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

The Passenger Interface: Live Bus Map, Route Search, Boarding Stop Selection and Waiting Requests

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

Ngosa Muleba

Computer Number: 2022044371

Group members:

Kenneth Kaunda Musonda (Computer Number: 2020056283)
Madalitso Nyirenda (Computer Number: 2021502597)

Supervisor: Dr. Lighton Phiri

Date of Submission: [date to be confirmed]

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. The server, the deployment configuration and the driver dashboard were developed by my group members and are described in this report only so far as the passenger interface depends on them.

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 at each stop. The group built one integrated system made of a passenger interface, a driver dashboard and a Django REST Framework server.

This individual report concentrates on the passenger interface, which the author designed and implemented, and on the problem analysis and requirements that the group carried out jointly. The passenger interface is a browser page with four parts: a live map that shows the active buses and refreshes every 15 seconds; a route search that returns the routes matching a typed name; a trip-planning panel in which the passenger chooses a destination and a boarding stop from the ordered stops of the trip; and a waiting-request list in which the passenger registers that they are waiting, marks the request as boarded, cancels it or completes it, and sees the estimated arrival time of the bus when one is available.

The interface was checked by manual functional and integration checks on the deployed system, working together with the driver dashboard and the server. The server's automated tests of the arrival-time rules, the passenger-scoped waiting-request endpoints and the access rules support those checks. No user study, field trial or performance benchmark was carried out, so the report claims that the passenger interface is implemented at prototype level and works with the rest of the system, and makes no claim about usability, arrival-time accuracy or reduced waiting time.

Keywords: passenger information; bus tracking; Leaflet; route search; waiting request; 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. Kenneth Kaunda Musonda built the server and the deployment configuration on which the passenger interface depends, and Madalitso Nyirenda built the driver dashboard that supplies the bus positions the passenger interface displays. The problem analysis and requirements were prepared jointly by all three of us.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1
1.1 Background and Context 1
1.2 Problem Statement 1
1.3 Project Aim 1
1.4 Project Objectives 1
1.5 Scope of the Project 1
1.6 Significance of the Project 1
1.7 Limitations 1
1.8 Organisation of the Report 1
CHAPTER 2: LITERATURE REVIEW 1
2.1 Introduction 1
2.2 Key Concepts and Theories 1
2.3 Existing Systems / Related Work 1
2.4 Comparative Analysis 1
2.5 Research / Knowledge Gap 1
2.6 Conceptual or Theoretical Framework 1
2.7 Chapter Summary 1
CHAPTER 3: METHODOLOGY 1
3.1 Introduction 1
3.2 Research / Development Approach 1
3.3 Requirements / Data Collection 1
3.4 System Architecture / Research Workflow 1
3.5 Tools and Technologies 1
3.6 Ethical and Legal Considerations 1
3.7 Evaluation Method 1
3.8 Chapter Summary 1
CHAPTER 4: SYSTEM ANALYSIS AND DESIGN 1
4.1 Introduction 1
4.2 Stakeholder and User Analysis 1
4.3 Functional Requirements 1
4.4 Non-Functional Requirements 1
4.5 Use Cases / User Stories 1
4.6 System / Solution Architecture 1

4.7 Database / Data Design 1
4.8 Interface / Interaction Design 1
4.9 Algorithms / Models / Technical Design 1
4.10 Chapter Summary 1
CHAPTER 5: IMPLEMENTATION / DEVELOPMENT 1
5.1 Introduction 1
5.2 Development Environment 1
5.3 System Components / Modules 1
5.4 Key Implementation Details 1
5.5 Integration 1
5.6 Challenges and Solutions 1
5.7 Individual Contribution (Group Projects) 1
5.8 Chapter Summary 1
CHAPTER 6: TESTING, RESULTS AND EVALUATION 1
6.1 Introduction 1
6.2 Testing Strategy 1
6.3 Test Cases and Results 1
6.4 Experimental Results / Performance Results 1
6.5 User Evaluation 1
6.6 Analysis of Results 1
6.7 Objective-by-Objective Evaluation 1
6.8 Individual Results 1
6.9 Chapter Summary 1
CHAPTER 7: DISCUSSION 1
7.1 Introduction 1
7.2 Interpretation of Findings 1
7.3 Comparison with Existing Work 1
7.4 Contributions of the Project 1
7.5 Limitations and Implications 1
7.6 Chapter Summary 1
CHAPTER 8: CONCLUSION AND RECOMMENDATIONS 1
8.1 Introduction 1
8.2 Summary of the Project 1
8.3 Conclusions 1

