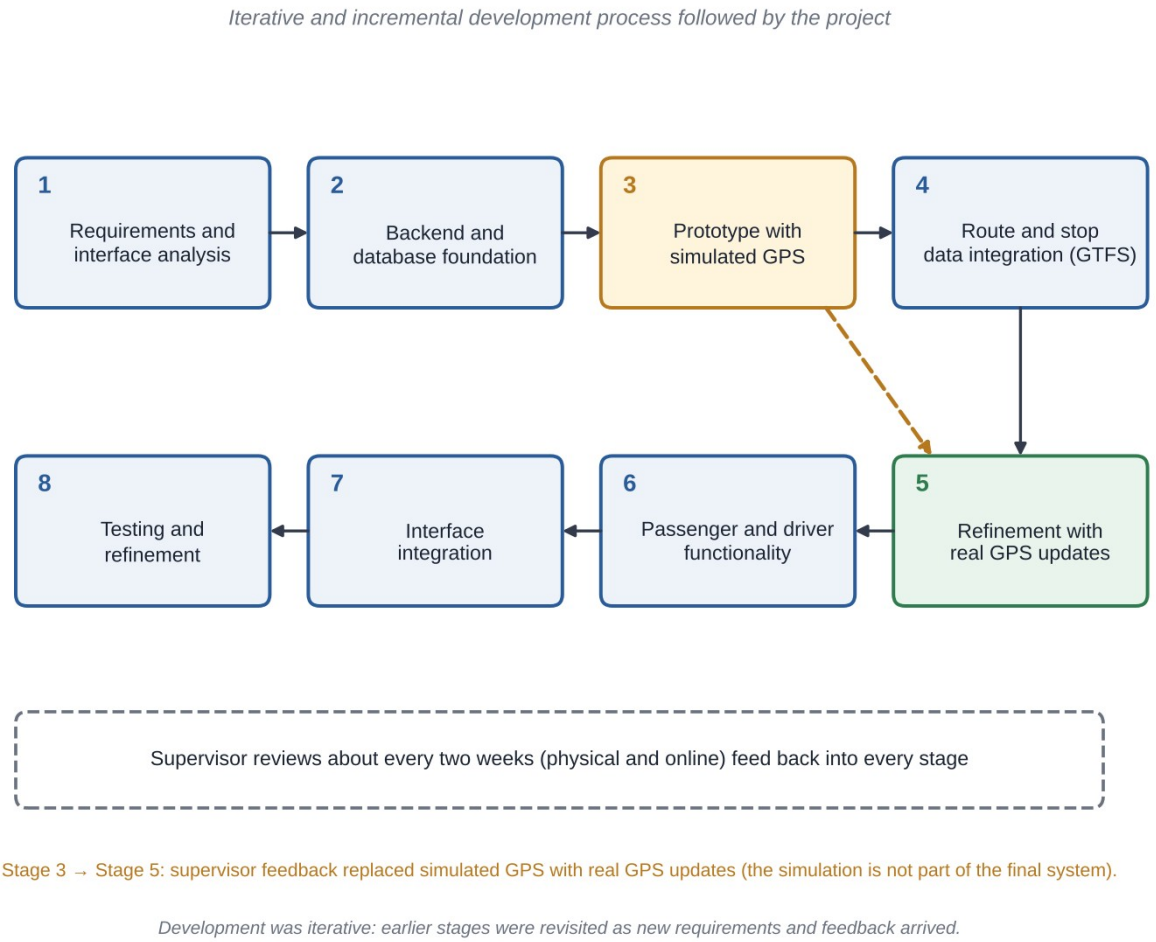


Figure 3.2.1.1: Iterative and incremental development process of the Bus Tracking System

3.2.2 Work in Each Stage

Table 3.2.2.1 summarises the work done in each stage. The stage descriptions follow the group's own account of its process.

Table 3.2.2.1: Stages of development and the work done in each stage

No.	Stage	Work in the stage
1	Requirements and interface analysis	Shared requirements analysis, including the study of the interaction concepts of the Yango application, from which the system's responsibilities were derived
2	Server and database foundation	Django project, initial models and migrations, REST endpoints for routes, stops and buses
3	Prototype with simulated GPS	Not part of the final system; replaced in Stage 5
4	Route and stop data integration (GTFS)	Models for Route, Stop, Trip, StopTime and Shape; import commands that load the GTFS files

No.	Stage	Work in the stage
5	Refinement with real GPS updates	Authenticated GPS update endpoint, bus state, status and next-stop logic, and a driver interface that sends the device's location, replacing the simulation
6	Passenger and driver functionality	Passenger and driver accounts, driver-to-bus assignment checks, waiting requests and waiting counts, and the passenger and driver screens that use them
7	Interface integration	The browser interfaces connected to the API endpoints
8	Testing and refinement	Automated tests of the server, manual checks of the deployed flows, and the deployment configuration

3.2.3 Evidence of Iteration in the Code

The migration history of the integration branch records some of the iterations. Migrations 0001 to 0009 show the data model being extended step by step: a Shape model and a bus status field were added in the second migration; a waiting request gained a link to a trip in the third and lost its earlier link to a route in the fourth; passengers were linked to Django users in the sixth and seventh; and the Driver and DriverBusAssignment models arrived in the eighth, followed by the removal of a driver name field in the ninth. The sequence shows the waiting-request design moving from a route-based to a trip-based model, which is the form documented in Chapter 4.

3.2.4 Proposal Compared with the Implemented System

The final system differs from the proposal in several respects. Table 3.2.4.1 records the differences so that the report does not describe the proposal as if it had been delivered.

Table 3.2.4.1: Proposal compared with the implemented system

Aspect	Proposal	Implemented system (integration branch)
Bus position source	Simulated GPS generated along routes	Real GPS latitude, longitude and speed sent by an authorised driver's device
Server framework	Django REST Framework	Django and Django REST Framework
Database	MySQL and Firebase	PostgreSQL, with its address read from DATABASE_URL
Passenger interface	React with Google Maps API	Browser interfaces using HTML, CSS and JavaScript with Leaflet and OpenStreetMap
Real-time delivery	Firebase for real-time data	Standard HTTP requests; no WebSockets
Arrival estimate	ETA within ±2 minutes of expected	ETA from OSRM road duration; accuracy not evaluated
Operator dashboard	Admin dashboard for operators	Django administration site for records; no separate dashboard claimed

Aspect	Proposal	Implemented system (integration branch)
Process	Agile Scrum, two-week sprints	Iterative and incremental development with supervisor reviews about every two weeks
Evaluation	Unit tests, UAT with five or more users, SUS of at least 70, 10+ simulated buses, load times	Automated tests of the server and manual functional checks of the deployed flows; no UAT, SUS or performance results

3.3 Requirements / Data Collection

3.3.1 Requirements Analysis

Requirements analysis started from the transport problem described in Chapter 1 and from the interface analysis described below. The team did not conduct questionnaires, interviews, focus groups or any other direct requirements-gathering exercise with passengers, bus drivers or transport operators. The requirements in this report were therefore derived from the analysis of the transport problem, the examination of Yango, the review of related literature and systems, and technical refinement during implementation. They were treated as revisable. The change from simulated to real GPS data is the clearest case of a requirement revised after supervisor feedback.

3.3.2 User Interface Analysis of the Yango Application

A major source of functional and interface requirements was the analysis of the Yango application. Yango was the only existing transport application examined directly by the development team for this purpose. All members of the team had previously used the application as passengers, which gave them familiarity with its general workflow. During requirements analysis the team recorded notes on relevant features so that suitable concepts could be considered in the design of the Bus Tracking System.

The analysis concentrated on how a location-based transport application presents information and guides a user through a transport workflow. The features examined were the display of vehicle locations, the selection of destinations, the identification of pickup locations, the presentation of vehicle information, and the interaction between passengers and drivers.

3.3.3 Adapting Ride-Hailing Ideas to Predefined Bus Routes

The aim was never to reproduce Yango as a ride-hailing service. A ride-hailing passenger requests an individually assigned vehicle to travel between two freely chosen points. A bus passenger instead joins a service that follows a predefined route and stops at predefined stops. The interaction ideas were therefore adapted as follows.

Table 3.3.3.1: Adaptation of ride-hailing interaction ideas to a public bus service

Ride-hailing concept observed	Adaptation for public buses	Support in the system
Vehicle locations on a map	Active buses shown on a map	Bus records with current latitude, longitude, speed and status
Choosing a destination	Choosing a route and then one of its destinations	Route search and destination endpoints
Choosing a pickup point	Choosing a boarding stop from the ordered stops of a trip	Trip-stops endpoint using imported stop times
Requesting a ride	Creating a waiting request at a boarding stop	WaitingRequest model and lifecycle
Vehicle information and arrival time	Bus status and an arrival estimate for the waiting stop	Status logic and OSRM-based ETA
Driver side of the interaction	Driver activates a bus, starts a trip and sends GPS updates	Driver authorisation and trip, location endpoints

The Yango user interface analysis therefore influenced the functional requirements and the interaction concepts of the interface. It did not influence the underlying public transport data model, which comes from GTFS.

3.3.4 Related Systems

The interface analysis was supplemented by the review of related systems in Chapter 2. Rea Vaya was treated as a contextual reference for structured public transport in an urban African setting, not as a system to reproduce. Its operational and technological model is much larger than what the project could realistically implement.

3.3.5 Transport Data

Transport data came from a GTFS feed stored in the project's gtfs directory. The feed's own metadata names DigitalTransport4Africa as the publisher, gives the feed version as lusaka_city_pt, and lists three agency categories: minibuses of 14 to 18 seats, big buses and shared taxis. The system uses four of the files: routes.txt, stops.txt, trips.txt and stop_times.txt. The shapes.txt file is imported into a Shape table but is not used in route selection, trip progression or ETA. Counts computed directly from the files in the repository are given in Table 3.3.5.1.

Table 3.3.5.1: Contents of the GTFS files stored in the repository

Item	Count or observation
Routes (routes.txt)	68 routes: 55 minibus, 10 big bus and 3 shared taxi
Stops (stops.txt)	793 rows; one row has no coordinates and is skipped by the import
Trips (trips.txt)	408 trips: 136 for each of three start times (08:00:00, 11:00:00 and 15:00:00); 204 in each direction
Stop times (stop_times.txt)	9,219 rows covering 793 distinct stops

Item	Count or observation
Shape points (shapes.txt)	13,431 points forming 136 shapes (imported; not used in route selection or ETA)
Representative trips	The 136 trips whose identifier ends in -08:00:00 (68 routes, two directions); they average about 23 stops, from 2 to 38

The three start-time variants of each trip carry the same stop sequence. The system therefore uses the 08:00:00 variant as a representative trip for each route and direction, both when a passenger plans a journey and when a driver starts a trip. A check of the files shows that the destinations reached by these 136 trips are the same as those of all 408 trips, so no destination is lost by using them. The consequence is that the system models route and direction, not a time-of-day schedule.

3.4 System Architecture / Research Workflow

The system follows a client–server architecture. Browser interfaces send HTTP requests with JSON to the Django backend, which reads and writes its database and calls an external OSRM service for road routes. The architecture is described in Section 4.6 with its diagram, so it is not repeated here. In methodological terms the workflow was: analyse the problem and interface concepts; build the data foundation; import the GTFS data; replace the simulated movement with authorised real GPS updates; add passenger and driver functions; integrate with the interfaces; and test and refine. Figure 3.2.1.1 shows this sequence.

3.5 Tools and Technologies

Table 3.5.1 lists the principal technologies of the whole system, server and browser interfaces. The server versions are those pinned in requirements.txt of the integration branch; the Leaflet version is the one referenced by the page templates.

Table 3.5.1: Principal tools and technologies

Tool	Version	Use in the system
Python	—	Implementation language
Django [12]	6.0.6	Web framework, ORM, migrations, authentication, administration site, password-reset views
Django REST Framework	3.17.1	Serialisers and API views
psycopg[binary]	3.3.6	PostgreSQL driver used when DATABASE_URL is set
dj-database-url	3.1.2	Reads DATABASE_URL into the Django database setting
WhiteNoise	6.12.0	Serving static files
gunicorn	26.2.0	WSGI server (the WSGI application is bus_tracking.wsgi.application)
requests	2.34.0	HTTP calls to OSRM
django-anymail[brevo]	15.2	Sending password-reset e-mail through Brevo

Tool	Version	Use in the system
PostgreSQL	—	Database of the deployed system; Django connects to it through psycopg using the address in DATABASE_URL
OSRM [8]	—	Road distance and duration, configured through OSRM_URL
HTML, CSS and JavaScript	—	Passenger interface, driver dashboard and landing page, written as Django templates with inline styles and scripts and no front-end framework; the scripts call the API with the browser's Fetch API and send the CSRF token
Leaflet	1.9.4	Interactive map library for the passenger map and the driver's location map, loaded from the unpkg content-delivery network
OpenStreetMap tiles	—	Map background tiles requested by Leaflet in the browser
Browser Geolocation API	—	`navigator.geolocation.watchPosition` supplies the driver device's position, which the driver dashboard sends to the server
Git and GitHub	—	Source control; the integration branch is the authoritative branch for this report
Render and Brevo	—	Hosting of the web application and e-mail delivery for password reset

3.6 Ethical and Legal Considerations

No participants were recruited, so no consent procedure, questionnaire or interview data was involved. The system does store personal data: a username, an e-mail address and a hashed password for each account, and a display name for passengers. It also stores the position and speed sent from a driver's device for a bus. The following considerations apply.

- **Credentials and secrets.** Passwords are stored by Django's authentication system as hashes. The secret key, database address, e-mail service key and routing address are read from environment variables, not written in the source. Debugging is off unless the debug flag is explicitly set to true, and when it is off the settings require the secret key to be set.
- **Access control.** Driver operations check driver and assignment status. A passenger's waiting-request endpoints are scoped to that passenger. Section 6.6 records the remaining security limitations.
- **Location data.** The stored location is that of a bus, supplied by its driver, and only the latest value is kept; there is no history table. The screenshots in Chapter 6 show the location reported by the device used for the manual checks, which was a test device and not an operating bus.
- **Data licensing.** The GTFS feed names its publisher in its metadata. This report does not examine its licence terms, which should be checked before any redistribution or operational use.

- **Third-party software.** The libraries listed in Table 3.5.1 are used under their own licences, and the sources in Chapter 2 are cited.

3.7 Evaluation Method

Evaluation was limited to functional verification. The repository contains automated tests, written with Django's test framework, for the arrival-time and waiting-overview logic, access control, registration, login throttling and the debug default. The command `python manage.py test` runs them [13]. The separate command `python manage.py check` only validates the project configuration and is not a test run. The passenger interface, the driver dashboard and the server were also checked together by hand on the deployed system (Section 6.3). The method does not include user acceptance testing, usability scoring, field trials, load tests or an accuracy comparison of ETA values with observed arrival times, because none was carried out.

3.8 Chapter Summary

The methodology combined requirements analysis, an interface analysis of Yango adapted to predefined bus routes, a review of related systems, iterative development with supervisor review, and verification by automated tests. The supervisor's feedback replaced simulated GPS with real GPS. The next chapter sets out the analysis and design of the system.

CHAPTER 4: SYSTEM ANALYSIS AND DESIGN

4.1 Introduction

This chapter presents the analysis and design of the Bus Tracking System as a whole, with the detailed design concentrated on the server-side modules and data. Table 4.1.1 states the purpose of each artefact and where it appears.

Table 4.1.1: Analysis and design artefacts used in this report

Artefact	What it shows	Level	Section
Use-case diagram	Goals of the human actors and the external system, and what the system offers them	Requirements	4.5
System sequence diagrams (SSDs)	For one scenario, the external events from an actor to the system treated as a black box, and its observable responses	Analysis	4.5
Architecture diagram	The parts of the implemented system and what it communicates with	Design	4.6
Domain model	Meaningful concepts of the problem domain and the associations between them, with no implementation detail	Analysis (conceptual)	4.7.1
Entity-relationship (ER) diagram	The persisted tables, keys and relationships actually implemented	Design (data)	4.7.2

Artefact	What it shows	Level	Section
Design model	Design classes (here, modules), their operations, dependencies and interactions that realise the system operations	Design (software)	4.9

4.2 Stakeholder and User Analysis

Table 4.2.1 identifies the people and external systems that interact with, or are affected by, the system. The requirements analysis relied on the problem analysis and interface study described in Chapter 3. No stakeholder interviews were conducted.

Table 4.2.1: Stakeholders and their needs

Stakeholder	Need	How the system serves it
Passenger	Find a route, choose a destination and boarding stop, register that they are waiting, and know when the bus will arrive	Route, destination and stop endpoints; waiting requests; ETA
Driver	Sign in, operate only the bus they are assigned, start a trip and report position	Driver login; assignment check; trip start; GPS update; waiting counts
Administrator	Maintain buses, drivers, assignments and transport records	Django administration site; import commands

4.3 Functional Requirements

The functional requirements are grouped under the three features of the system, in the order Real-Time Bus Tracking, Journey Planning and Arrival Time Prediction. Next-stop determination is not listed as a feature, because passengers never use it directly; it is internal logic that supports Real-Time Bus Tracking and Arrival Time Prediction. Table 4.3.1 lists the requirements implemented by the system.

Table 4.3.1: Functional requirements implemented by the system

ID	Feature	Requirement
FR1	Real-Time Bus Tracking	Accept latitude, longitude and speed from an authenticated driver for a bus to which the driver is actively assigned
FR2	Real-Time Bus Tracking	Store the current coordinates, speed, status and last-update time of each bus
FR3	Real-Time Bus Tracking	Derive the bus status (IDLE, IN_TRANSIT or AT_STOP) from the position relative to the current stop
FR4	Real-Time Bus Tracking	Keep the current trip and current stop of the bus and advance to the next stop when the bus has left the current one
FR5	Real-Time Bus Tracking	Allow an authorised driver to activate or deactivate a bus, to reset its operational state and to start a trip for a chosen route and destination
FR6	Real-Time Bus Tracking	Provide bus records (registration number, status, position, speed, last update) to clients

ID	Feature	Requirement
FR7	Driver access	Authenticate drivers through session login and refuse accounts that are not driver accounts or that are inactive
FR8	Driver access	Refuse any bus operation or GPS update unless there is an active driver-to-bus assignment (HTTP 403 otherwise)
FR9	Driver access	List the buses actively assigned to the signed-in driver
FR10	Journey Planning	Search routes by name
FR11	Journey Planning	List the destinations of a route, and the ordered stops of the trip for a route and destination
FR12	Journey Planning	Register, sign in and sign out passengers
FR13	Journey Planning	Create a waiting request for a trip and a stop served by that trip; allow one active request (WAITING or ON_BOARD) per passenger
FR14	Journey Planning	Support the waiting-request lifecycle: board, cancel, complete and expire
FR15	Journey Planning	Return the number of waiting passengers for a trip and stop, and for each stop of a bus's current trip
FR16	Arrival Time Prediction	Return an ETA in minutes for a passenger's own WAITING request, or null when no valid estimate exists
FR17	Arrival Time Prediction	Return the road-based duration from a given bus to a given stop
FR18	Data	Import routes, stops, trips, stop times and shapes from GTFS files repeatably
FR19	Account support	Support password reset by e-mail

4.4 Non-Functional Requirements

Table 4.4.1: Non-functional requirements and design response

Quality	Requirement	Design response and evidence
Security	Driver operations must be restricted to authorised drivers; passenger request data must be scoped to its owner	Session authentication; get_authorized_bus_for_driver; waiting-request queryset filtered by the signed-in passenger. Gaps are listed in Section 6.6
Reliability	Failure of the routing service must not crash a GPS update or produce a false estimate	Services catch routing request failures; ETA returns null when no valid duration exists
Data integrity	Prevent duplicate assignments, stop sequences and shape points	Unique constraints on (driver, bus), (trip, stop_sequence) and (shape_id, shape_pt_sequence); atomic transactions for bus state changes
Maintainability	Business rules separated from request handling	Function-based service modules; models for operational state; management commands for imports
Portability and deployability	Run locally and on a hosting platform with environment-based settings	DATABASE_URL, DJANGO_SECRET_KEY and other settings read from the environment; WSGI application bus_tracking.wsgi.application

Quality	Requirement	Design response and evidence
Simplicity of transport	Work over standard web infrastructure	Plain HTTP requests and JSON; no WebSockets
Performance	No target claimed	A GPS update can call OSRM up to twice (Section 4.9); not measured

4.5 Use Cases / User Stories

4.5.1 Use-Case Diagram

Figure 4.5.1.1 shows the use cases of the system. The human actors, Passenger and Driver, are on the left. The OSRM routing service is a supporting external system outside the system boundary, on the right. Only goals understandable to the users are shown as use cases. The use case *Retrieve ETA* includes *Calculate ETA*, which in turn includes *Obtain Road Duration*; the last step is performed by OSRM. The diagram describes the system as a whole.