8.4 Achievement of Objectives....1
8.5 Recommendations....1
8.6 Future Work....1
REFERENCES....1
APPENDICES....1
APPENDIX A: INDIVIDUAL CONTRIBUTION STATEMENT (GROUP PROJECTS)....1
APPENDIX B: USER MANUAL — THE PASSENGER INTERFACE....1
APPENDIX C: ADDITIONAL SYSTEM DESIGNS / DIAGRAMS....1
APPENDIX D: ADDITIONAL TEST CASES AND RESULTS....1
APPENDIX E: DATA COLLECTION INSTRUMENTS....1
APPENDIX F: ADDITIONAL CODE / CONFIGURATION....1
APPENDIX G: OTHER SUPPORTING MATERIAL....1

The table of contents is generated by Word. If page numbers are missing, right-click the table and choose Update Field.

LIST OF FIGURES

Figure	Title
Figure 2.6.1	Conceptual framework derived from the reviewed literature
Figure 3.2.1.1	Iterative and incremental development process of the Bus Tracking System
Figure 4.5.1.1	UML use-case diagram of the Bus Tracking System
Figure 4.5.3.1	System sequence diagram: passenger plans a journey, creates a waiting request and retrieves the ETA
Figure 4.6.1	System architecture of the Bus Tracking System
Figure 4.7.1.1	Domain model of the Bus Tracking System (conceptual)
Figure 4.8.1	Landing page of the deployed system
Figure 4.8.2	Passenger map with the header controls and an active bus marker
Figure 4.8.3	Passenger interface screens for route search, boarding-stop selection and waiting-request creation
Figure 4.8.4	A waiting request in the WAITING state
Figure 4.8.5	A waiting request in the ON_BOARD state
Figure 6.3.1	Passenger sign-in
Figure 6.3.2	Route search and destinations offered
Figure 6.3.3	Boarding-stop selection
Figure 6.3.4	Waiting request created, with the arrival estimate and the options to board or cancel
Figure 6.3.5	Passenger waiting request (left) and the driver dashboard showing one passenger waiting at Civic Center (right)
Figure C.1	Entity-relationship diagram of the persisted data

LIST OF TABLES

Table	Title
Table 1.4.1	Proposal objectives and the passenger-side objectives of the final project
Table 1.5.1	Scope of the project as implemented
Table 2.4.1	Comparison of reviewed systems and standards with the Bus Tracking System
Table 3.2.2.1	Stages of development and the passenger-interface work in each stage
Table 3.2.3.1	Proposal compared with the implemented passenger interface
Table 3.3.3.1	Adaptation of ride-hailing interaction ideas to a public bus service
Table 3.3.4.1	Contents of the GTFS files that the passenger interface draws on
Table 3.5.1	Principal tools and technologies
Table 4.2.1	Stakeholders and their needs
Table 4.3.1	Functional requirements of the passenger interface
Table 4.3.2	Server requirements the passenger interface depends on
Table 4.4.1	Non-functional requirements and design response
Table 4.5.2.1	Principal passenger use cases
Table 4.5.3.1	System operations used by the passenger interface and their endpoints
Table 4.7.1	Data held about passengers and waiting requests
Table 4.8.1	Parts of the passenger page and what each one does
Table 4.9.1	Controls offered for each waiting-request status
Table 5.3.1	Components used by the passenger interface and their responsibilities
Table 5.5.1	Passenger-interface actions and the API calls they trigger
Table 5.7.1	Contribution summary by activity
Table 6.3.1	Manual checks of the passenger flows on the deployed system
Table 6.3.2	Automated server tests that support the passenger flows
Table 6.6.1	Limitations of the passenger interface
Table 6.7.1	Evaluation of the objectives
Table A.1	Project and group information
Table A.2	Nature of the individual contribution by activity
Table D.1	Manual passenger test script
Table F.1	API endpoints used by the passenger 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. This report does not attempt to measure that situation. It takes the proposal's premise as the motivation for building a prototype and uses the literature to place the work in context.

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]. In every such system the passenger meets the information through an interface, and the quality of that interface decides whether the data on the server is usable by the person at the stop. Chapter 2 reviews this literature and its limits.

The project was carried out by a group of three students: Kenneth Kaunda Musonda (2020056283), Madalitso Nyirenda (2021502597) and the author, Ngosa Muleba (2022044371), 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 is an individual report. It describes the shared project and its requirements, and it concentrates on the part the author built: the **passenger interface**, comprising the live bus map, the route search, the choice of a boarding stop and the waiting requests. The author also took part, jointly with the other members, in the problem analysis and the requirements analysis. The server and the driver dashboard were built by the other members and are described only where the passenger interface depends on them.