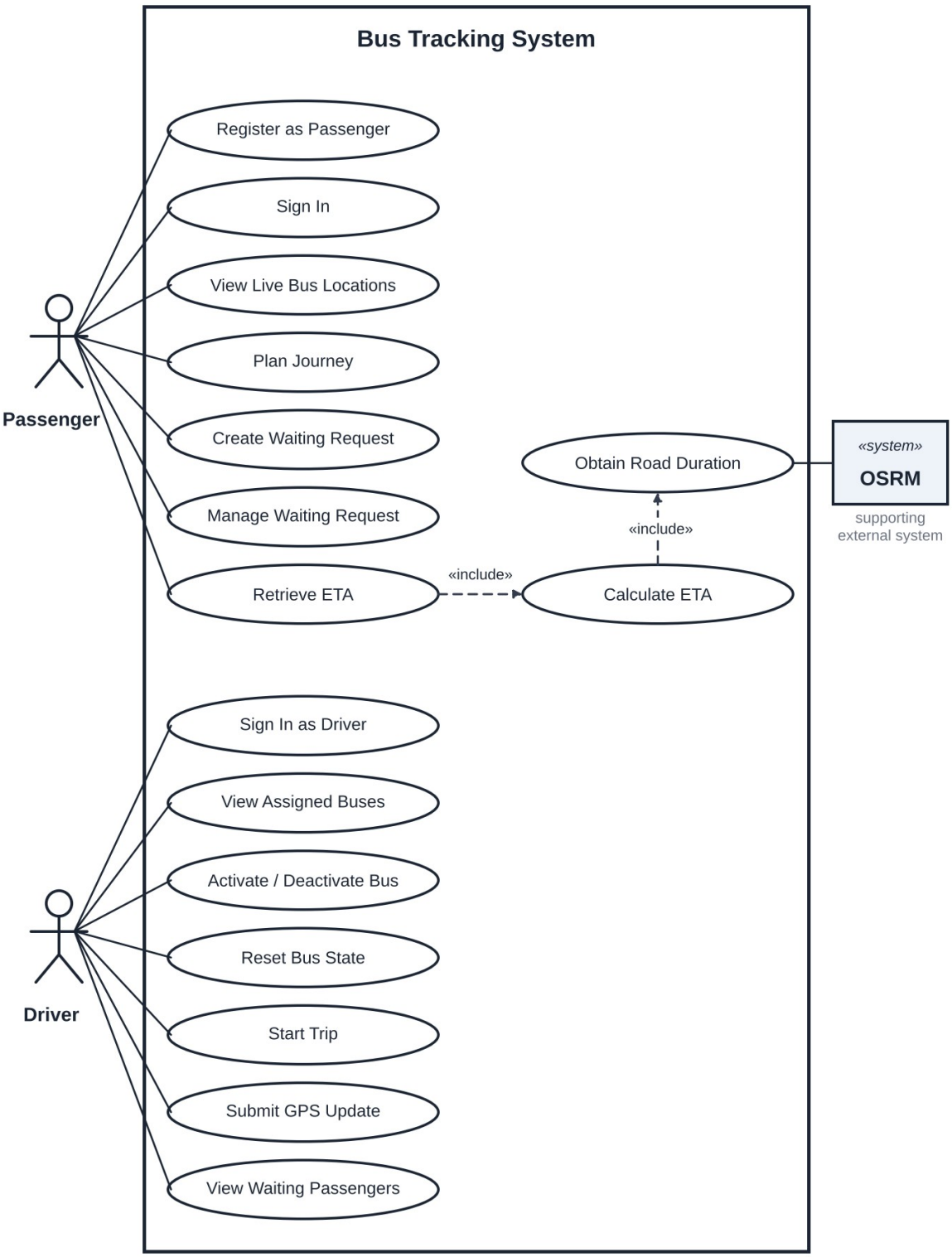


Figure 4.5.1.1: UML use-case diagram of the Bus Tracking System

4.5.2 Use-Case Summaries

Table 4.5.2.1 gives brief descriptions of the use cases that exercise the core logic of the system.

Table 4.5.2.1: Brief descriptions of the principal use cases

Use case	Actor	Description
Start Trip	Driver	The driver chooses a route and destination for an activated bus. The system checks that the driver is active and assigned, selects the trip, sets the bus's current trip and first stop and reports the trip started
Submit GPS Update	Driver	While a trip is in progress, the driver's device sends latitude, longitude and speed. The system stores them, updates the next stop and status and returns the current state
Plan Journey	Passenger	The passenger searches a route, selects a destination and sees the ordered stops of the trip so that a boarding stop can be chosen
Create Waiting Request	Passenger	The passenger registers that they are waiting at a boarding stop of a trip. The system allows only one active request per passenger
Manage Waiting Request	Passenger	The passenger marks the request as boarded, cancels it, or completes the trip
Retrieve ETA	Passenger	For a WAITING request, the system calculates the arrival time of the nearest eligible bus using the road duration from OSRM, or reports that no estimate is available
View Waiting Passengers	Driver	The system returns the number of waiting passengers for each stop of the bus's current trip

4.5.3 System Sequence Diagrams

A system sequence diagram (SSD) shows, for one scenario of a use case, the events that external actors generate and the system operations they trigger, with the system treated as a black box. An event is something that happens and requires a system response, and events trigger system operations. The SSDs therefore show only external actors, the events they cause and the observable responses. They do not show database tables, modules or internal calls, which belong to the design model in Section 4.9. Operation names are given at the level of intent and are derived from the endpoints of the implementation (Table 4.5.3.1).

Two separate SSDs are given. Figure 4.5.3.1 covers the scenario in which an authorised driver starts a trip and updates the GPS position of the bus. OSRM does not appear in it, because the driver is not an actor of the routing exchange; it is an internal dependency of the system. Figure 4.5.3.2 covers the scenario in which a passenger plans a journey, creates a waiting request and retrieves the ETA. OSRM appears there because it provides the road duration that determines the response.

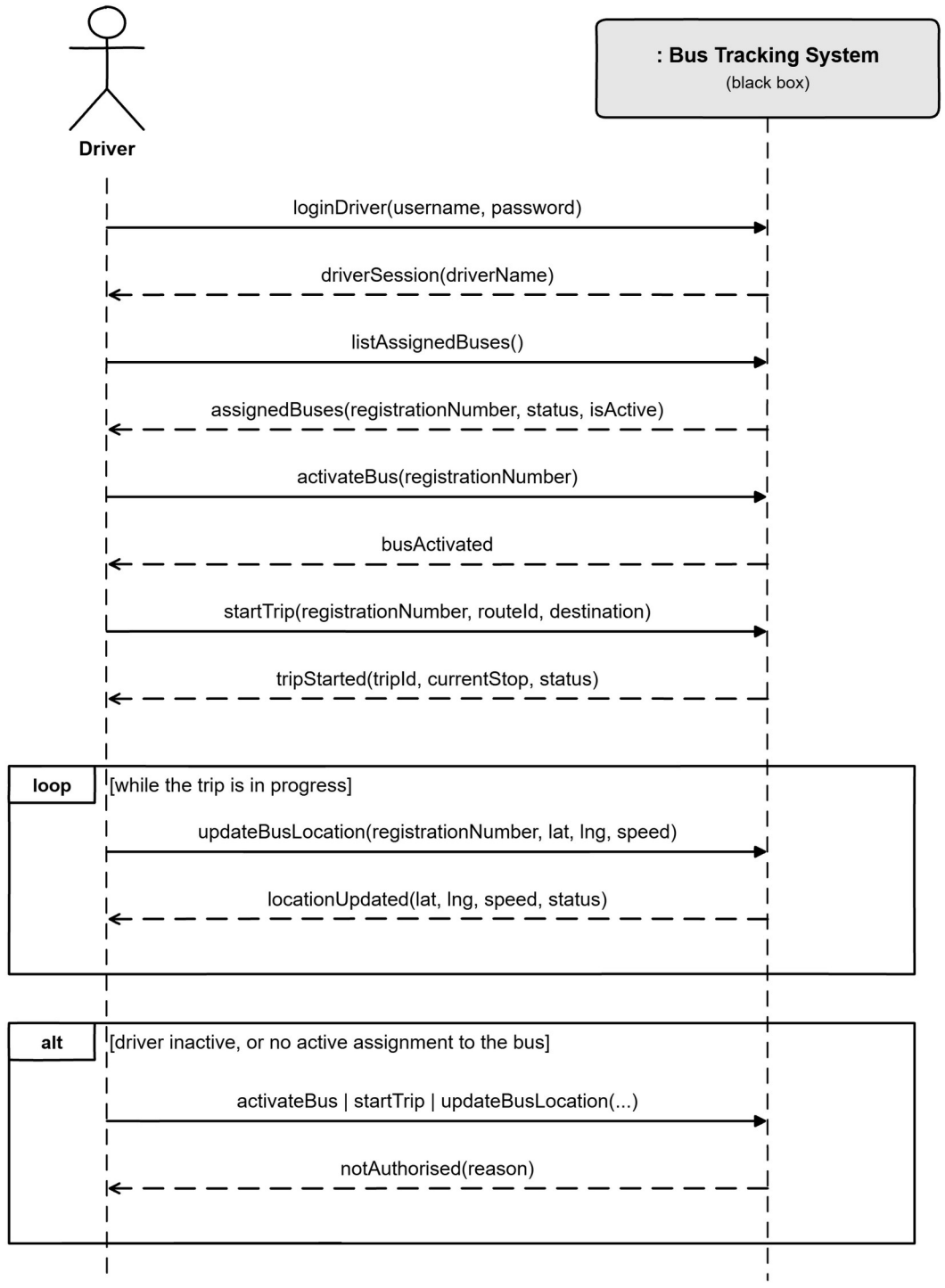


Figure 4.5.3.1: System sequence diagram: authorised driver starts a trip and updates the bus GPS position

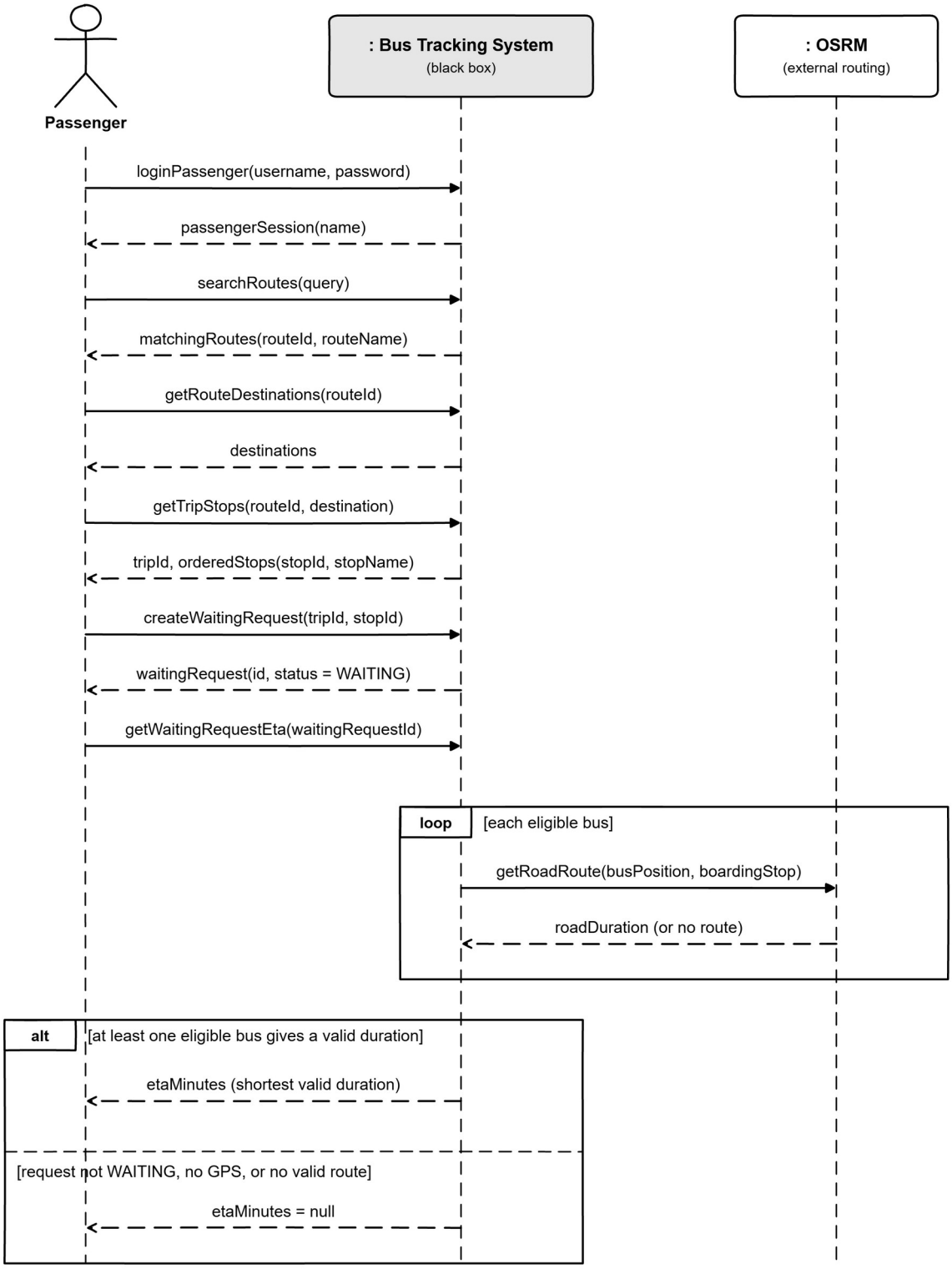


Figure 4.5.3.2: System sequence diagram: passenger plans a journey, creates a waiting request and retrieves the ETA

Table 4.5.3.1: System operations and the endpoints that implement them

System operation	Method	Endpoint (under /api/)
loginDriver	POST	auth/driver/login/
listAssignedBuses	GET	auth/driver/buses/
activateBus	POST	buses/<registration_number>/activate/
startTrip	POST	buses/start-trip/
updateBusLocation	POST	buses/<registration_number>/location/
loginPassenger	POST	auth/login/
searchRoutes	GET	routes/search/?q=
getRouteDestinations	GET	routes/<route_id>/destinations/
getTripStops	GET	routes/<route_id>/stops/?destination=
createWaitingRequest	POST	waiting-requests/
getWaitingRequestEta	GET	waiting-requests/<id>/eta/

4.6 System / Solution Architecture

Figure 4.6.1 shows the architecture of the implemented system. The browser clients, which belong to the interface developers, communicate with the Django project over HTTP using JSON and a session cookie. The Django project `bus_tracking` handles URL routing, settings, session authentication and static files through WhiteNoise, and is served by gunicorn through the WSGI application. Inside it, the `tracking` app contains the API views and serialisers, an authorisation helper, the service modules, the models, and the management commands that import GTFS data. Data is stored in a PostgreSQL database whose address is given by DATABASE_URL. The GTFS text files are read only by the import commands. OSRM is an external service reached through HTTP. Password-reset e-mail is sent through Brevo using the Django built-in password-reset views.

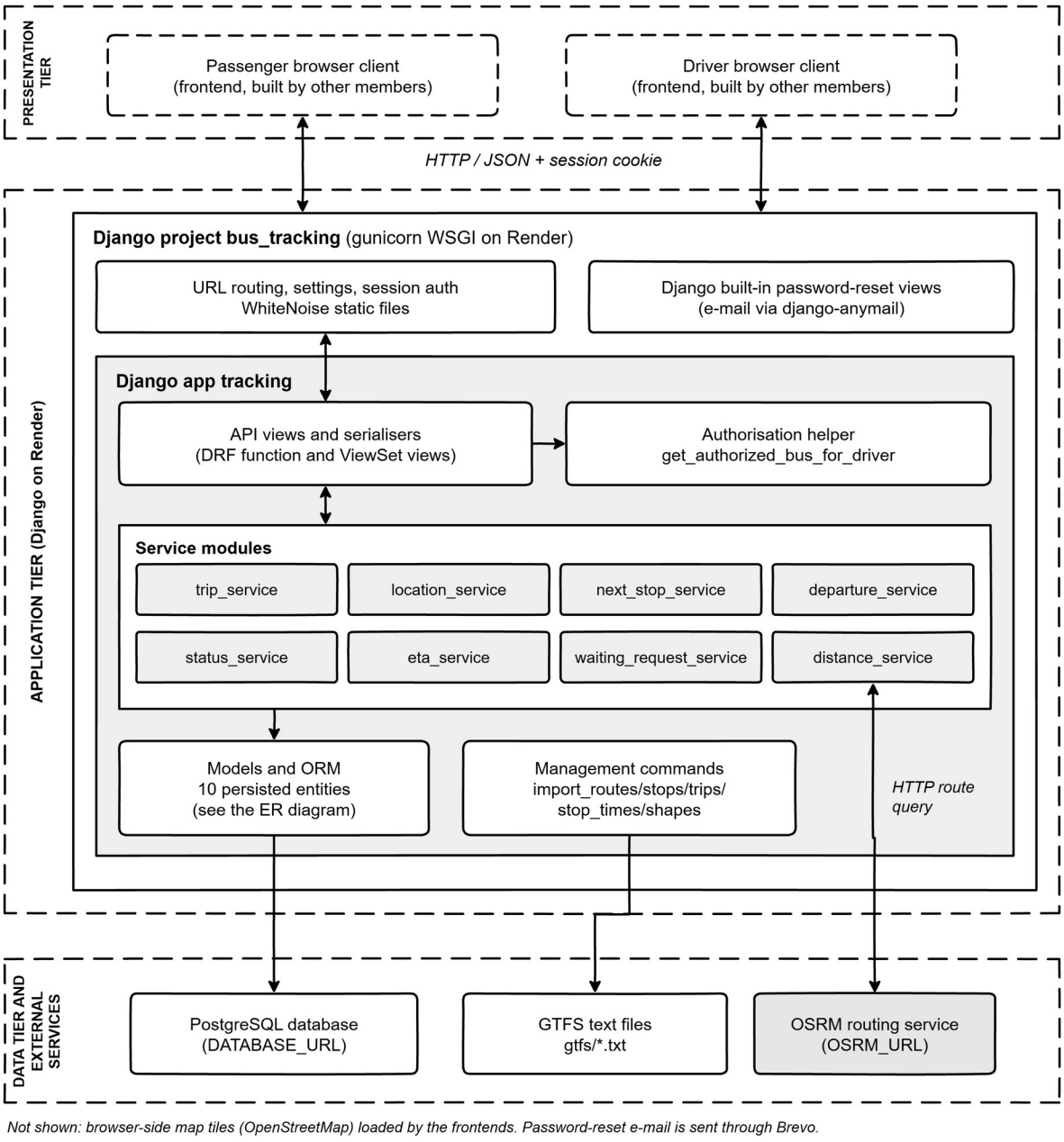


Figure 4.6.1: System architecture of the Bus Tracking System

The system is a client–server application with a three-tier (n-tier) structure, which the figure marks with its three bands. The presentation tier is the passenger interface and the driver dashboard, which run in the user's browser and are delivered as pages by the server. The application tier is the Django project, in which the API views call service modules that use the models and the ORM. The data tier is the PostgreSQL database, together with the GTFS files from which the data is imported. The system also calls an external road-routing service, OSRM. The layering inside the application tier is not strict: some views, such as the read-only route, stop and bus endpoints, query the models directly (Section 4.9.1).