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 buses that the passenger interface shows on its map are therefore buses reporting real device positions. 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.

From the passenger's side the problem is therefore how to give a person standing at a stop, using only a web browser, a way to (a) see which buses are active and where they are, (b) find the route they need and choose its destination, (c) choose the stop at which they will board, (d) record that they are waiting there, so that the driver can see demand, and (e) see an estimated arrival time that is shown only when the system has enough valid information to calculate one. The group's problem also includes the server side (accepting positions only from an authorised driver, keeping the state of each bus and calculating the estimate) and the driver side. This report concentrates on the passenger-facing part. 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**.

The aim of the author's individual work is to design and implement the passenger interface through which those three features reach the passenger, and to document and evaluate it honestly within the limits of the available evidence.

1.4 Project Objectives

The proposal listed five objectives. They were reviewed against what was built. Table 1.4.1 shows the two that concern the passenger interface most directly, how they were treated in the final project, and the objectives that depend on the passenger interface indirectly.

Table 1.4.1: Proposal objectives and the passenger-side objectives of the final project

Proposal objective	Objective as implemented	Link to the passenger interface
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)	The passenger map displays the buses that the server holds, with registration, status and speed
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	This is the objective the author implemented

Proposal objective	Objective as implemented	Link to the passenger interface
	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	The passenger page presents the estimate with the waiting request
4. Enable operators to monitor and manage the fleet	Records managed through the Django administration site; no separate operator dashboard is claimed	Not a passenger-interface objective
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	Waiting requests created on the passenger page supply those counts

For the author's individual work, the objectives are:

- to show the active buses on a live map, with their registration number, status and speed, and to refresh them automatically;
- to let a passenger register, sign in and sign out, and reach the password-reset page;
- to let a passenger search a route by name, choose one of its destinations and see the ordered stops of the trip;
- to let a passenger choose a boarding stop and create a waiting request, within the rule that only one request may be active at a time;
- to let a passenger see their own requests and mark a request as boarded, cancel it or complete it;
- to show the arrival estimate for a waiting request when one is available; and
- to verify these flows on the deployed system, together with the driver dashboard and the server.

The author also contributed, jointly with the other members, to the problem analysis and the requirements analysis from which these objectives were derived.

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, and the author's part is identified.

Table 1.5.1: Scope of the project as implemented

Scope item	Scope	What was delivered
Real-time GPS tracking of	In scope	Implemented. The driver dashboard sends the device's coordinates

Scope item	Scope	What was delivered
buses		and speed and the server accepts them only from an authorised driver. The passenger map shows the result. The proposal's simulated GPS data was replaced after supervisor feedback.
Display of bus routes and stops on a map	In scope	Author's part. 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	Author's part. 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 by another member. The driver dashboard offers sign-in, bus selection, activation, trip start, GPS tracking and the number of passengers waiting at each stop.
Admin dashboard for transport operators	In scope	Implemented through the Django administration site. 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 passenger interface is accessed through a web browser.

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 server that keeps the state of each bus, a road-routing service and a browser interface for the passenger. The passenger interface is the point at which that pipeline becomes useful to a person. It gives passengers a route-aware way to choose a bus, to register at a boarding stop and to obtain an arrival estimate, using only a web browser and no installation. For the university it provides a documented, tested base that later groups can extend. For the author it demonstrates the design of a map-based passenger interface for a public-transport service and the application of object-oriented analysis to a real system, using notation from Larman [4].

The significance should not be overstated. The passenger interface has not been evaluated by passengers, it has not run as a city-wide service, and the estimates it displays 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, and test coverage. 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 passenger interface was therefore verified against its design, not validated against an operating bus service.
- **Time.** The project had to be completed within a short academic year, while the author also carried the workload of other academic activities.
- **Scope and evidence.** No user study, user acceptance test, usability scoring, field trial or performance benchmark was carried out, although the proposal planned user testing and performance targets. The project therefore shows that the passenger interface works under test conditions. It does not show that it is easy to use or that it improves passengers' waiting experience.
- **Data.** No observed arrival times were collected, so the accuracy of the estimate shown to passengers could not be measured. Journey planning uses 136 representative trips (68 routes, two directions each), not a full timetable.
- **Technology.** Arrival times use OSRM road durations and do not model live traffic or dwell time at stops. The passenger map shows buses as markers; routes and stops are presented as lists, not drawn on the map. Bus data reaches the page by polling every 15 seconds, not by a push connection, and the map library and map tiles are loaded from external services.
- **Test coverage.** The 35 automated tests belong to the server. There are no automated tests of the passenger interface, which was checked manually.

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, with the detailed design of the passenger interface in Section 4.8. Chapter 5 documents the implementation of the passenger interface and the author's 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 and Appendix B is a user guide for the passenger interface.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter reviews the concepts and related work that informed the Bus Tracking System, with attention to what they imply for the passenger interface. It covers Intelligent Transportation Systems, GPS-based vehicle tracking, estimated time of arrival, real-time passenger information, public-transport data standards, maps and road routing, and the software-engineering approach. Existing systems are compared, a research gap is identified and a conceptual framework is derived. The literature review was prepared jointly by the group; this chapter is the author's adaptation of it for the passenger interface.

2.2 Key Concepts and Theories

2.2.1 Intelligent Transportation Systems and GPS-Based Tracking

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 identifies smart infrastructure, data-driven mobility and smart public transportation among the prominent research areas and treats ITS as an integrated technological environment rather than a single technology [1]. 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.

GPS-based tracking determines the position of a vehicle and communicates it to a central system. 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, with client applications that displayed bus positions and arrival information to passengers [2]. 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. For the passenger interface this means that the position on the map is only as current as the last update and the last refresh, which is why the interface refreshes its data at a fixed interval and offers a manual refresh control.

2.2.2 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 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. 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. The passenger interface follows the same principle: it displays an estimate only when the server supplies one.

2.2.3 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 passenger interface should not assume that it reduces actual waiting time in every setting. The delivery of the information, which is the work of the passenger interface, is part of what determines the effect.

2.2.4 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 [7]. 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. These files are what the passenger interface presents when it lists routes, destinations and the ordered stops of a trip. 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.

2.2.5 Maps and Road Routing

Bus tracking is geographical, so a passenger benefits from seeing buses on a map. The passenger interface uses Leaflet, an open-source JavaScript library for interactive maps [13], with map tiles from OpenStreetMap [14]. These two components are used only for display in the browser. The system's own use of routing is different and takes place on the server: it asks the Open Source Routing Machine (OSRM) for the road route between two coordinates, and the OSRM route service returns a distance in metres and a duration in seconds [8]. The server uses these results to decide whether a bus is at a stop and how long a bus will take to reach a stop, and the passenger interface shows the resulting estimate.

2.2.6 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. Larman describes iterative development and the

Unified Process artefacts used to analyse and design object-oriented systems, including the use-case model, system sequence diagrams and the domain model [4]. Chapters 3 and 4 use this approach. Larman's material establishes how to analyse and document a system; it does not itself establish what passengers need, which comes from the transport literature and the interface analysis described in Chapter 3.

2.3 Existing Systems / Related Work

2.3.1 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 [3]. 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.

2.3.2 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 [2]. 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, so it does not address the problem of a passenger choosing among many routes and boarding stops.

2.3.3 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.

2.3.4 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, such as showing vehicles on a map and guiding the user through choosing a destination and a pickup point, not for the transport data model. Because the passenger interface is where those concepts were applied, the Yango analysis is of particular importance to the author's work.

2.3.5 Mobile and Browser Access

GSMA Intelligence describes the growth of mobile connectivity and mobile Internet in Sub-Saharan Africa [10]. This supports delivering transport information through the browser of a mobile device, which needs no application installation. The same source is regional, so it does not support a claim about the Internet access of Lusaka bus passengers, and the project makes none.

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	Limitation for this project
Rea Vaya [9]	Formal BRT infrastructure	Structured service information	BRT scale and infrastructure
OneBusAway [3]	Transit agency real-time data	Next-bus information by web, phone, text and apps	Established agency environment
University bus tracking system [2]	Consumer-grade GPS on buses, mobile Internet	Bus positions and arrival information	Not a multi-route public network
GTFS and GTFS Realtime [7], [6]	Data standards (feeds)	Static schedules; real-time positions, updates and alerts	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	Live map, route search, boarding stops, waiting requests, OSRM-based ETA in a browser	Prototype; not evaluated with passengers or 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, together with a browser interface through which a passenger uses them.

2.5 Research / Knowledge Gap

The literature supports five gaps, expressed cautiously because the reviewed sources cannot establish that no solution exists in Zambia.

- **Geographic.** Much of the cited empirical work originates outside Africa: Watkins et al. and Liu and Miller studied U.S. transit [3], [5], and Khalid et al. developed their

system in Malaysia [2]. The ITS review also notes geographical concentration in the field [1].