The service modules separate business rules from request handling: `trip_service` selects and starts trips, `location_service` stores GPS updates, `next_stop_service` and `departure_service` keep the stop state, `status_service` derives bus status, `distance_service` calls OSRM, `eta_service` estimates arrival, and `waiting_request_service` implements the waiting-request lifecycle and counts. A further module, `route_service`, defines a function that is not used by the current code.

The environment variables read by the application are listed in Appendix F.

4.7 Database / Data Design

4.7.1 Domain Model

The domain model in Figure 4.7.1.1 is a conceptual model. Following Larman's notation it shows conceptual classes as boxes with names only, associations named in the form *TypeName–VerbPhrase–TypeName*, and multiplicities [4]. It includes need-to-know associations, that is, those whose memory the system must preserve, and omits associations that would only aid comprehension [4]. It deliberately contains no keys, field types, serialisers, services, APIs or framework tables, which belong to the ER diagram and the design model.

Conceptual classes and associations in Larman notation (no keys, fields or framework classes)

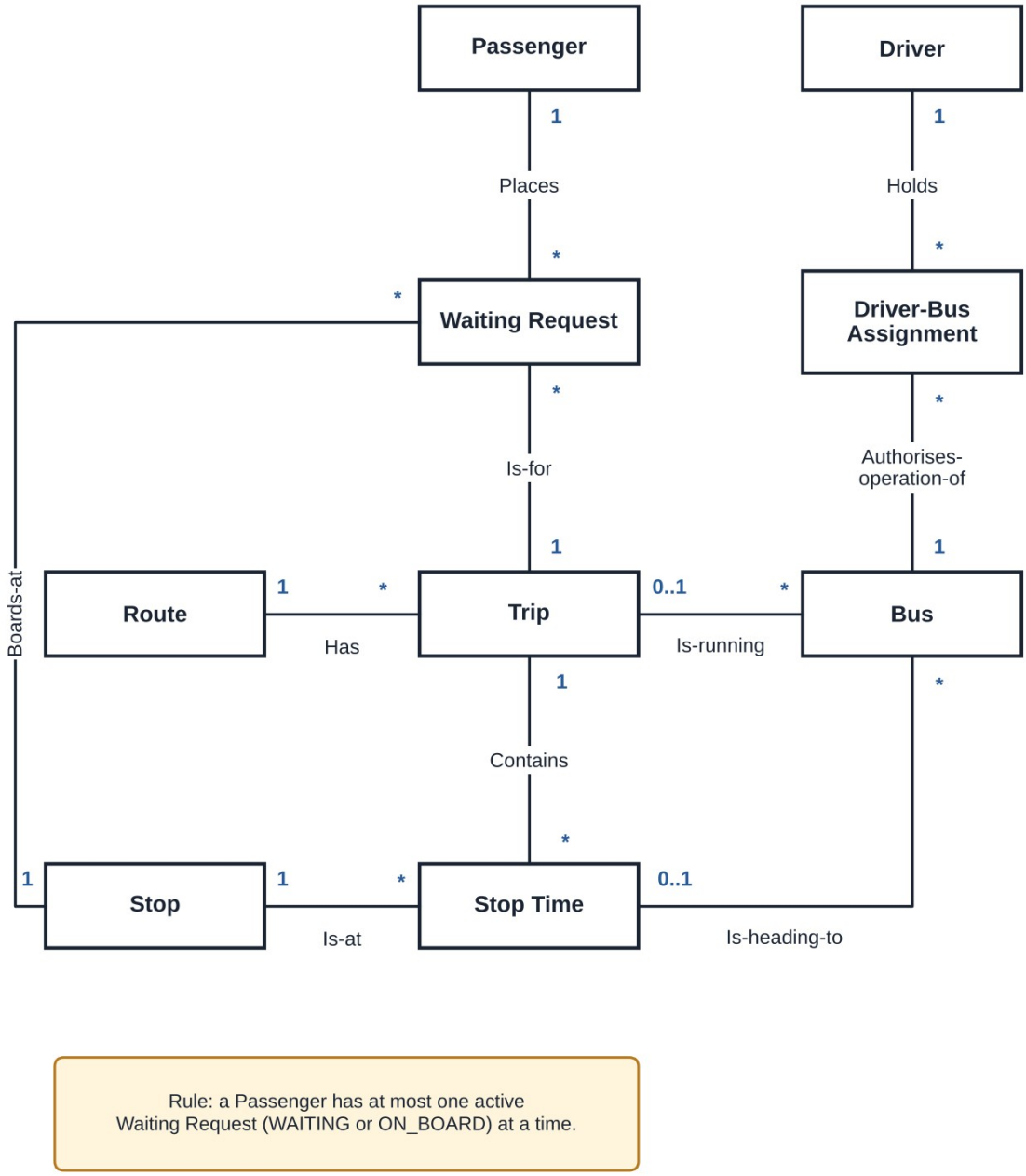


Figure 4.7.1.1: Domain model of the Bus Tracking System (conceptual)

A *Passenger* places *Waiting Requests*, each of which is for one *Trip* and boards at one *Stop*. A *Route* has *Trips*, and a trip contains an ordered sequence of *Stop Times*, each at one *Stop*. Multiplicities are the ones the database enforces, so a route or trip with no trips or stop times is not excluded by the schema. A *Driver* holds *Driver–Bus Assignments*, and each assignment authorises the operation of one *Bus*. A *Bus* may be running one *Trip* and heading to one *Stop Time* at a time. The rule that a passenger has at most one active waiting request at a time is a constraint on the model and is shown as a note.

4.7.2 Entity-Relationship Design

The ER diagram in Figure 4.7.2.1 shows the persisted design in crow's foot notation: each entity is a table with its primary key (PK) and foreign keys (FK), and the symbol at the end of each line shows whether one, zero or one, or zero or many instances of that entity take part in the relationship. It includes the User table of Django's authentication system, restricted to the fields used, and omits Django's internal tables. Driver and Passenger each extend User through a one-to-one relationship. Bus has two nullable foreign keys, `current_trip` and `current_stop_time`. WaitingRequest links to Passenger, Trip and Stop. The field `Trip.shape_id` is a plain identifier that matches `Shape.shape_id`; it is not a foreign key, and Shape is not used in route selection, trip progression or ETA.

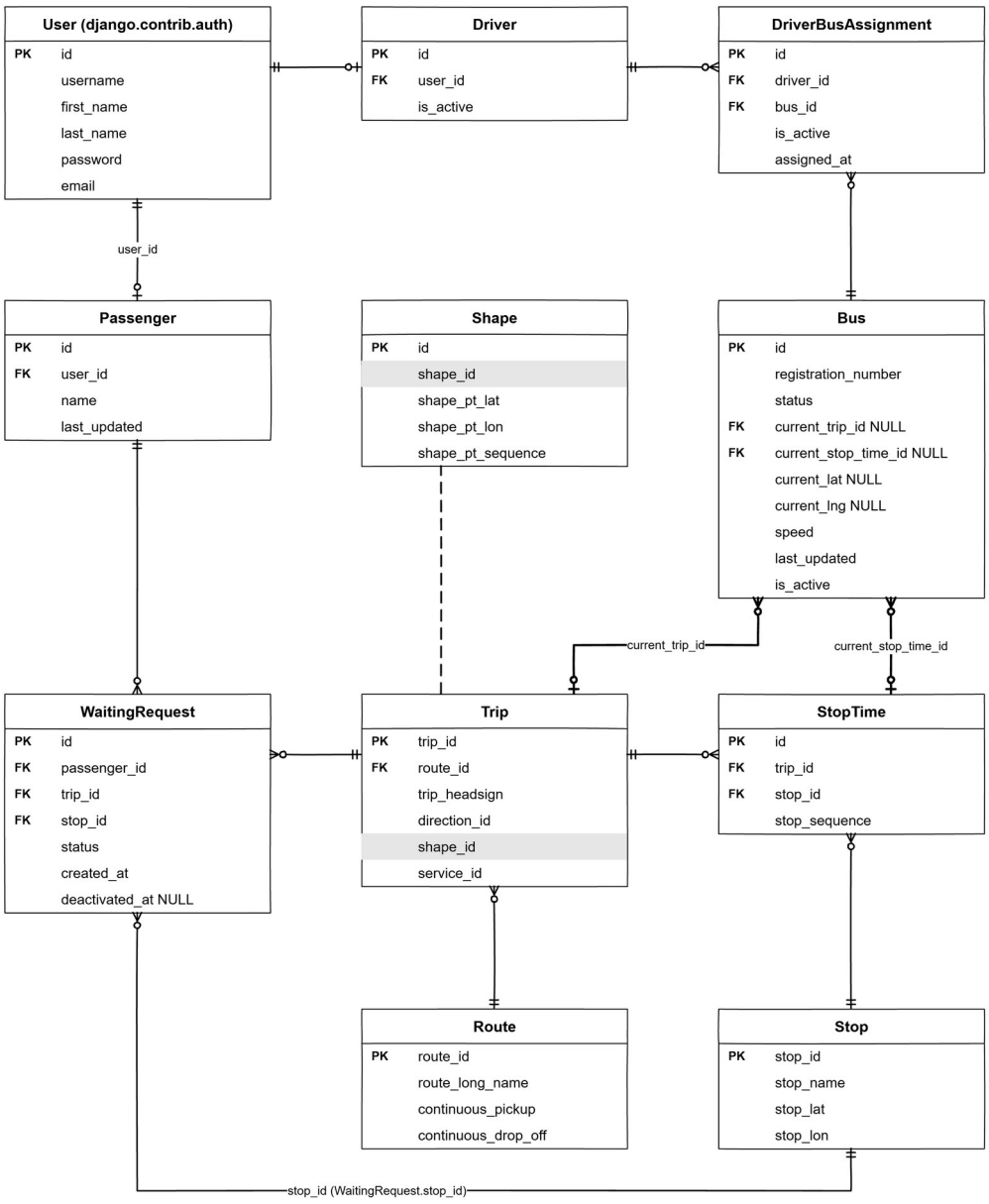


Figure 4.7.2.1: Entity-relationship diagram of the persisted data

Table 4.7.2.1 describes the main entities and the rules the model enforces.

Table 4.7.2.1: Data dictionary of the principal entities

Entity	Principal fields	Rules
Route	route_id (key), route_long_name, continuous_pickup, continuous_drop_off	One route has many trips
Stop	stop_id (key), stop_name, stop_lat, stop_lon	Imported from stops.txt; a row without coordinates is skipped
Trip	trip_id (key), route, trip_headsign, direction_id, shape_id, service_id	Identifier ends in the start time; trips ending in -08:00:00 are the representative trips
StopTime	trip, stop, stop_sequence	Unique on (trip, stop_sequence)
Shape	shape_id, shape_pt_lat, shape_pt_lon, shape_pt_sequence	Unique on (shape_id, shape_pt_sequence); imported but unused in logic
Bus	registration_number (unique), status, current_trip, current_stop_time, current_lat, current_lng, speed, last_updated, is_active	Status is IDLE, IN_TRANSIT or AT_STOP; only the latest state is stored
Driver	user (one-to-one), is_active	Deleting the user deletes the driver
DriverBusAssignment	driver, bus, is_active, assigned_at	Unique on (driver, bus)
Passenger	user (one-to-one), name, last_updated	Deleting the user deletes the passenger
WaitingRequest	passenger, trip, stop, status, created_at, deactivated_at	Status is WAITING, ON_BOARD, CANCELLED, EXPIRED or COMPLETED

The application stores only the current state of each bus. It has no history table for locations. Past positions are therefore not available for analysis.

4.8 Interface / Interaction Design

The browser interfaces were designed and implemented by the other group members. The passenger page shows a Leaflet map with OpenStreetMap tiles on which active buses appear as markers; the popup of a marker gives the registration number, status and speed, and the page refreshes the bus data every 15 seconds by polling the API. A header offers the active buses, trip planning, the passenger's own requests and a refresh control. In the trip-planning panel the passenger registers or signs in, searches for a route, chooses a destination, chooses a boarding stop from the ordered stops of the trip and creates a waiting request. The requests list offers controls to mark a request as boarded, cancel it or complete it, and shows the arrival estimate of a waiting request when one is available.

Figure 4.8.1 shows passenger screens: a route search that returns routes matching a typed name, a boarding stop confirmation, and the control that creates a waiting request. These screens call the route search, destination, trip-stops and waiting-request endpoints described in Table 4.5.3.1.

(a) Route search and selection



(b) Boarding-stop selection confirmed



(c) Waiting-request submission control

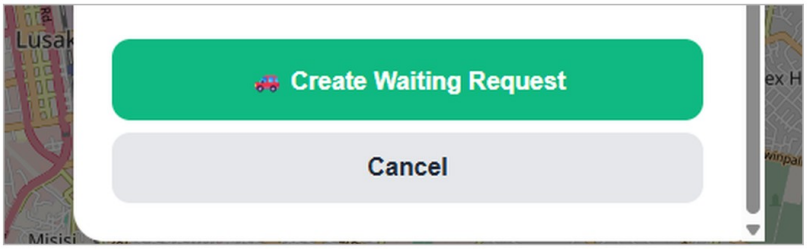


Figure 4.8.1: Passenger interface screens for route search, boarding-stop selection and waiting-request creation (interface developed by other group members)

Figure 4.8.2 and Figure 4.8.3 show a waiting request in the WAITING state, with controls to mark it as boarded or to cancel it, and in the ON_BOARD state, with a control to complete the trip. They correspond to the waiting-request lifecycle that the system enforces (Section 4.9.4).

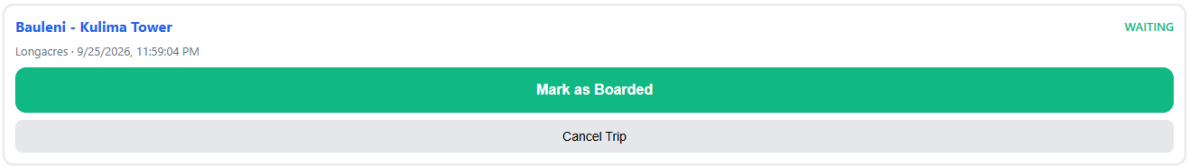


Figure 4.8.2: A waiting request in the WAITING state (interface developed by other group members)

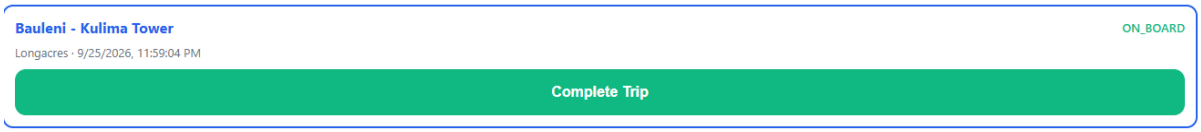


Figure 4.8.3: A waiting request in the ON_BOARD state (interface developed by other group members)

The driver dashboard implements the sequence in Figure 4.5.3.1. The driver signs in and sees the buses assigned to the account, selects one, and can activate it, deactivate it or reset its operational state. The driver chooses a route and destination and starts a trip, and the dashboard lists the number of passengers waiting at each stop of the trip. When the driver starts GPS tracking, the browser's geolocation service sends the device's latitude, longitude and speed to the server, and the dashboard shows the reported location, speed and vehicle status and marks the latest location on a map. Screenshots of the driver dashboard in use are given in Figures 6.3.3 to 6.3.7 in Chapter 6.

4.9 Algorithms / Models / Technical Design

4.9.1 Design Model

The design model describes how software objects realise the system operations. It consists of the design class diagram, interaction diagrams and package structure. One caution is necessary: the implementation is written as Django views, function-based service modules and Django models, not as a hierarchy of classes with behaviour. The design class diagram therefore shows modules with the «module» stereotype, listing their functions as operations, and shows the Django models as the persistent classes. The diagram shows every module of the services package, the controller and authorisation functions of `views.py` that the two interaction diagrams below use, and the persistent classes that carry behaviour or are used by the services.

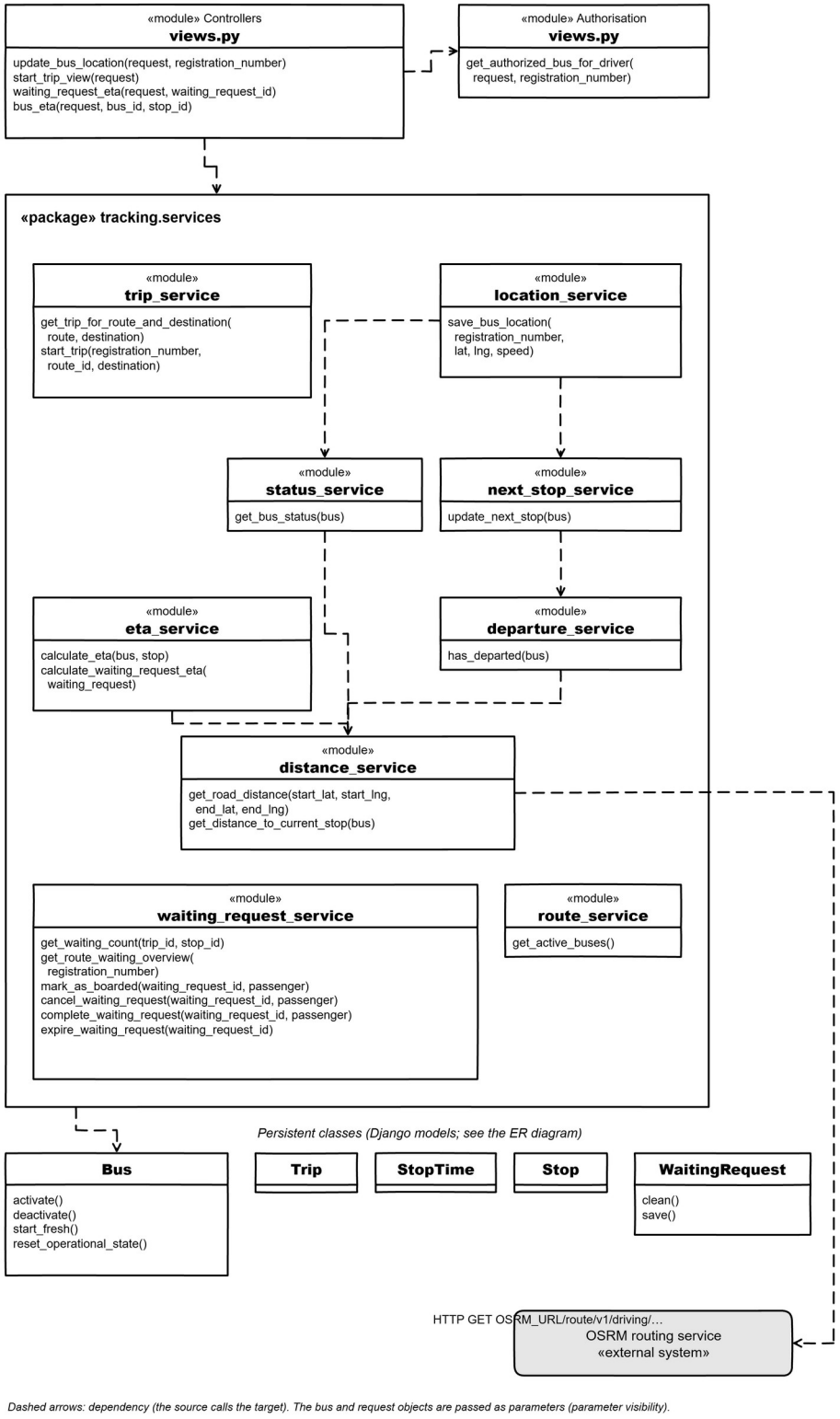


Figure 4.9.1.1: Design class diagram of the system modules and persistent classes

The two interaction diagrams realise the two operations with the most logic. Figure 4.9.1.2 shows `updateBusLocation`. The view first calls the authorisation helper. It then calls

`save_bus_location`, which loads the bus, sets the position and speed, and asks the stop services to update the next stop. The departure check calls `get_distance_to_current_stop`, which calls OSRM. `get_bus_status` then calls the distance service a second time, so one GPS update can involve two OSRM requests. The bus is finally saved and returned.