- **Contextual.** Reported systems are often tied to a specific organisation, for example a university service [2]. This project considers representative public transport routes in Lusaka, where a passenger must first choose among many routes and directions.
- **Implementation.** Many deployed systems rely on dedicated GPS equipment and agency data feeds. A student prototype can demonstrate the whole information flow, up to the passenger's screen, 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 [3], [5]. 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 [7], [6]. The project uses static GTFS for its data and leaves GTFS Realtime as future work.

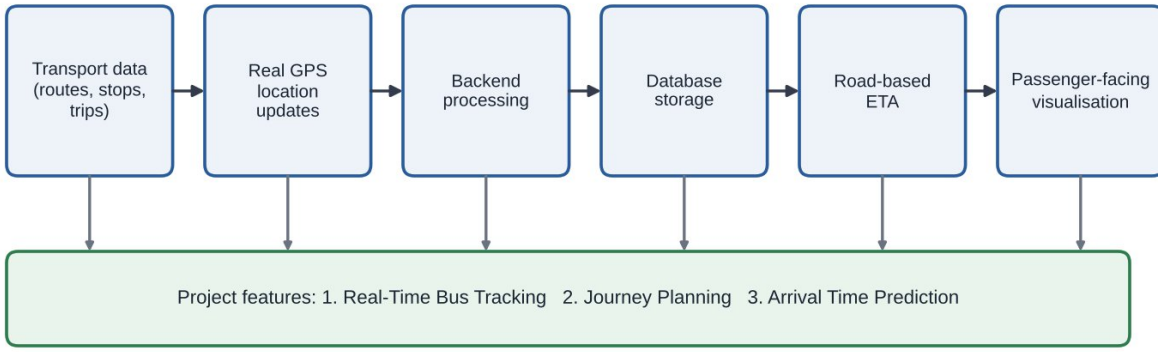
The gap addressed by this project is therefore a localised implementation gap: a working, documented system that connects real driver-device GPS updates, GTFS-derived route and stop data, current bus state, waiting requests and a conditional ETA for Lusaka routes. The passenger interface is the part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

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 interface. 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. The passenger interface is the output stage of all three.

conceptual framework

Conceptual framework derived from the reviewed literature



Conceptual diagram drawn from the literature; not the deployed architecture of this project.

Figure 2.6.1: Conceptual framework derived from the reviewed literature

2.7 Chapter Summary

The literature establishes that GPS can supply vehicle position [2], that real-time passenger information can influence waiting experience [3] but not in every setting [5], and that ITS is an established area of work [1]. GTFS provides a standard model for transport data [7]. The review supports building a localised prototype, treating ETA as a conditional estimate that the interface shows only when it exists, and being cautious about claims on waiting time. The next chapter explains the methodology used.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This chapter describes the methodology used to develop the Bus Tracking System and, within it, the passenger interface. It explains the iterative and incremental approach, the requirements analysis that the group carried out jointly, the analysis of the Yango application and its adaptation to predefined bus routes, the transport data the interface presents, the tools and technologies used, and the way the work was evaluated.

The development process combined findings from the literature review with practical analysis of existing public-transport applications. The Yango application was analysed through the team members' prior use of it. Its interface and interaction patterns informed notes on potentially relevant functions, including location-based information, route and destination selection, vehicle information and passenger interaction. These are the functions that the passenger interface had to provide.

The implemented system developed beyond the original proposal. Early work included simulated vehicle movement, but supervisor feedback led the group to replace simulation with real GPS coordinates obtained from a driver's device. The methodology therefore distinguishes between the original proposal, development prototypes and the functionality implemented in the final system.

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; Larman presents this as a core idea of the Unified Process [4]. The approach suited this project because the system has several connected parts whose requirements became clearer as earlier parts were exercised. The passenger interface in particular depended on server endpoints that were being built at the same time, so its screens were developed and revised as those endpoints became available.

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.

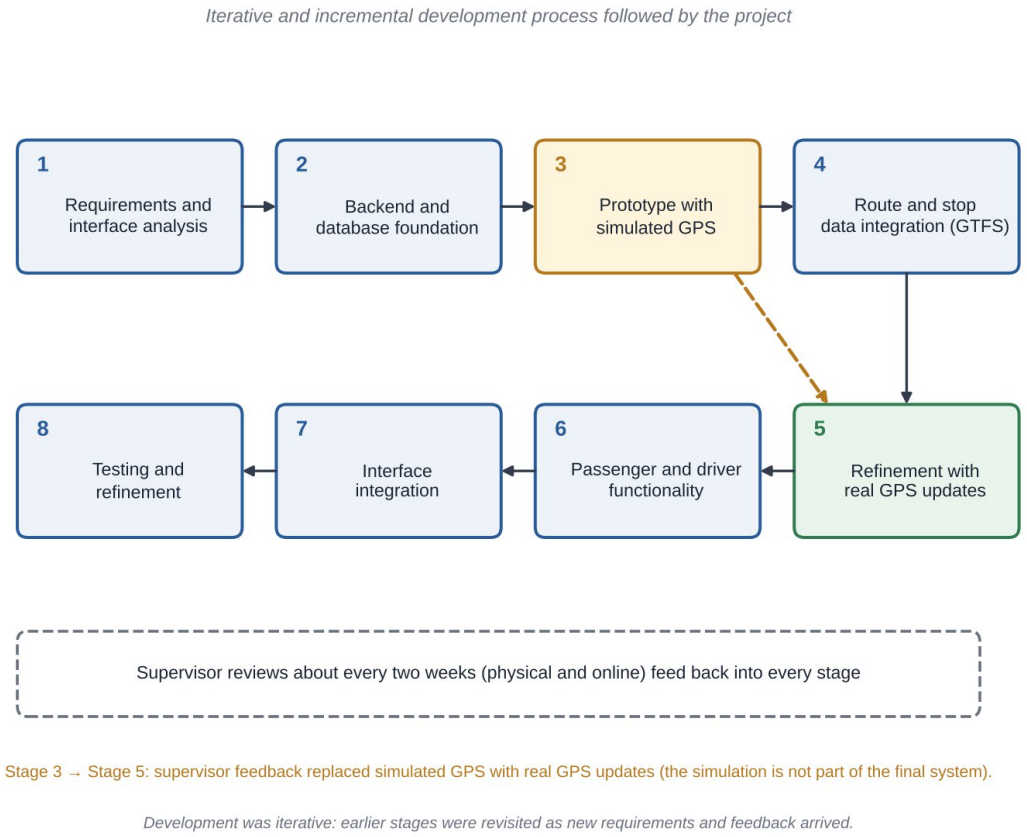


Figure 3.2.1.1: Iterative and incremental development process of the Bus Tracking System

3.2.2 Work in Each Stage

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

Table 3.2.2.1: Stages of development and the passenger-interface work in each stage

No.	Stage	Work in the stage	Passenger-interface work
1	Requirements and interface analysis	Shared requirements analysis, including the study of the interaction concepts of the Yango application	Joint contribution; derived the passenger flow of map, route, stop and request
2	Server and database foundation	Django project, initial models and migrations, REST endpoints for routes, stops and buses	First map page showing buses returned by the bus endpoint
3	Prototype with simulated GPS	Not part of the final system; replaced in Stage 5	Map displayed simulated positions; replaced
4	Route and stop data integration (GTFS)	Models and import commands for the GTFS files	Route search, destination choice and listing of the stops of a trip
5	Refinement with real GPS updates	Authenticated GPS update endpoint, bus state and status logic; a driver interface that sends the device's location	Map shows real driver-device positions, with status and speed in the marker popup

No.	Stage	Work in the stage	Passenger-interface work
6	Passenger and driver functionality	Passenger and driver accounts, assignment checks, waiting requests and waiting counts	Passenger sign-in and registration, boarding-stop selection, waiting-request creation and list, controls for board, cancel and complete
7	Interface integration	Browser interfaces connected to the API endpoints	Request list shows the arrival estimate; polling and refresh control
8	Testing and refinement	Automated server tests, manual checks of the deployed flows, deployment configuration	Manual verification of the passenger flows on the deployed system

The first three columns follow the group report. The right-hand column places the passenger-interface work within each stage.

3.2.3 Proposal Compared with the Implemented Passenger Interface

The final passenger interface differs from the proposal in several respects. Table 3.2.3.1 records the differences so that the report does not describe the proposal as if it had been delivered.

Table 3.2.3.1: Proposal compared with the implemented passenger interface

Aspect	Proposal	Implemented passenger interface
Bus position source	Simulated GPS generated along routes	Real GPS latitude, longitude and speed sent by an authorised driver's device and displayed on the map
Passenger interface technology	React with Google Maps API	Browser page using HTML, CSS and JavaScript with Leaflet and OpenStreetMap, delivered as a Django template, with no front-end framework
Real-time delivery	Firebase for real-time data	Standard HTTP requests; the page polls the API every 15 seconds; no WebSockets
Arrival estimate	ETA within ±2 minutes of expected	ETA from OSRM road duration shown with the waiting request; accuracy not evaluated
Route display	Routes shown on a map	Buses shown as markers on the map; routes, destinations and stops shown as lists
Evaluation	User acceptance testing with five or more users, SUS of at least 70, load times	Manual functional checks of the deployed passenger 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 author took part in this analysis together with the other group members. 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 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 shown in Table 3.3.3.1. The rightmost column names the part of the passenger interface that realises each adaptation.