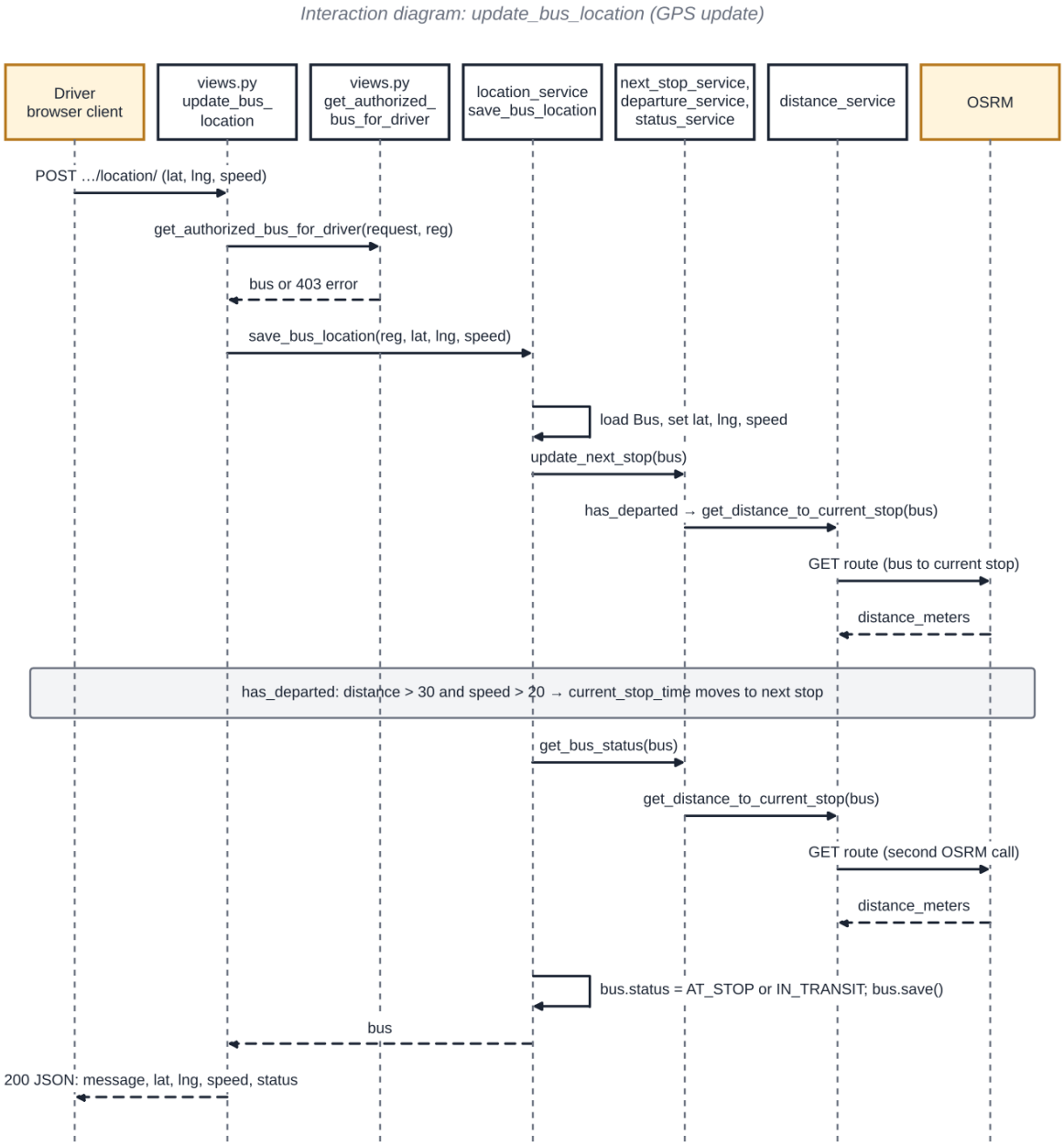


Figure 4.9.1.2: Interaction diagram for the system operation updateBusLocation

Figure 4.9.1.3 shows `getWaitingRequestEta`. The view retrieves only the signed-in passenger's request, then `calculate_waiting_request_eta` queries the stop sequences and eligible buses. For each eligible bus it calls the distance service, which calls OSRM. The shortest valid duration, converted to minutes, is returned, or null when none is valid.

Interaction diagram: waiting_request_eta (arrival time prediction)

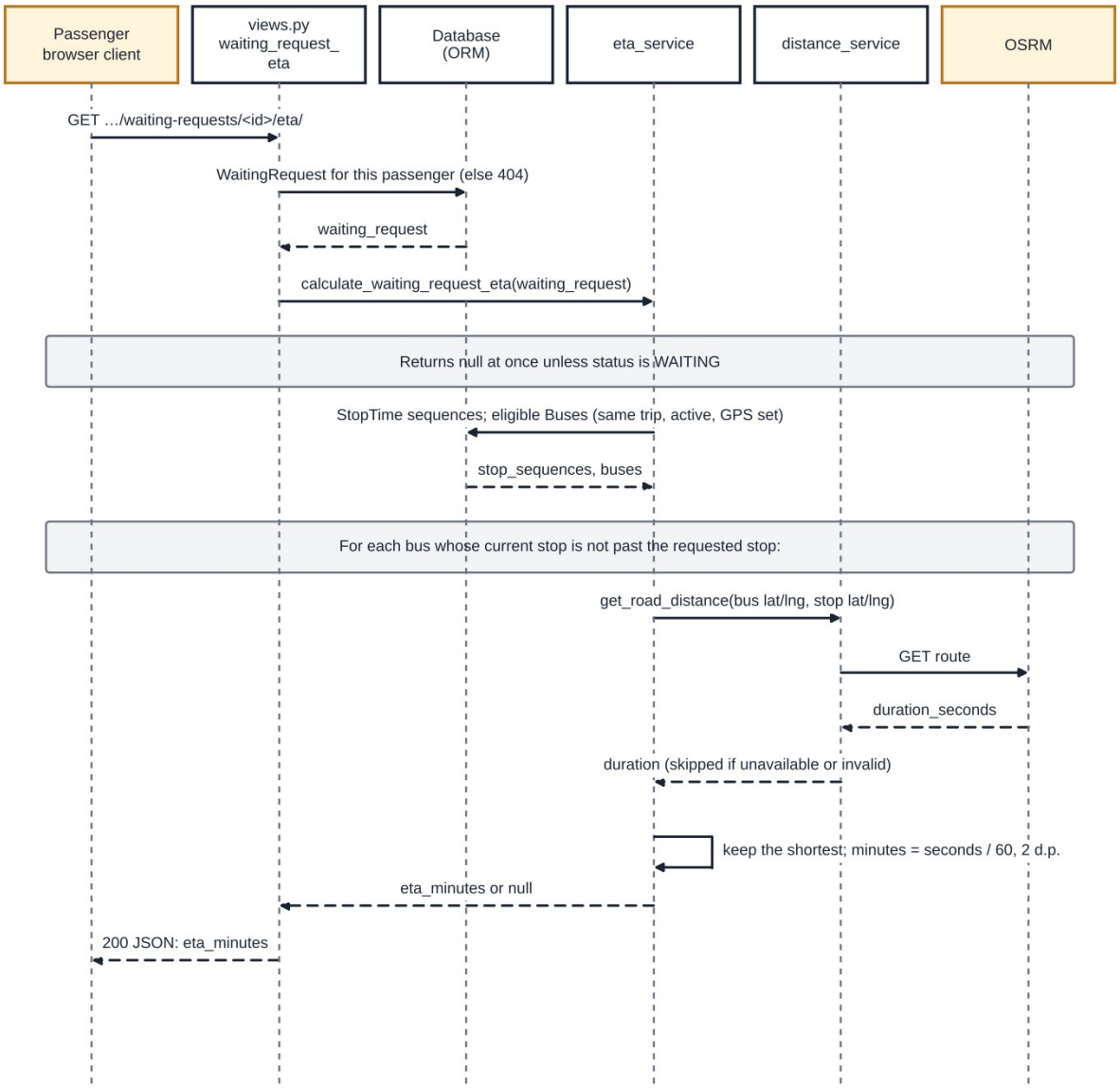


Figure 4.9.1.3: Interaction diagram for the system operation getWaitingRequestEta

4.9.2 Visibility

Visibility in the design is mostly parameter visibility: a service obtains the bus or request as a parameter, as in `update_next_stop(bus)` and `calculate_waiting_request_eta(waiting_request)`. Module-level (global) visibility applies to the OSRM base URL, which is read once from the environment in the distance module. Local visibility arises where a service loads records itself, for example `save_bus_location` loads the bus by registration number.

4.9.3 Trip Selection and Bus State Logic

A driver chooses a route and a destination. `get_trip_for_route_and_destination` examines the representative trips of the route and returns the one whose last stop has the chosen destination name; if none is found it returns nothing and `start_trip` raises an error. `start_trip` sets the bus's current trip and first stop time, sets the status to IDLE and the bus to active, inside one database transaction.

On each GPS update the position and speed are stored without validation, the next stop is updated and the status is derived. Table 4.9.3.1 states the rules. Distances come from OSRM and are in metres. The code does not specify the unit of the speed value received from the device, so this report does not assign one.

Table 4.9.3.1: Rules for next-stop update and bus status

Rule	Condition
Next stop advances (has_departed)	The bus has a current stop time, the road distance to the current stop is greater than 30, and the speed is greater than 20. If the routing request fails, the bus is treated as not departed
Status IDLE	The bus has no current stop time, or the distance to the current stop is unavailable
Status AT_STOP	The road distance to the current stop is at most 30 and the speed is at most 20
Status IN_TRANSIT	Any other case, including when the routing request fails

4.9.4 Waiting-Request Lifecycle

A waiting request is created in the WAITING state. A passenger can mark it as boarded (ON_BOARD) or cancel it (CANCELLED). An on-board request can be completed (COMPLETED). A WAITING request can also be set to EXPIRED by an expiry endpoint. Cancelled, completed and expired requests record the time of deactivation. A passenger cannot create a request while another is WAITING or ON_BOARD.

4.9.5 ETA Algorithm

The ETA for a waiting request is calculated by the following steps. If any condition fails, the estimate is not available and null is returned instead of a guessed value.

1. Return null unless the request status is WAITING.
2. Find the stop sequences at which the requested stop appears in the requested trip; return null if there are none.
3. Select candidate buses: active, with status IN_TRANSIT or AT_STOP, on the same trip, with coordinates present and a current stop time on that trip.
4. Discard a candidate whose coordinates are not finite or are outside valid latitude and longitude ranges, or whose current stop sequence is greater than every sequence of the requested stop (the bus has passed it).

5. For each remaining candidate, request the road duration from the bus position to the stop from OSRM. Skip the candidate if the request fails, the response is malformed, or the duration is not finite or is negative.
6. Return the shortest remaining duration, converted to minutes and rounded to two decimal places, or null if no candidate remains.

The duration is that of the OSRM route profile and is returned in seconds. It is not adjusted for live traffic or dwell time at stops. Section 6.6 discusses the consequences.

4.10 Chapter Summary

The analysis defines three features supported by a use-case diagram and two system sequence diagrams. The design is shown through the architecture, a conceptual domain model, an ER diagram and a design model made of modules, interaction diagrams and the rules for trip selection and bus state. The key technical rules are trip selection from representative trips, bus status and next-stop thresholds, the waiting-request lifecycle and the conditional ETA algorithm. Chapter 5 documents their implementation.

CHAPTER 5: IMPLEMENTATION / DEVELOPMENT

5.1 Introduction

This chapter describes how the Bus Tracking System was implemented. It gives an overview of the whole system and then documents in detail the backend and the deployment configuration completed by myself. It describes the environment, the components, the key implementation details, the integration points, the challenges met. Code excerpts are abridged from the deployed system source code.

The implemented system has three parts. The first is the Django server, which exposes a JSON API, stores the data and calls the routing service. The second is the passenger interface, a browser page with a map of active buses and a trip-planning panel. The third is the driver dashboard, a browser page for bus operations and GPS tracking. The browser interfaces were developed by the other group members and are described in Section 4.8.

5.2 Development Environment

The server was developed in Python with Django and Django REST Framework, using Git and GitHub for source control. The project is started locally with `python manage.py migrate` and `python manage.py runserver`. Deployment used Render for hosting and Brevo for password-reset e-mail. The verified environment variables are listed in Table F.2 in Appendix F; the settings read each of them with `os.environ`, so no secret is stored in the source.

The repository contains the Django project `bus_tracking` (settings, URL configuration and WSGI entry point), the application `tracking`, and the `gtfs` directory with the GTFS text files. Within `tracking` are the models, views, serialisers, URL configuration, administration

registrations, the `services` package, the `management/commands` package, the migrations and the tests.

5.3 System Components / Modules

Table 5.3.1: System components and their responsibilities

Component	Responsibility
models.py and migrations	Route, Stop, Trip, StopTime, Shape, Bus, Driver, DriverBusAssignment, Passenger and WaitingRequest; Bus methods activate, deactivate, start_fresh and reset_operational_state
management/commands (5 commands)	import_routes, import_stops, import_trips, import_stop_times and import_shapes, which read the GTFS files
views.py and urls.py	Passenger and driver authentication, route and trip endpoints, bus operations, GPS update, waiting requests, ETA and waiting counts; the authorisation helper
serializers.py	Serialisation and validation, including the trip and stop checks of a waiting request and the start-trip payload
services/trip_service.py	get_trip_for_route_and_destination and start_trip
services/location_service.py	save_bus_location
services/next_stop_service.py and departure_service.py	update_next_stop and has_departed
services/status_service.py	get_bus_status
services/distance_service.py	get_road_distance (OSRM call) and get_distance_to_current_stop
services/eta_service.py	calculate_eta and calculate_waiting_request_eta
services/ waiting_request_service.py	Waiting counts and overview; board, cancel, complete and expire operations
admin.py	Registers Route, Stop, Bus, Passenger, WaitingRequest, Driver and DriverBusAssignment in the administration site
Passenger interface (other members)	Browser page with a Leaflet map of active buses refreshed every 15 seconds, route search, destination and boarding-stop selection, waiting-request creation and the requests list (Section 4.8)
Driver dashboard (other members)	Browser page for driver sign-in, bus selection, activation, trip start, waiting counts by stop and GPS tracking from the device's geolocation (Section 4.8)
tests.py	Automated tests for ETA, waiting-overview, access-control, registration, throttling and debug-default behaviour
bus_tracking/settings.py	Environment-based settings for database, secrets, hosts, CSRF origins, static files, secure cookies and e-mail

The application stores only the current state of each bus. It does not have a historical location table. The `route_service` module defines a function `get_active_buses` that no other code calls.

5.4 Key Implementation Details

5.4.1 Data Model and Migrations

The models implement the design in Figure 4.7.2.1. Driver and Passenger each have a one-to-one relationship to Django's User model, with the related names `driver_profile` and `passenger_profile`, which the views use to decide whether an account is a driver or a passenger. Bus holds the current trip and current stop time as nullable foreign keys that are set to null if the referenced record is deleted. The `Bus.deactivate`, `Bus.activate` and `Bus.start_fresh` methods run inside database transactions. `start_fresh` and `deactivate` call `reset_operational_state`, which clears the trip, stop, coordinates and speed and sets the status to IDLE. The nine migrations record how the schema evolved (Section 3.2.3).

5.4.2 GTFS Import Commands

Each import command opens its file under the `gtfs` directory, reads it with Python's `csv` module and uses `update_or_create`, so running a command again updates existing rows instead of duplicating them. Trips are imported only for routes that exist, and stop times only for trips and stops that exist; other rows are counted as skipped. The stop import skips a row with no coordinates, which is the one stop in the supplied feed that has none. Each command prints the numbers imported, updated and skipped. The commands must therefore be run in dependency order: routes and stops, then trips, then stop times; shapes are independent.

5.4.3 Driver Authentication and Authorisation

Driver sign-in authenticates the credentials with Django, rejects accounts that have no driver profile or whose driver is inactive, and then starts a session. Every protected bus operation passes through the helper `get_authorized_bus_for_driver`, abridged below. It returns the bus, or an error response with HTTP status 403.

```
def get_authorized_bus_for_driver(request, registration_number):
    bus = get_object_or_404(Bus,
registration_number=registration_number)
    if not hasattr(request.user, "driver_profile"):   # 403: not a
driver account
    driver = request.user.driver_profile
    if not driver.is_active:                          # 403: inactive
driver
    assignment_exists = DriverBusAssignment.objects.filter(
        driver=driver, bus=bus, is_active=True).exists()
    if not assignment_exists:                         # 403: not
assigned to this bus
    return bus, None
```

In the final stage of development the access rules of the API were tightened, and automated tests were added for each rule. The settings define `IsAuthenticated` as the default permission class for Django REST Framework, so an endpoint is protected unless it explicitly allows anonymous access [14]. The routes, stops and buses viewsets became read-only and public,

because passengers need them to plan a journey, and the passenger viewset was removed from the router. The waiting-request viewset allows only listing, retrieving and creating, requires a passenger account, and treats status, passenger and timestamps as read-only, so a client cannot set its own lifecycle state. The `expire` endpoint is restricted to administrators, and the waiting-overview endpoint uses the same assignment helper as the driver operations. Registration now applies Django's configured password validators. The passenger and driver sign-in views are limited to five attempts per minute for each client address by throttle classes in `tracking/throttles.py`.

The helper is called by the driver-session, activate, deactivate, start-fresh, start-trip and location views, so an unauthorised driver can neither operate nor report the position of a bus to which they are not actively assigned. The views that use it require an authenticated session.

5.4.4 Bus Operations and Trip Start

Activation, deactivation and reset are POST operations that call the Bus methods after authorisation. `start_trip` retrieves the bus and route (returning HTTP 404 if either does not exist), chooses the trip with `get_trip_for_route_and_destination`, finds the first stop time by stop sequence, and in one transaction sets the bus's current trip and current stop time, sets the status to IDLE and marks the bus active. If no trip ends at the chosen destination, `start_trip` raises a `ValueError`. The start-trip view does not catch this exception, so that case would produce a server error rather than a clear client error. This is recorded as a defect in Section 6.6.

5.4.5 Real GPS Update Handling

The location view authorises the driver and then calls `save_bus_location` with the latitude, longitude and speed from the request body; the speed defaults to zero if absent. `save_bus_location` loads the bus, stores the three values, calls `update_next_stop`, derives the status with `get_bus_status` and saves the bus. The response contains a confirmation message, the registration number, the latitude, the longitude, the speed and the status. The values are not validated before storage; validation of coordinates is applied only when an ETA is calculated.

`update_next_stop` does nothing if the bus has no trip or no current stop. Otherwise it asks `has_departed` whether the bus has left its current stop. If so, it moves the current stop time to the next stop time of the same trip. The status logic and thresholds are in Table 4.9.3.1. A failure of the routing request is caught in both services, so a GPS update is still saved when OSRM is unreachable.

5.4.6 Waiting Requests

The waiting-request viewset is restricted to authenticated users and returns only requests that belong to the signed-in passenger. When a request is created, the serialiser accepts only representative trips and checks that the chosen stop is served by the trip. The view then refuses creation while the passenger has a request that is ON_BOARD or WAITING, so a passenger cannot hold two active requests. The service functions `mark_as_boarded`, `cancel_waiting_request`, `complete_waiting_request` and `expire_waiting_request` change the status, filter on the expected current status and set the deactivation time where

relevant. `get_waiting_count` counts WAITING requests for a trip and stop, and `get_route_waiting_overview` returns, for the current trip of a bus, each stop in sequence with its waiting count, or an empty list when the bus has no trip.

5.4.7 OSRM Integration

All routing calls pass through `get_road_distance`, which builds a request to the OSRM route service from the configured base URL and returns the distance in metres and the duration in seconds of the first route [8]. Coordinates are sent as longitude then latitude, and the overview geometry is switched off because only distance and duration are needed. The base URL is read from OSRM_URL, with the default http://localhost:5000, and any trailing slash is removed.

```
OSRM_URL = os.getenv("OSRM_URL", "http://localhost:5000").rstrip("/")

def get_road_distance(start_lat, start_lng, end_lat, end_lng):
    url = (f"{OSRM_URL}/route/v1/driving/{start_lng},{start_lat};
{end_lng},{end_lat}"
           "?overview=false")
    response = requests.get(url)
    response.raise_for_status()
    route = response.json()["routes"][0]
    return {"distance_meters": route["distance"], "duration_seconds":
route["duration"]}
```

The call has no explicit timeout. The status and departure services catch only `RequestException`, so a malformed OSRM response in those services is not handled in the way it is in the ETA service, which also catches index, key, type, value and overflow errors. This limitation is recorded in Section 6.6.

5.4.8 ETA Calculation

`calculate_waiting_request_eta` implements the steps in Section 4.9.5. It uses a database query to select candidate buses and then filters them in Python, checking that coordinates are finite and in range, that the bus has not passed the stop and that the OSRM duration is finite and non-negative. A second function, `calculate_eta`, returns the road duration in minutes from a bus to a stop for the bus-and-stop endpoint, and it does not apply these checks.

5.4.9 Password Reset and Deployment Configuration

Django’s built-in password-reset views are routed at `password-reset/`, with a reset timeout of 3600 seconds. E-mail is sent with the Brevo backend of django-anymail, using the credential and sender address from the environment. The settings read the secret key, debug flag, allowed hosts, trusted CSRF origins and database address from the environment, and refuse to start without a secret key when debugging is off. Debugging is off unless the debug flag is set to true. In that mode the settings also mark session and CSRF cookies as secure and trust the proxy’s forwarded-protocol header. The application is served by gunicorn through `bus_tracking.wsgi.application`, and WhiteNoise serves static files.

On Render the service is configured with the following commands, copied from its settings. The build command installs the pinned dependencies and collects the static files; the start command applies any pending migrations and then starts gunicorn, bound to all interfaces on the port that the platform supplies in the PORT variable. Migrations therefore run each time the service starts.

```
Build command:
pip install -r requirements.txt && python manage.py collectstatic --
noinput

Start command:
python manage.py migrate --noinput && gunicorn
bus_tracking.wsgi:application --bind 0.0.0.0:$PORT
```

5.5 Integration

Integration took place at four boundaries. First, the browser interfaces developed by the other group members call the JSON API under `/api/`; the Django project also serves their HTML pages for the landing, passenger and driver screens, which are described in Section 4.8. Second, the views call the service modules, which use the ORM to read and write the database. Third, the distance service calls OSRM over HTTP. Fourth, the settings read the database address from DATABASE_URL, and the e-mail backend sends through Brevo.

5.6 Challenges and Solutions

- **From simulated to real GPS.** The proposal assumed simulated positions. After supervisor feedback, the system was changed to accept real coordinates and speed from a driver's device, protected by the assignment check, so that the source of position is an identified, authorised driver.
- **Giving a coordinate meaning.** A coordinate alone does not tell a passenger which stop is next. GTFS routes, trips and ordered stop times were imported and a bus keeps references to its current trip and stop, so position can be related to the route and the stops ahead.
- **Repeated trip variants.** The GTFS feed holds three start-time variants of each trip with identical stops. Using the 08:00:00 variant as the representative trip avoids ambiguity when a passenger or driver selects a route and direction, at the cost of not modelling the time of day.
- **Unreliable routing input.** OSRM may be unreachable or return unusable data. The ETA returns null in such cases instead of an unsupported number, and the status services fall back to IN_TRANSIT or not departed.
- **Change of the waiting-request model.** The migration history shows the request moving from a route-based to a trip-based design so that a request refers to a specific route direction and stop.
- **Secrets and environments.** Secrets and addresses were moved into environment variables, and the settings fail safely when the secret key is missing outside debug

mode. Debug mode was also made opt-in, so a missing variable can no longer enable it.

- **Open endpoints.** A code review found that the model viewsets and the `expire` endpoint allowed anonymous access. The default permission, read-only viewsets, the removal of the passenger endpoint and the administrator restriction closed these gaps, and tests now check that anonymous, passenger and driver users are refused.

5.7 Individual Contribution (Group Projects)

The backend features as well as the deployment configuration were carried out by myself. . Table 5.7.1 distinguishes the author's work from shared and others' work. Appendix A gives the formal Individual Contribution Statement.

Table 5.7.1: Contribution summary by activity

Activity	Contribution	Notes
Django and DRF backend, models and migrations	Kenneth	Section 5.4.1
GTFS import commands	Kenneth	Section 5.4.2
Driver authentication, authorisation and assignment checks	Kenneth	Section 5.4.3
Bus activation, deactivation, reset and trip start	Kenneth	Section 5.4.4
Real GPS update handling; bus state, status and next-stop logic	Kenneth	Section 5.4.5
Waiting-request services and OSRM integration; ETA logic	Kenneth	Sections 5.4.6 to 5.4.8
Automated backend tests	Kenneth	Section 6.3
Deployment configuration	Primary (author)	Section 5.4.9
Problem analysis, requirements and Yango interface analysis	Joint (group)	Section 3.3; all members used Yango as passengers
Literature review	Joint (group);	
Browser interfaces (passenger and driver screens, map display)	Other group members	Not claimed; credited in Section 4.8

5.8 Chapter Summary

The backend implements the data model, GTFS imports, driver authorisation, bus and trip operations, GPS handling, waiting requests, OSRM integration and conditional ETA in function-based service modules, configured for deployment through environment variables. Some implementation limits were identified and are recorded for Chapter 6.

CHAPTER 6: TESTING, RESULTS AND EVALUATION

6.1 Introduction

This chapter reports the testing evidence for the web application as a whole and evaluates the system against the objectives of Section 1.4, within its limits. Testing was conducted at two levels: automated tests of the server, and manual functional and integration checks of the deployed passenger interface, driver dashboard and server working together. A run of the automated tests is reported in Section 6.3, together with the manual checks. The chapter also draws on a review of the source code. Section 6.8 then separates the results attributable to the author. The testing was not exhaustive acceptance testing, and no performance benchmark, user study or field trial was carried out.

6.2 Testing Strategy

The automated tests use Django's test framework with Django REST Framework's `APITestCase`. Each test builds its own data in a temporary test database, authenticates a test user and calls the real endpoint through the test client, so the tests exercise the view, serialiser, service and model together. The routing function used by the ETA service is replaced by a mock, so the tests do not depend on a running OSRM server and do not test OSRM itself. The tests are run with `python manage.py test`, which discovers test modules in the project [13]. The command `python manage.py check` was also used. It is different: it validates the Django configuration and is not a test run.

The strategy first concentrated on the most conditional piece of logic, the ETA, together with the waiting-passenger overview used by drivers. These tests cover valid behaviour, unavailable conditions and access restriction. In the final stage of development, tests were added for the access rules of the API, registration, login throttling and the debug default. Other modules were verified by inspection of the source code during development and during the preparation of this report, but they have no automated tests.

The second level was manual functional and integration testing. Key flows were checked through the deployed passenger interface and driver dashboard and the connected Django application, so that the parts were tested working together rather than as isolated services. The browser interfaces have no automated tests, so this level is the only evidence for them. This level is described in Section 6.3.

6.3 Test Cases and Results

Table 6.3.1 lists the behaviours of the server that the automated tests verify. The table states what each test asserts. `python manage.py test` was run in the deployment working copy of the integration branch, with the debug environment variable set to True because debugging is off by default; Figure 6.3.1 shows the console output. Django found 35 tests, ran all 35 in 19.248 s and reported OK, and the system check that precedes the run identified no issues. The earlier 9 tests of the ETA and waiting-overview classes are rows 1 to 9 of the table, and rows 10 to 19 are the access-control, registration, throttling and debug-default tests added in the final stage. The result does not show that the untested modules are correct.

Table 6.3.1: Behaviours verified by the automated tests

No.	Behaviour	Condition and asserted outcome	Test class
1	Shortest valid duration is chosen	Two eligible buses on the trip, OSRM mocked with different durations: the response contains the smaller duration in minutes and the routing function was called once per bus	WaitingRequestEtaTests
2	No ETA when GPS is unavailable	A bus without coordinates: the response has `eta_minutes` null and OSRM is not called	WaitingRequestEtaTests
3	Passed stop is excluded	A bus whose current stop is beyond the requested stop: ETA null and OSRM not called	WaitingRequestEtaTests
4	Non-WAITING requests have no ETA	Requests in other statuses: ETA null and OSRM not called	WaitingRequestEtaTests
5	Passenger scoping	A different passenger asks for the request: the response is HTTP 404 and OSRM is not called	WaitingRequestEtaTests
6	Unavailable routing response	The mocked routing function fails with an index error (no route): ETA null	WaitingRequestEtaTests
7	Waiting counts per stop	A boarding request is counted only at its own stop in the overview of the bus's current trip	WaitingOverviewTests
8	Non-waiting and other-trip requests excluded	Requests that are not WAITING, or belong to another trip, add nothing to the counts	WaitingOverviewTests
9	No active trip	A bus without a current trip returns a null trip and an empty stop list	WaitingOverviewTests
10	Transport writes disabled	POST, PUT, PATCH and DELETE on routes, stops and buses return HTTP 405 for anonymous, passenger and driver users, and the records are unchanged	ApiAccessSecurityTests
11	Passenger directory removed and public reads kept	The passenger endpoint returns 404; routes, stops, buses, route search, destinations, trip stops, bus ETA and waiting count return 200 without signing in	ApiAccessSecurityTests
12	Default permission	The default permission class is IsAuthenticated, and anonymous requests to the current-user and waiting-request endpoints return 403	ApiAccessSecurityTests
13	Driver authorisation	Passengers, unassigned drivers, inactive drivers and inactive assignments receive 403 on every bus operation and on the waiting overview, and the bus is unchanged; an assigned driver can update GPS and operate the bus	ApiAccessSecurityTests
14	Expiry restricted	Anonymous, passenger and driver users receive 403 and the request stays WAITING; an administrator receives 200 and the request becomes EXPIRED	ApiAccessSecurityTests

No.	Behaviour	Condition and asserted outcome	Test class
15	Password validation	Short, common and numeric passwords, and one equal to the username, are rejected with 400 and no account is created; the configured validators are the ones applied	ApiAccessSecurityTests
16	Waiting-request integrity	Update and delete return 405; status, passenger and timestamps supplied by the client are ignored on creation; another passenger cannot board, cancel or complete a request	ApiAccessSecurityTests
17	Passenger account required	Anonymous and non-passenger users receive 403 on every waiting-request path, and reads stay limited to the owner	ApiAccessSecurityTests
18	Login throttling	The sixth sign-in attempt in a minute from one address returns 429 for passenger and driver sign-in, other addresses are unaffected, the counters are separate and registration is not throttled	LoginThrottleTests
19	Debug default	The debug setting is false when the environment variable is absent and true when it is set to True	DebugDefaultTests

The ETA tests cover five behaviours in particular: selecting the shortest valid routing duration when several suitable buses are available; a bus whose GPS coordinates are missing or unavailable; excluding a bus that has already passed the requested boarding stop; producing no ETA for a waiting request that is not in the WAITING state; and an unavailable or unsuccessful routing response. The run is automated testing. It covers application and service-layer behaviour, not the visual layout of the frontend or every end-to-end user scenario.

```
(deployment_venv) C:\Users\Muson\bus_tracking_deployment>set DJANGO_DEBUG=True

(deployment_venv) C:\Users\Muson\bus_tracking_deployment>python manage.py check
System check identified no issues (0 silenced).

(deployment_venv) C:\Users\Muson\bus_tracking_deployment>python manage.py test
Found 35 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...................................
----------------------------------------------------------------------
Ran 35 tests in 19.248s

OK
Destroying test database for alias 'default'...

(deployment_venv) C:\Users\Muson\bus_tracking_deployment>
```

Figure 6.3.1: Console output of python manage.py test: 35 tests run, result OK

These cases confirm, within the fixtures that the tests build, that the ETA logic declines to produce a value when its inputs are missing or invalid, selects the shortest valid duration, and restricts a request to its owner.

Manual functional and integration checks. The deployed system was also checked by hand through its passenger interface and driver dashboard. The checks covered:

- access to the deployed application interfaces;
- the passenger-side and driver-side authentication and access flows;
- use of the driver dashboard by registered and authorised drivers;
- the driver's trip and GPS-update workflow against the system;
- the GPS-related responses and the update of the bus's current state;
- communication between the passenger-side functions and the deployed system; and
- the operation of the deployment configuration and the deployed Django application during the verification sessions.

These checks were made on the deployed system, using a test passenger account, a test driver account assigned to the bus BUS123, and the location reported by the device used for the checks, which was not an operating bus. Figures 6.3.2 to 6.3.14 are screenshots taken during the checks. Each shows the outcome of a flow at one moment. The browser screens were developed by the other group members, and the server behaviour behind each of them is described in Chapter 5. They demonstrate verification of key implemented functions. They do not constitute exhaustive acceptance, performance or large-scale field testing.

Driver flow. Figure 6.3.2 shows the landing page, where the user chooses the passenger or the driver sign-in. Figure 6.3.3 shows the driver sign-in. After signing in, the dashboard lists the buses assigned to the driver account and shows the status of the selected bus, IDLE before activation (Figure 6.3.4). Figure 6.3.5 shows the message that the bus was activated and the destinations offered for the route Bauleni – Kulima Tower.

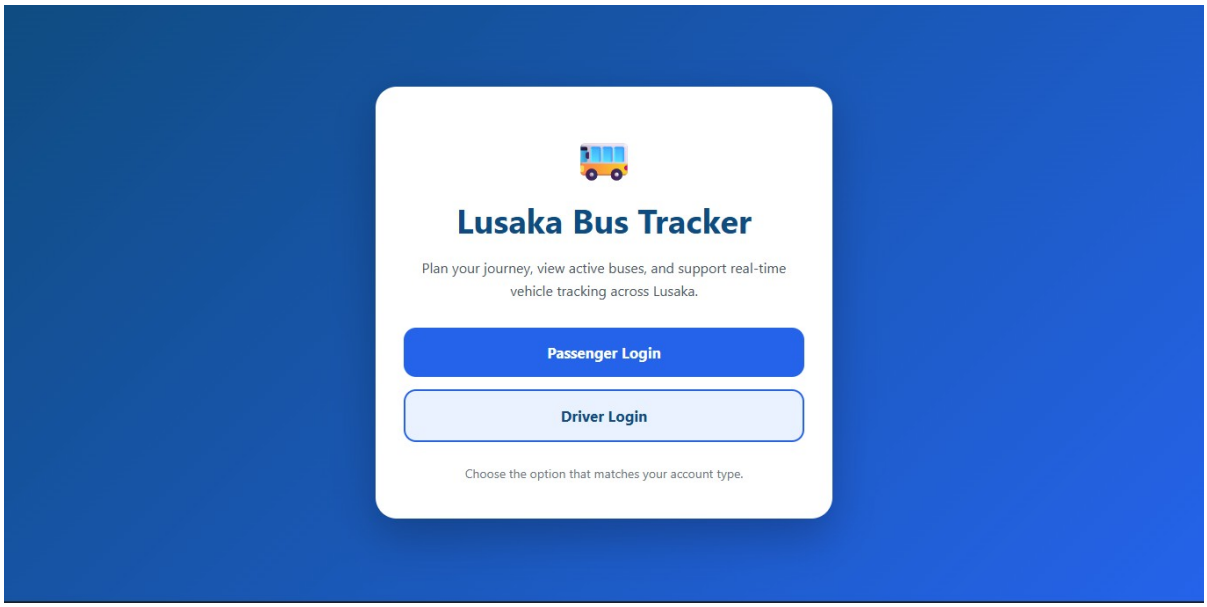


Figure 6.3.2: Landing page of the deployed system (interface developed by other group members)

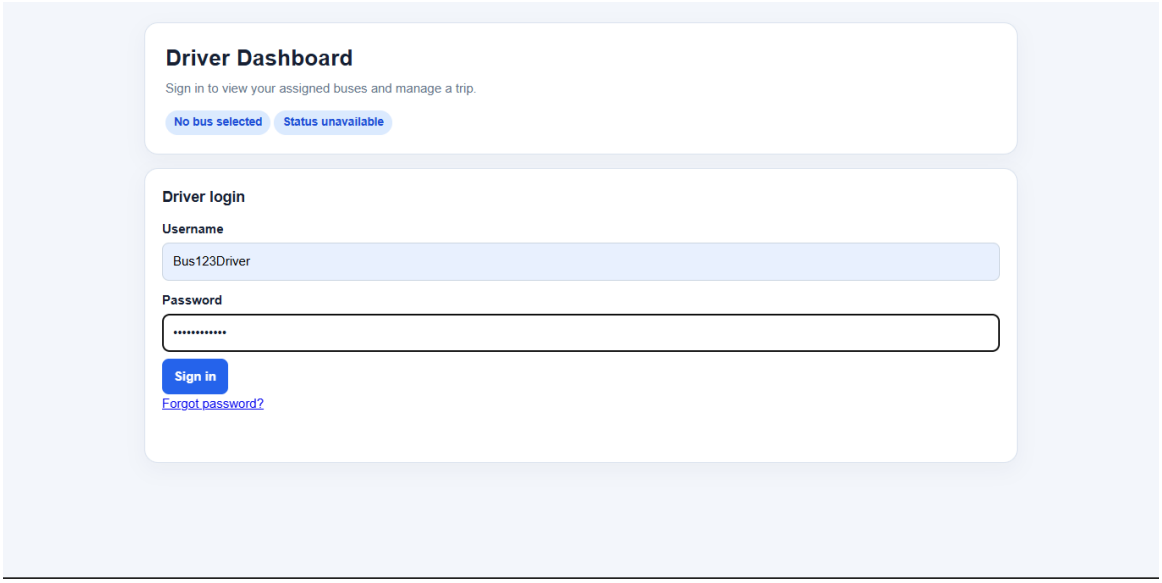


Figure 6.3.3: Driver sign-in (interface developed by other group members)

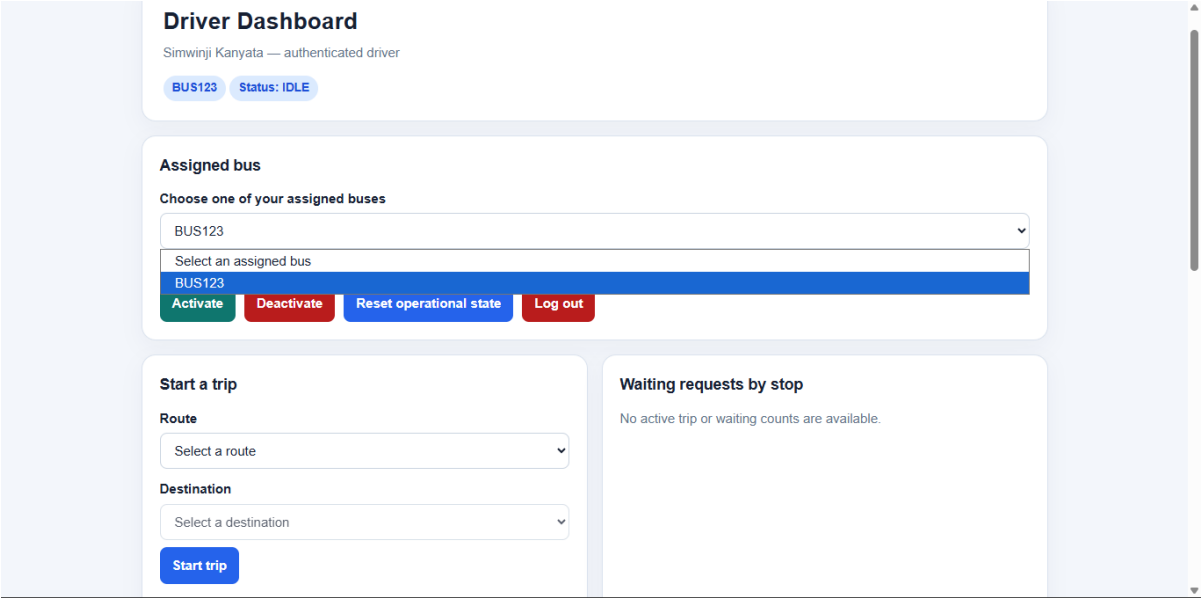


Figure 6.3.4: Driver dashboard with the assigned bus selected and its status IDLE (interface developed by other group members)