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

Ride-hailing concept observed	Adaptation for public buses	Part of the passenger interface
Vehicle locations on a map	Active buses shown on a map	Live map with bus markers and popups
Choosing a destination	Choosing a route and then one of its destinations	Search route field, route list and destination buttons
Choosing a pickup point	Choosing a boarding stop from the ordered stops of a trip	Ordered stop list; confirmation message for the selected stop
Requesting a ride	Creating a waiting request at a	Create Waiting Request control

Ride-hailing concept observed	Adaptation for public buses	Part of the passenger interface
	boarding stop	
Vehicle information and arrival time	Bus status and an arrival estimate for the waiting stop	Marker popup (status, speed) and the estimate in the request list
Driver side of the interaction	Driver activates a bus, starts a trip and sends GPS updates; sees waiting counts	Not part of the passenger interface; the request it creates is what the driver sees

The Yango 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 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 [11], gives the feed version as lusaka_city_pt, and lists three agency categories: minibuses of 14 to 18 seats, big buses and shared taxis. Table 3.3.4.1 gives the counts that matter to the passenger interface, computed directly from the files in the repository.

Table 3.3.4.1: Contents of the GTFS files that the passenger interface draws on

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
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. For the passenger this means that choosing a route and a destination identifies exactly one trip, which makes the choice unambiguous. The consequence is that the passenger interface models route and direction, not a time-of-day schedule.

3.4 System Architecture / Research Workflow

The system follows a client–server architecture. The browser interfaces send HTTP requests with JSON to the Django server, 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. 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.

3.5 Tools and Technologies

Table 3.5.1 lists the technologies that the passenger interface uses directly and the server technologies it depends on. The server versions are those pinned in requirements.txt of the final 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
HTML, CSS and JavaScript		Passenger interface, written as a Django template with inline styles and scripts and no front-end framework
Browser Fetch API		The page's scripts call the JSON API and send the CSRF token
Leaflet [13]	1.9.4	Interactive map library for the passenger map, loaded from the unpkg content-delivery network
OpenStreetMap tiles [14]		Map background tiles requested by Leaflet in the browser
Python and Django [12]	6.0.6	Web framework that serves the pages and hosts the API
Django REST Framework	3.17.1	Serialisers and API views called by the page
PostgreSQL		Database of the deployed system
OSRM [8]		Road distance and duration used for the arrival estimate
Visual Studio Code		Editor used in development
Git and GitHub		Source control; the integration branch is the final state of the system
Render		Hosting of the deployed application, on which the passenger flows were checked

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. A passenger's waiting requests record the route, the boarding stop and times. The following considerations apply to the passenger interface.

- **Credentials.** The sign-in and registration forms of the passenger page send credentials to the server, which stores passwords as hashes. The page does not store a password itself.
- **Access control.** A passenger's waiting-request endpoints are scoped to that passenger, so the request list shows only the signed-in passenger's own requests.
- **Location data.** The location shown on the passenger map is that of a bus, supplied by its driver. In the checks reported in Chapter 6 the location came from a test device and not from 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 and services.** Leaflet, OpenStreetMap tiles and the other libraries are used under their own licences and terms, and the sources in Chapter 2 are cited.

3.7 Evaluation Method

Evaluation was limited to functional verification. The passenger interface has no automated tests. It was checked by hand on the deployed system, together with the driver dashboard and the server, using a test passenger account, a test driver account assigned to a test bus, and the location reported by the device used for the checks (Section 6.3). The server's 35 automated tests, written with Django's test framework, support these checks because they verify the arrival-time rules and the passenger-scoped endpoints on which the passenger page relies. 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 manual checks supported by the server's automated tests. The supervisor's feedback replaced simulated GPS with real GPS. The next chapter sets out the analysis and design of the system, with the detailed design of the passenger interface.

CHAPTER 4: SYSTEM ANALYSIS AND DESIGN

4.1 Introduction

This chapter presents the analysis and design of the Bus Tracking System as seen from the passenger. It follows the Unified Process artefacts described by Larman [4]: a use-case diagram, a system sequence diagram, an architecture diagram and a domain model. The analysis artefacts of the group (stakeholders, requirements, use cases, architecture and domain model) were prepared jointly and are reproduced here as far as they concern the passenger. The detailed design of the passenger interface (Section 4.8) and its supporting logic (Section 4.9) is the author's own work. The server-side design, including the entity-relationship diagram, the design model and the driver flow, was prepared by another member and appears only in summary; the entity-relationship diagram is in Appendix C.