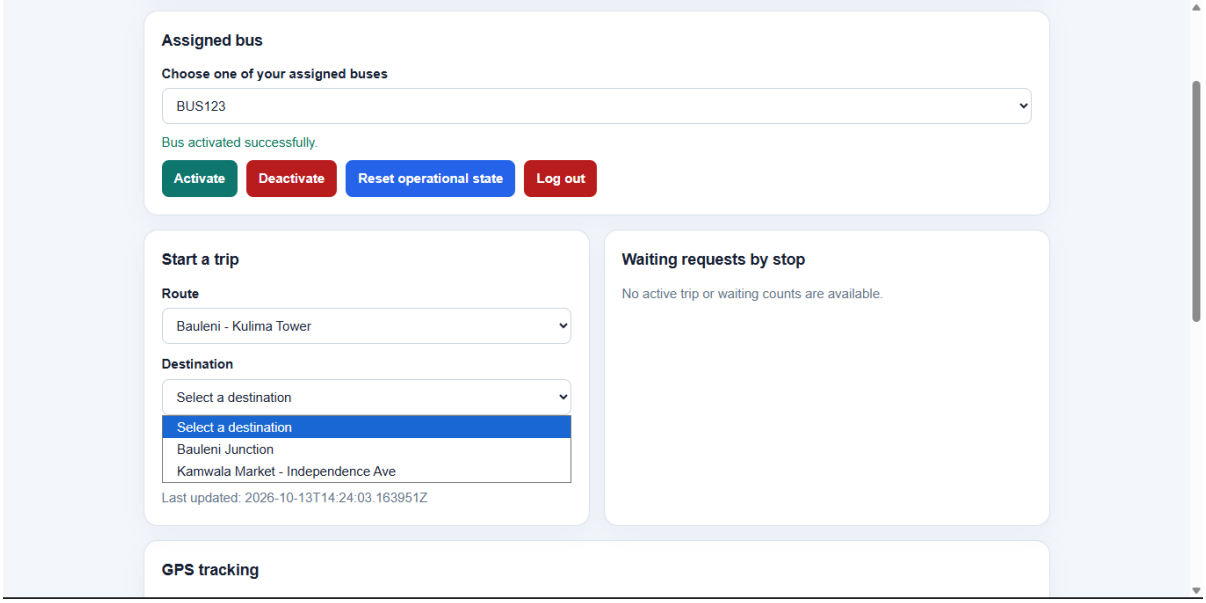


Figure 6.3.5: Bus activated and destination choice for the route Bauleni – Kulima Tower (interface developed by other group members)

Figure 6.3.6 shows the browser asking permission to read the device's location, with the dashboard waiting for a fix. Figure 6.3.7 shows the result after the fix: the message that the GPS update was accepted, the reported coordinates, a speed of 0, the status IN_TRANSIT and the same position on the live map. The status is IN_TRANSIT and not AT_STOP because the device was not within 30 of the bus's current stop, which the status rule requires for AT_STOP (Table 4.9.3.1).

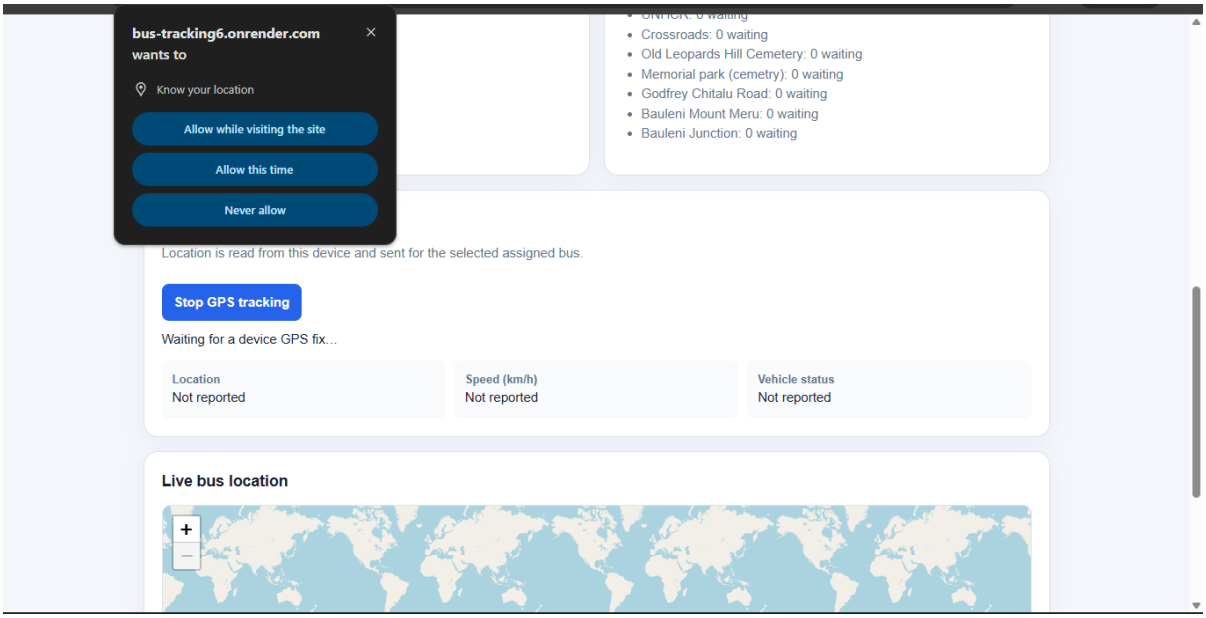


Figure 6.3.6: Browser permission request and waiting for a device GPS fix (interface developed by other group members)

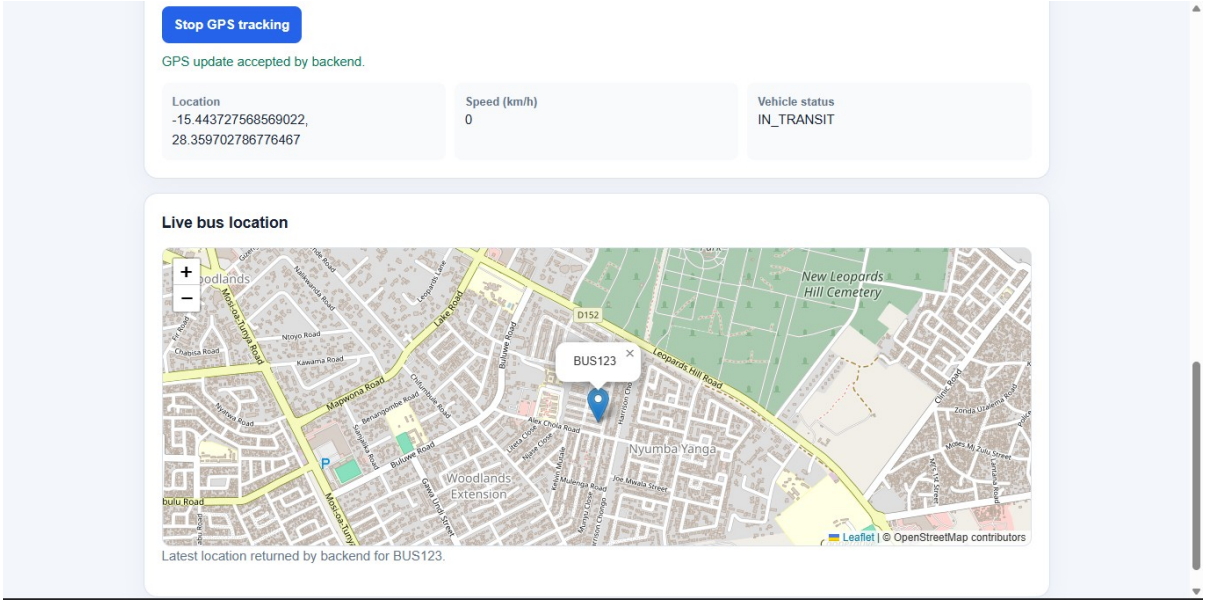


Figure 6.3.7: GPS update accepted, reported location, speed, status and live map (interface developed by other group members)

Passenger flow. Figure 6.3.8 shows the bus as a marker on the passenger map at the position reported in Figure 6.3.7. Figure 6.3.9 shows passenger sign-in, Figure 6.3.10 the route search and the destinations offered, and Figure 6.3.11 the choice of a boarding stop. Figure 6.3.12 shows the waiting request created for the journey, with its arrival estimate of about 10 minutes and the options to mark the passenger as boarded or to cancel. The estimate comes from the routing service; it was not compared with an observed arrival time.

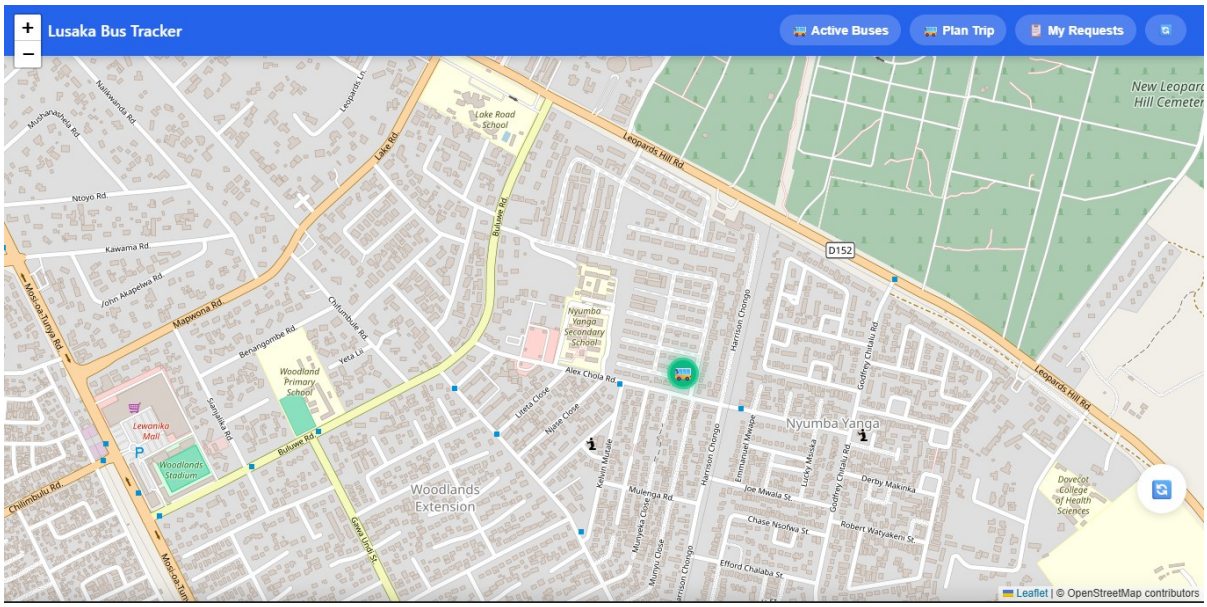


Figure 6.3.8: Passenger map showing the bus marker (interface developed by other group members)

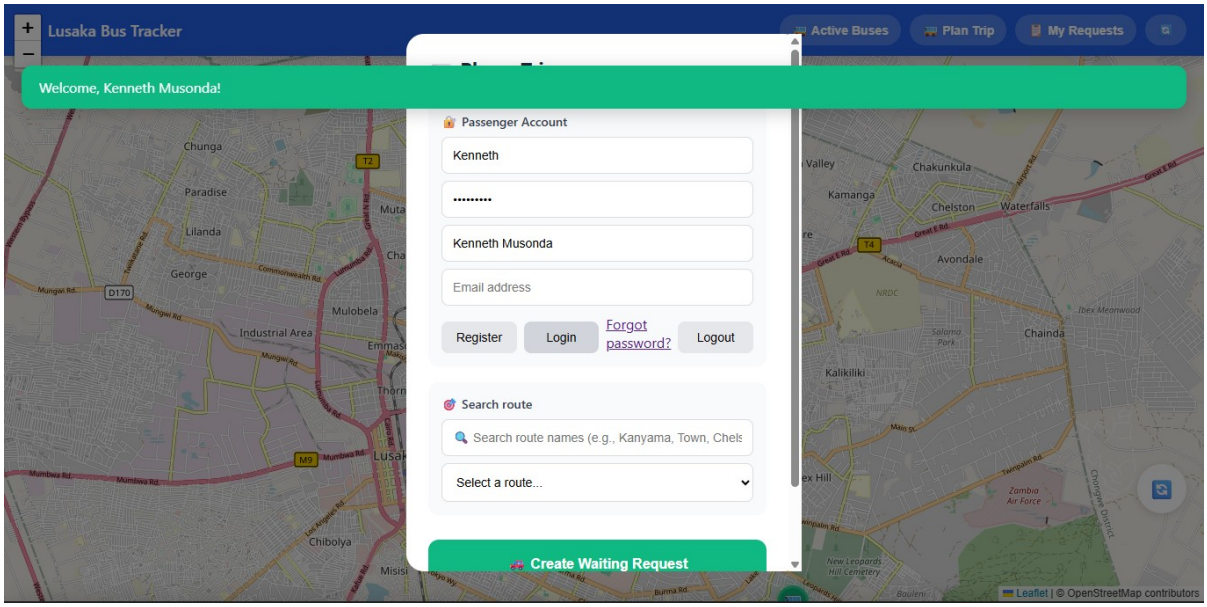


Figure 6.3.9: Passenger sign-in (interface developed by other group members)

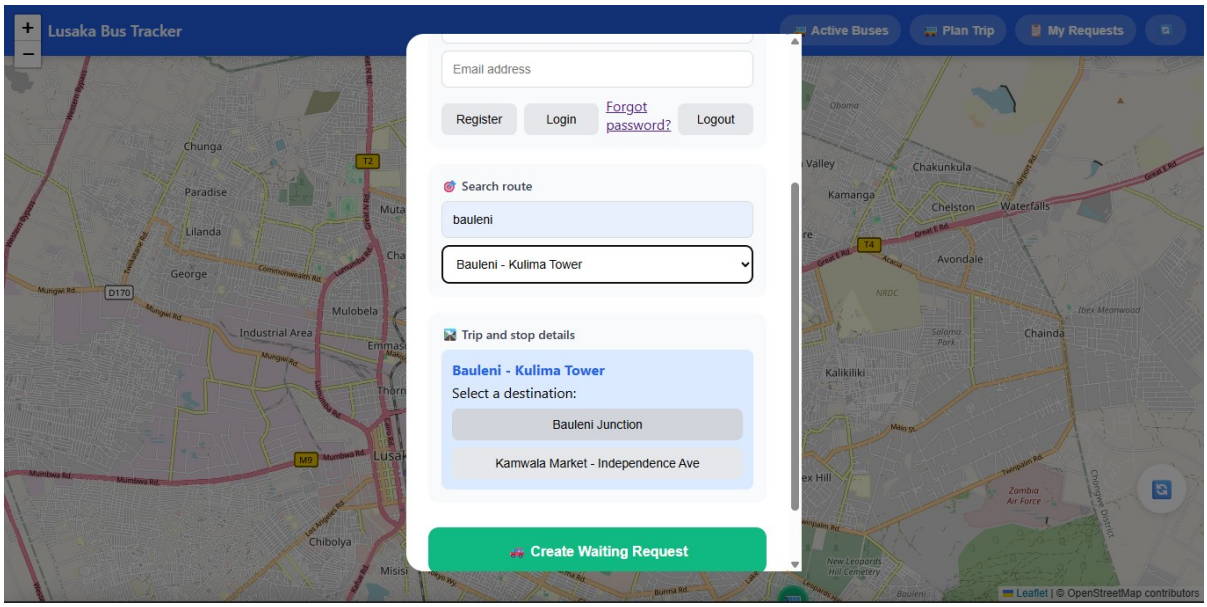


Figure 6.3.10: Route search and destinations offered (interface developed by other group members)

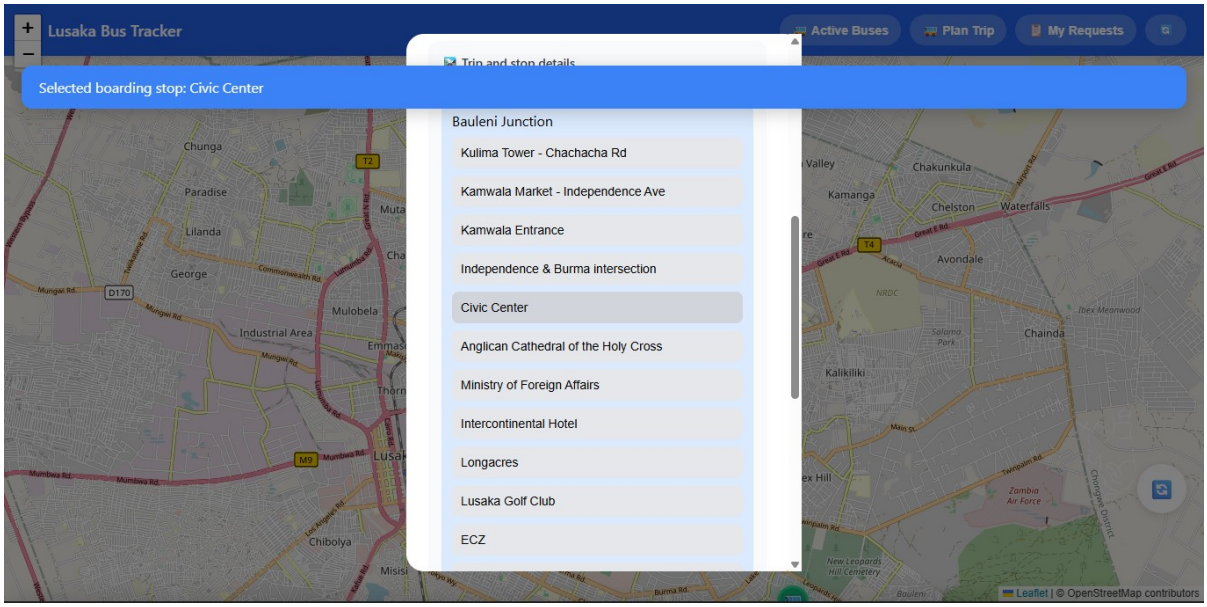


Figure 6.3.11: Boarding-stop selection (interface developed by other group members)

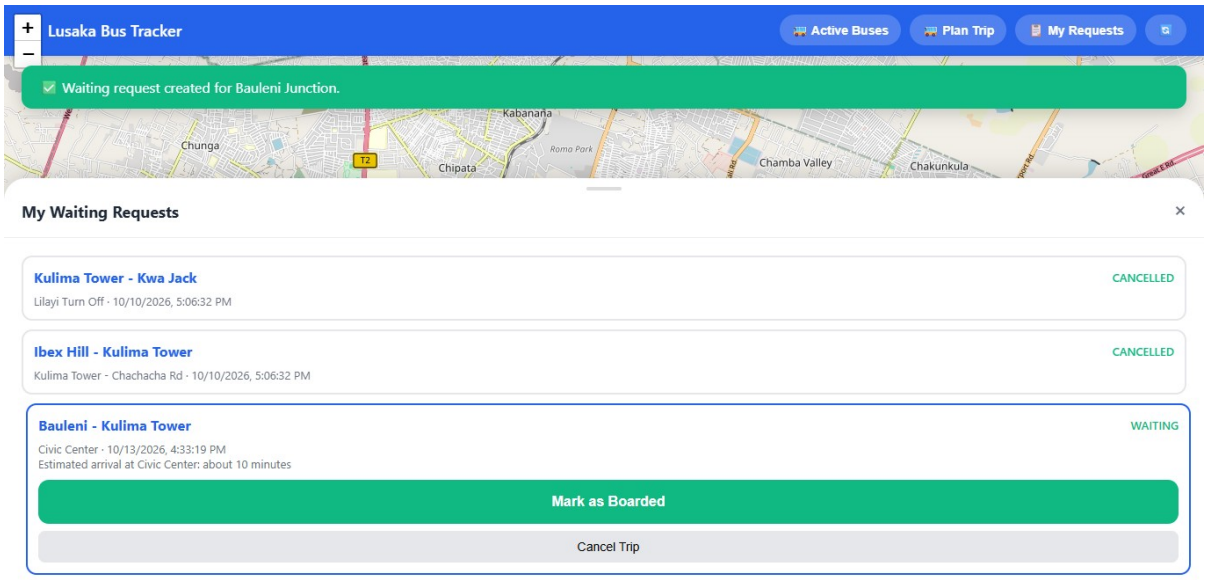


Figure 6.3.12: Waiting request created, with the arrival estimate and the options to board or cancel (interface developed by other group members)

Passenger request seen by the driver. Figure 6.3.13 shows both interfaces together: the passenger's waiting request on the left and, on the right, the driver dashboard listing one passenger waiting at Civic Center, the stop chosen by the passenger. **Operator management.** Figure 6.3.14 shows the Django administration site of the deployed system, which lists the bus, driver assignment, driver, passenger, route, stop and waiting-request records that the operator can manage.

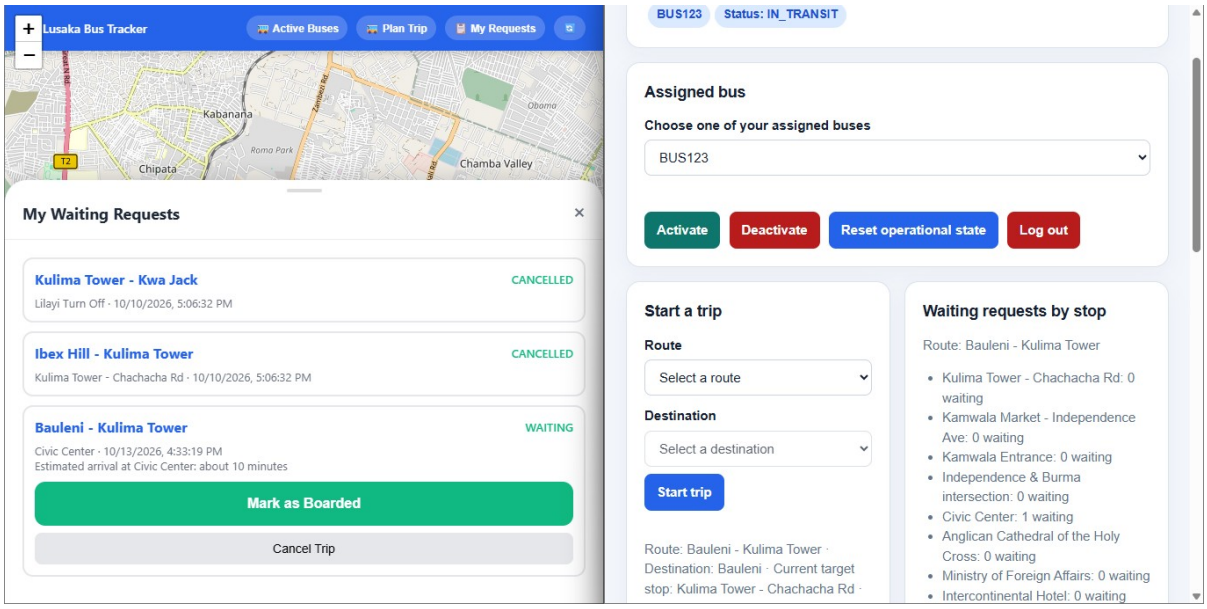


Figure 6.3.13: Passenger waiting request (left) and the driver dashboard showing one passenger waiting at Civic Center (right) (interface developed by other group members)

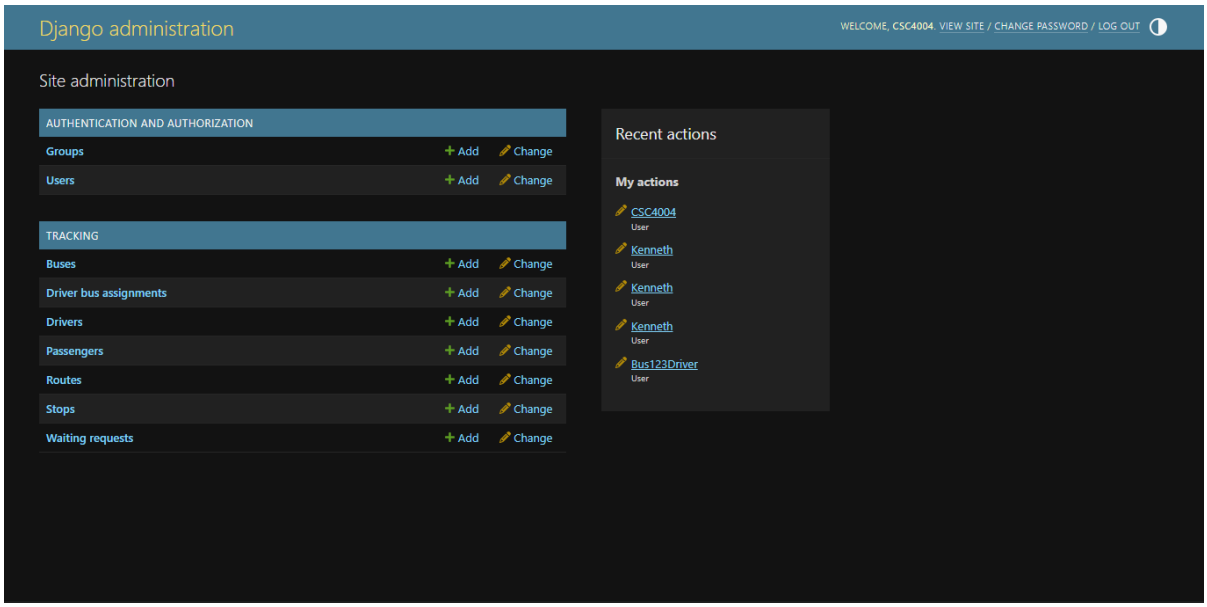


Figure 6.3.14: Django administration site of the deployed system

6.4 Experimental Results / Performance Results

No performance experiment was carried out. There is no response-time benchmark, concurrency test, update-rate measurement or load test, so the proposal's planned targets (location updates within 3 seconds, ten or more buses, page loads under 4 seconds) are neither met nor shown to fail. The code review does show that one GPS update can call OSRM up to twice, once for the departure check and once for the status, and that the OSRM request has no timeout.

6.5 User Evaluation

No user evaluation was conducted. The proposal planned user acceptance testing with five or more users and a System Usability Scale score of at least 70. Neither was done, and no usability claim is made.

6.6 Analysis of Results

The automated tests give moderate confidence in the ETA rules, the waiting-overview counts and the access rules as specified by the fixtures, and the manual checks of the deployed flows add some confidence that the server, the passenger interface, the driver dashboard and the deployment work together. They do not show that ETA values are accurate for real buses, because the routing results are mocked and no observed arrival times were collected. The limitations of the project itself, such as the absence of user, field and performance evidence, are given in Section 1.7. Table 6.6.1 lists the defects and technical limitations found by reviewing the code. The severity is an assessment made from the code review. Items L1 and L2 were fixed in the final stage and are covered by the access-control tests of Table 6.3.1; the other items are open.

Table 6.6.1: Defects and technical limitations identified in the system

ID	Finding	Impact	Severity and status
L1	Earlier in the project the Bus, Route, Stop and Passenger viewsets had no permission class and the settings defined no default, so Django REST Framework allowed unrestricted access [14]	Records could be read and modified without signing in, including passenger names, usernames and e-mail addresses	High; resolved. The default is IsAuthenticated, the transport viewsets are read-only and the passenger endpoint is removed (Section 5.4.3)
L2	The `expire` endpoint was open to anyone, and the waiting-overview endpoint did not check the driver	Anyone could end any WAITING request or read the waiting counts of a bus	Medium; resolved. `expire` is limited to administrators and the overview needs an assigned driver. Route, stop, bus, ETA and waiting-count reads remain public by design
L3	GPS latitude, longitude and speed are stored without validation of type or range	Invalid values can be stored; the ETA ignores invalid coordinates but other consumers would see them	Medium; open
L4	A destination with no matching trip raises an unhandled error in the start-trip view	A server error instead of a clear client error	Low; open
L5	The OSRM request has no timeout; status and departure services catch only request exceptions; the bus-ETA endpoint does not catch routing errors; one GPS update can call OSRM twice	Slow or malformed routing replies can delay or break a request	Medium; open
L6	Only 136 representative trips (08:00:00 variants) are used; the time of day is not modelled; shape data is unused	The system models route and direction, not a timetable	Low; open
L7	The 35 automated tests do not cover the GTFS import commands, the status and next-stop thresholds, validation of GPS values or a real OSRM instance, which is mocked	Faults in those parts would not be detected automatically	Medium; open
L8	Only the current state of a bus is stored	No history for audits or analysis	Low; open
L9	The thresholds of 30 for distance and 20 for speed are fixed values, and the speed unit is not specified in the code	Status and next-stop decisions are not calibrated or documented against field data	Medium; open
L10	Sign-in throttling allows five attempts per minute for each client address and covers passenger and driver sign-in only. The settings do not state how many proxies are in front of the application, and the counters are kept in each server process's own memory. Registration is not throttled and reports whether a username or e-mail address is already registered	Guessing and bulk account creation are slowed but not reliably prevented	Medium; open

Findings L1 and L2 were the most serious, because they gave anonymous users a second route to the data that driver authorisation protects. They were resolved in the final stage, and the default permission now makes every endpoint require sign-in unless it explicitly allows anonymous access. The main open security point is L10. The table describes the final version of the system.

6.7 Objective-by-Objective Evaluation

Table 6.7.1: Evaluation of the objectives

Objective	Evidence	Status
Group objective 1: Real-Time Bus Tracking from real GPS	Driver dashboard sends the device position to the authorised update endpoint, which stores bus state and status (Sections 4.8, 5.4.3, 5.4.5); Figures 6.3.5 to 6.3.8 show activation, trip start, the accepted GPS update and the resulting marker on the passenger map; no field trial	Implemented; not field-tested
Group objective 2: passenger interface to view buses and routes	Passenger page shows bus markers on a Leaflet map with registration, status and speed, refreshed every 15 seconds, and lists routes, destinations and stops (Section 4.8); Figures 6.3.8 to 6.3.11 show the marker, the route search, the destinations and the stops; bus locations are viewed as map markers and routes as lists	Met at prototype level
Group objective 3: Arrival Time Prediction	ETA algorithm and automated tests of its rules (Sections 4.9.5, 6.3); Figure 6.3.12 shows an estimate displayed to a passenger; accuracy not evaluated	Implemented; accuracy not established
Group objective 4: operators monitor and manage the fleet	Django administration site (Figure 6.3.14); no separate custom dashboard	Met at prototype level
Group objective 5: drivers see passengers at pickup points	Waiting-overview endpoint returns counts per stop and the driver dashboard displays them (Section 4.8); endpoint covered by automated tests; Figure 6.3.13 shows the dashboard listing a waiting passenger	Met at prototype level
Individual objective 1: data model and GTFS import	Models, migrations and five import commands (Sections 5.4.1, 5.4.2)	Achieved
Individual objective 2: driver authentication and assignment checks	Section 5.4.3; default permission, read-only model endpoints and administrator-only expiry (L1, L2 resolved)	Achieved; sign-in hardening gaps remain (L10)
Individual objective 3: bus operations and trip start	Section 5.4.4; error handling gap L4	Achieved
Individual objective 4: GPS handling, status and next stop	Section 5.4.5; validation gap L3 and calibration gap L9	Achieved at prototype level
Individual objective 5: waiting requests and ETA	Sections 5.4.6 to 5.4.8 and Table 6.3.1	Achieved
Individual objective 6: tests and deployment configuration	35 automated tests, with gaps listed in L7; environment-based configuration verified in the settings	Partly achieved

6.8 Individual Results

The results attributable to myself are the implemented and reviewed server-side and deployment components listed in Table 5.7.1; the browser interfaces evaluated above were developed by the other group members.

6.9 Chapter Summary

The available evidence supports the correctness of the server's ETA rules, waiting counts and access rules under the tested conditions, and shows by manual checks that the passenger interface, driver dashboard and server work together on the deployed system. It does not support claims about accuracy, performance, usability or field operation. The main limitations of the project are the absence of user, field and performance evidence and the open technical items in Table 6.6.1, and they are carried into the discussion.

CHAPTER 7: DISCUSSION

7.1 Introduction

This chapter interprets the findings in relation to the problem, the objectives, the literature and the limits of the evidence.

7.2 Interpretation of Findings

The results of Chapter 6 are interpreted below in relation to the problem stated in Section 1.2 and the objectives of Section 1.4, within the prototype setting. Table 6.7.1 gives the evidence for each objective.

Real-Time Bus Tracking from real GPS updates. A driver's device sends latitude, longitude and speed to an authenticated endpoint. The helper `get_authorized_bus_for_driver` accepts the update only from an active driver with an active assignment to the bus. The system stores the values, updates the next stop and derives the status, so each bus carries a current trip, current stop, position, speed and status. The method works for the implemented design, but the thresholds used have not been calibrated against field data, and the values are not validated at entry.

Journey planning with GTFS data. The passenger interface offers this flow on top of the imported data. Imported routes, trips, stops and ordered stop times let a passenger search a route, choose a destination and see the stops of a trip in order, then register at a boarding stop. Using representative trips makes selection unambiguous. It does so by treating a route and a direction as the unit of planning, not a timetable.

Arrival time from bus state and road durations. The ETA is the shortest road duration, from OSRM, among buses that are active, on the same trip, with valid coordinates and not yet past the stop. The system returns null when no such bus exists or when routing fails, instead of an invented value. Whether the numbers match real arrival times is not known.

Protecting operations and data. Driver operations are protected by the assignment check, and the passenger's waiting-request endpoints are scoped to the owner; the interfaces sign users in through the same endpoints. In the final branch the default permission requires sign-in, the transport viewsets are read-only, the passenger directory is removed and expiry is limited to administrators. Sign-in is rate-limited, but the limit has weaknesses (L10), and no independent

security review was carried out. The objective is therefore met for the implemented controls, not as a security guarantee.

Waiting passengers and fleet management. The driver dashboard shows the number of waiting passengers at each stop of the trip, taken from the waiting-overview endpoint, which only the assigned driver may call. Operators manage buses, drivers, assignments and transport records through the Django administration site. No separate operator dashboard was built.

7.3 Comparison with Existing Work

The system follows the conceptual flow found in the literature: positions are sent to a central server, combined with route and stop information, and turned into passenger information [2]. Like OneBusAway it presents next-bus style information, but it is far smaller; it has one passenger-facing flow, no multichannel delivery and no study of its effect on waiting time [3]. The caution of Liu and Miller, that real-time information does not automatically reduce waiting time, applies equally: this project shows that the information can be produced, not that it improves passengers' waiting [5]. In relation to Rea Vaya, the project is a lightweight prototype with none of the infrastructure of a BRT system [9]. In relation to GTFS, the project adopts the static data model for routes, trips and stops and leaves GTFS Realtime for the future [7], [6].

7.4 Contributions of the Project

The group produced an integrated prototype for Lusaka routes: a passenger interface with a live map, journey planning and waiting requests, a driver dashboard with GPS reporting and passenger counts, and a server with the data, ETA logic, access control and deployment configuration behind them. Appendix A, is the server side and the deployment configuration: the data model, the GTFS import, driver authorisation, real GPS handling, trip and stop state, waiting requests, OSRM-based ETA with defensive handling, automated tests for ETA, waiting overview and access control, access-control hardening, and environment-based configuration. These provide a verified base for later work.

7.5 Limitations and Implications

The limitations in Section 1.7 and Table 6.6.1 determine how the results may be used. The principal limitation was the absence of an operational bus fleet and an official operator data feed, so the GPS positions could not be verified as those of an operating bus and live service changes could not be received. In short, the system could be verified against its design but not validated against an operating bus service. The findings apply to a prototype verified functionally, with synthetic fixtures and mocked routing in the automated tests. They do not apply to a city-wide, continuously operating service. Without user, field and performance evidence, no claim can be made about ETA accuracy, usability or the effect on passengers, and the thresholds and the ETA method must be evaluated with observed data first. The GTFS feed supplied is a snapshot with a stated validity period, and keeping data current is an operational task that the project did not address. The browser interfaces were verified only by manual checks, not by automated tests. Technically, the open items in Table 6.6.1 and an independent security review should be dealt with before real users' data are handled.

7.6 Chapter Summary

The objectives were met at prototype level, with the protection of operations and data only partly satisfied. The results are consistent with the literature's view that information can be provided but its benefit must be shown empirically.

CHAPTER 8: CONCLUSION AND RECOMMENDATIONS

8.1 Introduction

This chapter concludes the report, states whether the objectives were achieved and gives recommendations and future work.

8.2 Summary of the Project

The project set out to implement a web-based real-time public bus tracking system for Lusaka. It started from a proposal that simulated GPS data. On supervisor feedback, it was changed to accept real GPS coordinates and speed from a driver's device. The group followed an iterative and incremental process and produced a passenger interface with a live map, journey planning and waiting requests; a driver dashboard that reports the device's GPS position and shows waiting passengers; and a Django and Django REST Framework server with the data model, GTFS imports, driver authorisation, real GPS handling, waiting requests, OSRM integration and ETA logic, configured for deployment. The system was verified by automated tests of the server's ETA, waiting-overview and access-control behaviour, by manual checks of the deployed interfaces, and by code review.

8.3 Conclusions

- A Django application can maintain the operational state of buses from authorised real GPS updates and relate it to imported GTFS routes, trips and stops (group objectives 1 and 2).
- A conditional, road-based ETA can be produced from this state and an external routing service, and it can decline to estimate when the data is missing or invalid (group objective 3).
- Driver operations are protected by an active driver and an active assignment, and the final branch requires sign-in by default, so protection is implemented at prototype level but has not been independently tested (individual objective 2).
- The three intended features, Real-Time Bus Tracking, Journey Planning and Arrival Time Prediction, are implemented at prototype level across the passenger interface, driver dashboard and server, with bus locations viewed as map markers and routes as lists.
- The project does not show city-wide operation, ETA accuracy, usability, user acceptance or performance.

8.4 Achievement of Objectives

The system objectives were met at prototype level for real-GPS tracking, arrival-time prediction and waiting counts for drivers. The passenger interface objective was met, with bus locations viewed as map markers and routes as lists. The operator objective was met through the administration site. . Table 6.7.1 gives the evidence for each objective.

8.5 Recommendations

1. Resolve the open security item (L10): state the number of trusted proxies, keep throttle counters in a shared cache, and throttle registration and stop revealing which usernames exist; then arrange an independent security review.
2. Validate GPS input for type and range, handle the start-trip error cleanly, and set timeouts and error handling for every OSRM call (L3 to L5).
3. Extend automated tests to the GTFS imports, the status and next-stop thresholds and GPS value validation, and add an integration test against a real OSRM instance (L7).
4. Document the unit of the speed value and calibrate the status thresholds with field data (L9).
5. Before any operational use, establish privacy, consent and data-governance policies, verify the licence of the GTFS data, and add audit logging.

8.6 Future Work

Future work may include: a historical location table for analysis; time-of-day schedules using all GTFS trips; use of the shape data for map-matching and route display; live-traffic-aware ETA; publication of a GTFS Realtime feed; user research and usability evaluation with passengers and drivers; field trials with measured ETA accuracy; performance and load testing; notifications; and an operator dashboard that shows which buses that are facing challenges such as a breakdown. These are possible extensions and are not features already delivered.

REFERENCES

[1] A. Fernando, K. Francisco, and M. Guillermo, “Intelligent transportation systems for sustainable urban mobility: A systematic literature review of research and applications in public transport,” *Urban Sci.*, vol. 10, no. 7, p. 380, 2026.

[2] S. K. A. Khalid, N. S. Mohd Salleh, and N. A. Samsudin, “A bus tracking information system using consumer grade GPS: A case study,” *J. Telecommun. Electron. Comput. Eng.*, vol. 8, no. 4, pp. 47–51, 2016.

[3] K. E. Watkins, B. Ferris, A. Borning, G. S. Rutherford, and D. Layton, “Where is my bus? Impact of mobile real-time information on the perceived and actual wait time of transit riders,” *Transp. Res. A, Policy Pract.*, vol. 45, no. 8, pp. 839–848, 2011.

[4] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[5] L. Liu and H. J. Miller, “Does real-time transit information reduce waiting time? An empirical analysis,” *Transp. Res. A, Policy Pract.*, vol. 141, pp. 167–179, 2020.

[6] GTFS, “GTFS Realtime,” General Transit Feed Specification Documentation, gtfs.org. Accessed: Jun. 16, 2026. [Online]. Available: https://gtfs.org/documentation/realtime/

[7] GTFS, “General Transit Feed Specification overview,” General Transit Feed Specification Documentation, gtfs.org. Accessed: Jun. 16, 2026. [Online]. Available: https://gtfs.org/documentation/overview/

[8] Project OSRM, “OSRM API documentation,” version 5.24.0. [Online]. Available: https://project-osrm.org/docs/v5.24.0/api/

[9] E. Adewumi and D. Allopi, “Rea Vaya: South Africa’s first bus rapid transit system,” *S. Afr. J. Sci.*, vol. 109, nos. 7/8, 2013.

[10] GSMA Intelligence, “The mobile economy Sub-Saharan Africa 2023,” GSMA, London, U.K., 2023.

[11] DigitalTransport4Africa, “Lusaka city public transport GTFS feed (feed version lusaka_city_pt, valid 2023-09-01 to 2024-12-30).” Accessed: Jun. 16, 2026. [Online]. Available: https://digitaltransport4africa.org/

[12] Django Software Foundation, “Django documentation, version 6.0.” [Online]. Available: https://docs.djangoproject.com/en/6.0/

[13] Django Software Foundation, “Writing and running tests,” Django documentation, version 6.0. [Online]. Available: https://docs.djangoproject.com/en/6.0/topics/testing/overview/

[14] Django REST framework, “Permissions,” API Guide. [Online]. Available: https://www.django-rest-framework.org/api-guide/permissions/

APPENDICES

APPENDIX A: INDIVIDUAL CONTRIBUTION STATEMENT (GROUP PROJECTS)

Purpose of this Statement

This statement documents the individual contribution of Kenneth Kaunda Musonda to the group project. Although the project was conceived, analysed and integrated collaboratively, this report presents individual work, and the statement is consistent with Chapters 5 and 6. The work of other group members is identified as theirs and is not claimed.

Project and Group Information

Table A.1: Project and group information

Item	Detail
Project Title	Implementation of a Web-Based Real-Time Public Bus Tracking System for Lusaka
Student Name	Kenneth Kaunda Musonda
Computer Number	2020056283
Supervisor	Dr. Lighton Phiri
Group Size	3 students
Group Members	Madalitso Nyirenda (2021502597); Kenneth Kaunda Musonda (2020056283); Ngosa Muleba (2022044371)

Overall Group Project

The group built the Bus Tracking System, a web-based system that lets passengers in Lusaka follow buses, plan a journey on predefined routes and obtain an arrival-time estimate, and lets authorised drivers operate their buses and report GPS positions. The common problem is the lack of route-specific, real-time information for passengers. The group's solution is an integrated web application made of a Django backend, GTFS-derived route data, an OSRM routing service and browser interfaces. The proposal planned simulated GPS data; after supervisor feedback the system accepts real GPS updates from a driver's device.

Individual Role and Responsibilities

The role was that of backend developer. The entire backend of the system and its deployment configuration were the responsibility of this role. The frontend was not part of it.

Detailed Individual Contribution

The following were implemented, and all are described in Chapters 4 to 6.

- the Django and Django REST Framework backend, the models and the migrations;

- the GTFS import commands for routes, stops, trips, stop times and shapes;
- the APIs for authentication, routes, destinations, trip stops, bus operations, GPS updates, waiting requests, waiting counts and ETA;
- driver authentication and authorisation, and the driver-to-bus assignment check;
- bus activation, deactivation, operational reset and trip start;
- real GPS update handling, bus status, next-stop and departure logic;
- waiting-request services and their lifecycle;
- OSRM integration and the ETA logic;
- the automated tests; and
- the deployment configuration, including environment-based settings.

Individual Deliverables

1. **Backend application** — the Django project and the `tracking` application in the integration branch.
2. **GTFS import commands** — five management commands that load the GTFS files into the database.
3. **API and service layer** — endpoints and service modules for driver operations, GPS updates, waiting requests and ETA.
4. **Automated tests** — tests for the ETA, the waiting-passenger overview, access control, registration, login throttling and the debug default.
5. **Deployment configuration** — environment-based settings and the WSGI entry point.
6. **Analysis and design artefacts in this report** — use-case diagram, system sequence diagrams, architecture diagram, domain model, ER diagram and design model, all drawn to match the implemented code.

Contribution to the Main System or Research Output

The backend is the component of the final system that was developed individually. It stores the data, enforces who may operate a bus, receives and interprets GPS updates, manages passenger waiting requests and computes arrival-time estimates. The browser interfaces, which call this backend, were developed jointly by the other group members and are not claimed.

Individual Implementation and Technical Work

The technical work consisted of ten database entities with nine migrations; five GTFS import commands; nine service modules; around thirty request handlers and viewsets; the authorisation helper; the OSRM distance and ETA logic with defensive error handling; and the settings that read secrets and addresses from the environment. Chapter 5 describes each in detail.

Individual Testing and Evaluation Contribution

The automated tests for ETA, waiting-overview and access-control behaviour were written (Table 6.3.1), and the code was reviewed to identify the limitations in Table 6.6.1. The manual functional and integration checks of the deployed passenger and driver flows (Section 6.3) relied on the backend that was implemented; the contribution to them was that backend support, not the interfaces, which other members built. No user testing, field trial or performance measurement was conducted, and the report makes no claim about them.

Collaboration with Other Group Members

The backend serves the interfaces built by the other group members, so its endpoint behaviour was a dependency for their screens. The problem analysis, the Yango interface analysis and the literature review were shared, and the literature review was adapted for this report. The contribution of Madalitso Nyirenda and Ngosa Muleba to those shared analyses and to the interfaces that make the backend usable is acknowledged, as is the supervisor's review feedback, which led to the replacement of simulated GPS with real GPS.

Contribution Summary

(1) **Work primarily completed individually:** the backend, GTFS imports, driver authorisation, bus and trip operations, GPS handling, waiting-request services, OSRM integration, ETA logic, automated tests, deployment configuration, and the analysis and design diagrams of this report.

(2) **Work completed jointly with other group members:** the problem analysis, requirements analysis, Yango interface analysis and the literature review.

(3) **Work primarily completed by other group members but used in or alongside the individual work:** the browser interfaces, including the map display and the passenger and driver screens, whose screenshots appear in Section 4.8 and in Section 6.3 as interface evidence.

Estimated Contribution

The template asks for approximate percentages. No time or effort records exist from which percentages could be derived without inventing them, so Table A.2 states the nature of the contribution to each activity.

Table A.2: Nature of the individual contribution by activity

Activity	Individual	Collaborative
Problem Analysis		
Literature Review		
Requirements Analysis		
System/Research Design		
Implementation/Development		
Testing and Evaluation		

Activity	Individual	Collaborative
Documentation		
Other	—	—

Statement of Authorship and Contribution

I confirm that the contribution described in this statement accurately represents the work that I personally undertook as part of the group project. I have identified collaborative work and have not knowingly claimed the work of another group member as my own.

Student Name: Kenneth Kaunda Musonda

Computer Number: 2020056283

Signature: _____

Date: _____

Supervisor Verification

I confirm that, to the best of my knowledge, the contribution described above is consistent with the student's participation in the group project.

Supervisor Name: Dr. Lighton Phiri

Signature: _____

Date: _____

APPENDIX B: USER MANUAL / INSTALLATION GUIDE

Running the system and loading the GTFS data

The commands below are the Django management commands defined in the project. They are run from the directory that contains `manage.py`, after the dependencies in `requirements.txt` are installed.

```
python manage.py migrate
python manage.py import_routes
python manage.py import_stops
python manage.py import_trips
python manage.py import_stop_times
python manage.py import_shapes
python manage.py runserver
python manage.py check        # configuration check only
python manage.py test         # runs the automated tests
```

Each import command prints the numbers of rows imported, updated and skipped. Routes and stops must be imported before trips, and trips before stop times. In production the settings expect the environment variables listed in Table F.2 in Appendix F.

APPENDIX C: ADDITIONAL SYSTEM DESIGNS / DIAGRAMS

No additional system designs or diagrams are included. The diagrams of the implemented system are in the main chapters of the report.

APPENDIX D: ADDITIONAL TEST CASES AND RESULTS

The tests are in `tracking/tests.py`. The names below are those of the test methods in the final version of the system, which defines 35 tests in five classes. A run of `python manage.py test` found 35 tests and all passed (Section 6.3, Figure 6.3.1). The classes are explained first and then listed in Table D.1.

WaitingRequestEtaTests (6 tests) check the arrival-time estimate for a waiting request. They verify that the shortest valid OSRM duration is used when several suitable buses are available, and that no estimate is produced when a bus has no GPS coordinates, when a bus has already passed the requested stop, when the request is not in the WAITING state, or when OSRM returns no usable route. They also verify that a passenger cannot read the ETA of another passenger's request.

WaitingOverviewTests (3 tests) check the waiting-passenger counts shown to a driver. They verify that a request is counted only at its own boarding stop, that requests which are not waiting or belong to another trip are excluded, and that a bus without an active trip returns an empty list of stops. The test data uses a stop named Bauleni station as its example.

ApiAccessSecurityTests (20 tests) check who may call which endpoint. They verify that routes, stops and buses are read-only and public, that the passenger directory no longer exists, that driver operations need an active driver with an active assignment, that only an administrator can expire a request, that registration applies password validators, and that a passenger can neither change a request's lifecycle fields directly nor act on another passenger's request.

LoginThrottleTests (4 tests) check that passenger and driver sign-in are limited to five attempts per minute for each client address, with separate counters, and that registration is not throttled. **DebugDefaultTests** (2 tests) check that debug mode is off unless it is explicitly enabled.

Table D.1: Test methods

Test class	Test method	What it verifies
WaitingRequestEtaTests	test_returns_smallest_osrm_duration_for_eligible_buses	With several eligible buses, the ETA is the shortest valid OSRM duration.
WaitingRequestEtaTests	test_returns_unavailable_when_no_bus_has_gps	No ETA is returned when no bus has GPS coordinates.
WaitingRequestEtaTests	test_excludes_bus_that_has_passed_the_requested_stop	A bus that has already passed the boarding stop is not used.
WaitingRequestEtaTests	test_non_waiting_requests_have_no_eta	Only a request in the WAITING state receives an ETA.
WaitingRequestEtaTests	test_another_passenger_cannot_read_the_request_eta	A different passenger asking for the ETA is refused.
WaitingRequestEtaTests	test_missing_osrm_route_returns_unavailable	No ETA is returned when OSRM gives no usable route.
WaitingOverviewTests	test_counts_bauleni_boarding_request_only_at_bauleni	A waiting request is counted only at its own boarding stop.

Test class	Test method	What it verifies
WaitingOverviewTests	test_excludes_non_waiting_requests_and_other_trips	Requests that are not waiting, or that belong to another trip, are not counted.
WaitingOverviewTests	test_no_active_trip_returns_no_stops	A bus with no active trip returns an empty list of stops.
ApiAccessSecurityTests	test_generic_transport_writes_are_disabled	Creating, replacing, changing or deleting routes, stops and buses returns 405 for anonymous, passenger and driver users, and the records remain.
ApiAccessSecurityTests	test_passenger_directory_is_removed	The passenger list and detail endpoints return 404.
ApiAccessSecurityTests	test_public_transport_reads_remain_available	Route, stop and bus reads, route search, destinations, trip stops, bus ETA and waiting count return 200 without signing in.
ApiAccessSecurityTests	test_registration_and_login_endpoints_are_public	Registration, passenger sign-in and driver sign-in work without an existing session.
ApiAccessSecurityTests	test_assigned_driver_can_update_gps	An assigned driver's GPS update is accepted and stored on the bus.
ApiAccessSecurityTests	test_assigned_driver_can_operate_bus	An assigned driver can activate, reset and deactivate the bus and start a trip.
ApiAccessSecurityTests	test_driver_operations_require_active_driver_and_assignment	Anonymous users, passengers, unassigned drivers, inactive drivers and inactive assignments receive 403 on bus operations and the waiting overview.
ApiAccessSecurityTests	test_expiry_requires_administrator	Only an administrator can expire a request; all other users receive 403 and the request stays WAITING.
ApiAccessSecurityTests	test_default_permission_requires_authentication	The default permission class is IsAuthenticated, and anonymous requests to protected endpoints return 403.
ApiAccessSecurityTests	test_registration_rejects_invalid_passwords_without_creating_accounts	Weak passwords are rejected with 400 and no user or passenger is created.
ApiAccessSecurityTests	test_registration_uses_configured_password_validators	Registration applies the validators in the settings, shown by a stricter minimum length being enforced.
ApiAccessSecurityTests	test_owner_cannot_update_or_delete_waiting_request_directly	Update and delete of a waiting request return 405, and its status and timestamps do not change.
ApiAccessSecurityTests	test_creation_ignores_client_supplied_lifecycle_fields	A new request is WAITING and belongs to the signed-in passenger, whatever status, passenger or timestamps the client sends.
ApiAccessSecurityTests	test_owner_can_board_and_complete_through_workflow_actions	The owner can list and read the request and move it from WAITING to ON_BOARD to COMPLETED through the workflow actions.
ApiAccessSecurityTests	test_other_passenger_cannot_use_workflow_actions	Another passenger receives 404 when trying to board, cancel or complete the request, and its status is unchanged.
ApiAccessSecurityTests	test_anonymous_waiting_request_paths_return_forbidden	Anonymous users receive 403 on every waiting-request path and nothing changes.

Test class	Test method	What it verifies
ApiAccessSecurityTests	test_non_passenger_waiting_request_paths_return_forbidden	Drivers and users without a passenger profile receive 403 on every waiting-request path.
ApiAccessSecurityTests	test_waiting_request_reads_remain_owner_scoped	Another passenger sees an empty list and receives 404 for the request.
ApiAccessSecurityTests	test_viewset_profile_guards_also_protect_direct_method_calls	The viewset's own queryset and create methods refuse anonymous and non-passenger users when called directly.
ApiAccessSecurityTests	test_non_passenger_creation_is_forbidden_before_payload_validation	A driver posting to the waiting-request endpoint receives 403 before the payload is validated.
LoginThrottleTests	test_passenger_login_rate_limit	The sixth passenger sign-in attempt within a minute from one address returns 429 with a Retry-After of 60 seconds; another address is unaffected and the limit lifts after a minute.
LoginThrottleTests	test_driver_login_rate_limit	The same limit applies to driver sign-in.
LoginThrottleTests	test_login_limits_are_separate_and_registration_is_unthrottled	Passenger and driver sign-in have separate counters; registration and route search are not throttled.
LoginThrottleTests	test_authenticated_requests_are_also_limited_by_ip	Signed-in requests to the sign-in endpoints are also counted by address.
DebugDefaultTests	test_debug_is_false_when_environment_variable_is_absent	Debug mode is off when DJANGO_DEBUG is not set.
DebugDefaultTests	test_debug_can_be_explicitly_enabled_for_local_development	Debug mode is on only when DJANGO_DEBUG is set to True.

The tests therefore cover the ETA, the waiting overview, access control, registration, login throttling and the debug default. They do not cover the GTFS import commands, the status and next-stop thresholds, validation of GPS values or a real OSRM instance; this is limitation L7 in Table 6.6.1.

APPENDIX E: DATA COLLECTION INSTRUMENTS

Not applicable. No questionnaires, interviews or other data collection instruments were used, because no user study was conducted.

APPENDIX F: ADDITIONAL CODE / CONFIGURATION

This appendix gives selected configuration for reference: the API endpoints and the environment variables read by the application. The complete source code is in the project repository.

API endpoint reference

All routes are under `/api/`. "Auth" states whether the endpoint requires a signed-in user in the final version of the system. Unless an endpoint allows anonymous access, the default permission requires sign-in. The ViewSets for routes, stops and buses are read-only and public.

Table F.1: API endpoints.

Endpoint	Method	Auth	Purpose
auth/register/	POST	No	Register a passenger account
auth/login/	POST	No	Passenger sign-in (five attempts per minute per client address)
auth/me/	GET	Yes	Current passenger
auth/logout/	POST	Yes	Passenger sign-out
auth/driver/login/	POST	No	Driver sign-in (five attempts per minute per client address)
auth/driver/me/	GET	Yes	Current driver
auth/driver/buses/	GET	Yes	Buses actively assigned to the driver
auth/driver/logout/	POST	Yes	Driver sign-out
buses/<registration_number>/driver-session/	GET	Yes + assignment	Current trip and state of a bus
buses/<registration_number>/activate/	POST	Yes + assignment	Activate a bus
buses/<registration_number>/deactivate/	POST	Yes + assignment	Deactivate and reset a bus
buses/<registration_number>/start-fresh/	POST	Yes + assignment	Reset operational state
buses/start-trip/	POST	Yes + assignment	Start a trip for a route and destination
buses/<registration_number>/location/	POST	Yes + assignment	Submit a GPS update
buses/<registration_number>/waiting-overview/	GET	Yes + assignment	Waiting counts per stop of the bus's trip
buses/<id>/eta/<stop_id>/	GET	No	Road duration from a bus to a stop
routes/search/?q=	GET	No	Search routes by name
routes/<route_id>/destinations/	GET	No	Destinations of a route
routes/<route_id>/stops/?destination=	GET	No	Ordered stops of the trip to a destination
waiting-requests/	GET, POST	Yes (passenger)	List own requests; create a request

Endpoint	Method	Auth	Purpose
waiting-requests/<id>/ (detail)	GET	Yes (passenger)	Own request (ViewSet; update and delete disabled)
waiting-requests/<id>/board/	POST	Yes	Mark as boarded
waiting-requests/<id>/cancel/	POST	Yes	Cancel a request
waiting-requests/<id>/complete/	POST	Yes	Complete an on-board request
waiting-requests/<id>/expire/	POST	Administrator	Expire a WAITING request
waiting-requests/<id>/eta/	GET	Yes	ETA for own request
trips/<trip_id>/stops/<stop_id>/waiting-count/	GET	No	Waiting count for a trip and stop
routes/, stops/, buses/	GET	No	Read-only ViewSets

Outside `/api/`, the project routes the landing page, the passenger and driver pages, Django's password-reset views under `password-reset/` and the administration site under `admin/`.

Environment variables

Table F.2 lists the environment variables read by the application, with their purpose. Only names are given.

Table F.2: Environment variables read by the application

Variable	Purpose
DJANGO_SECRET_KEY	Django secret key; required when debugging is off
DJANGO_DEBUG	Debug mode; off unless set to true
DJANGO_ALLOWED_HOSTS	Comma-separated allowed hosts; defaults to localhost, 127.0.0.1 and testserver
DJANGO_CSRF_TRUSTED_ORIGINS	Comma-separated trusted origins for CSRF checks
DATABASE_URL	Address of the PostgreSQL database
OSRM_URL	Base URL of the OSRM service; defaults to http://localhost:5000
BREVO_API_KEY	Credential for the Brevo e-mail backend
DEFAULT_FROM_EMAIL	Sender address for e-mail

When debugging is off, the settings also trust the forwarded protocol header of the hosting proxy and mark session and CSRF cookies as secure. The build and start commands used on Render are given in Section 5.4.9.

APPENDIX G: OTHER SUPPORTING MATERIAL

None.