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	Passenger interface: live map, route search, stop selection, waiting requests and the arrival estimate
Driver	Sign in, operate only the assigned bus, start a trip, report position and see waiting passengers	Driver dashboard; waiting counts that come from passengers' requests
Administrator	Maintain buses, drivers, assignments and transport records	Django administration site
OSRM routing service (external)	Receive route requests and return road distance and duration	Called by the server, never by the passenger page
Server developer (group member)	A passenger page that uses the endpoints as designed	The passenger interface calls the endpoints in Table 4.5.3.1
Supervisor and examiners	A documented, verifiable implementation	This report and the final branch

4.3 Functional Requirements

The functional requirements are grouped under the three features of the system: Real-Time Bus Tracking, Journey Planning and Arrival Time Prediction. Table 4.3.1 lists the

requirements placed on the passenger interface. Each is linked to the group-level requirement (FR number) that it realises. Table 4.3.2 lists the server requirements on which they depend.

Table 4.3.1: Functional requirements of the passenger interface

ID	Feature	Requirement	Group FR
PR1	Real-Time Bus Tracking	Show the active buses as markers on a map, with a popup giving the registration number, status and speed of the bus	FR6
PR2	Real-Time Bus Tracking	Refresh the bus data automatically every 15 seconds and on the passenger's request	FR2, FR6
PR3	Journey Planning	Let a passenger register, sign in and sign out, and reach the password-reset page	FR12, FR19
PR4	Journey Planning	Let a passenger search routes by typing part of a name and select one of the matches	FR10
PR5	Journey Planning	List the destinations of the selected route and let the passenger choose one	FR11
PR6	Journey Planning	Show the ordered stops of the trip for the chosen destination and let the passenger choose a boarding stop, confirming the choice	FR11
PR7	Journey Planning	Create a waiting request for the chosen trip and stop, confirm that it was created, and respect the rule that a passenger has one active request at a time	FR13
PR8	Journey Planning	List the signed-in passenger's own requests with route, boarding stop, time and status	FR13
PR9	Journey Planning	Offer controls to mark a request as boarded, cancel it or complete it, according to its status	FR14
PR10	Arrival Time Prediction	Show the estimated arrival time for a request in the WAITING status when the server supplies one	FR16
PR11	All	Run in a standard web browser with no installation, as one page with overlay panels over the map	—

Table 4.3.2: Server requirements the passenger interface depends on

Group FR	Server requirement	Used by
FR2, FR6	Store and provide bus records (registration number, status, position, speed, last update)	PR1, PR2
FR10	Search routes by name	PR4
FR11	List the destinations of a route and the ordered stops of the trip for a route and destination	PR5, PR6
FR12	Register, sign in and sign out passengers	PR3
FR13	Create a waiting request for a trip and a stop served by that trip; allow one active request per passenger	PR7, PR8

Group FR	Server requirement	Used by
FR14	Support the waiting-request lifecycle: board, cancel, complete and expire	PR9
FR16	Return an ETA in minutes for a passenger's own WAITING request, or null when no valid estimate exists	PR10
FR19	Support password reset by e-mail	PR3

4.4 Non-Functional Requirements

No numerical performance or scale target is claimed, because none was evaluated. The non-functional requirements below are design requirements and the evidence for each is stated.

Table 4.4.1: Non-functional requirements and design response

Quality	Requirement	Design response and evidence
Accessibility of access	The passenger must be able to use the system without installing anything	Browser page delivered by the server; no native application (Section 1.5)
Honesty of information	The interface must not present an invented arrival time	The estimate is shown only for a WAITING request and only when the server returns a value; the server returns null otherwise (Section 4.9.4)
Currency of information	The bus positions shown should be recent	Polling every 15 seconds and a manual refresh control; the age of the data also depends on the driver's device and the network
Clarity of interaction	The route-to-request flow should be a short, ordered sequence	Four steps (search, destination, stop, request) in one panel, with a confirmation message after the stop and the request (Section 4.8.3)
Security	A passenger must see only their own requests; credentials must not be handled by the page beyond sending them	Session cookie and CSRF token on every call; the server filters requests by the signed-in passenger
Simplicity of transport	Work over standard web infrastructure	Plain HTTP requests and JSON; no WebSockets
Performance	No target claimed	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; the passenger use cases are the ones the author's interface realises.

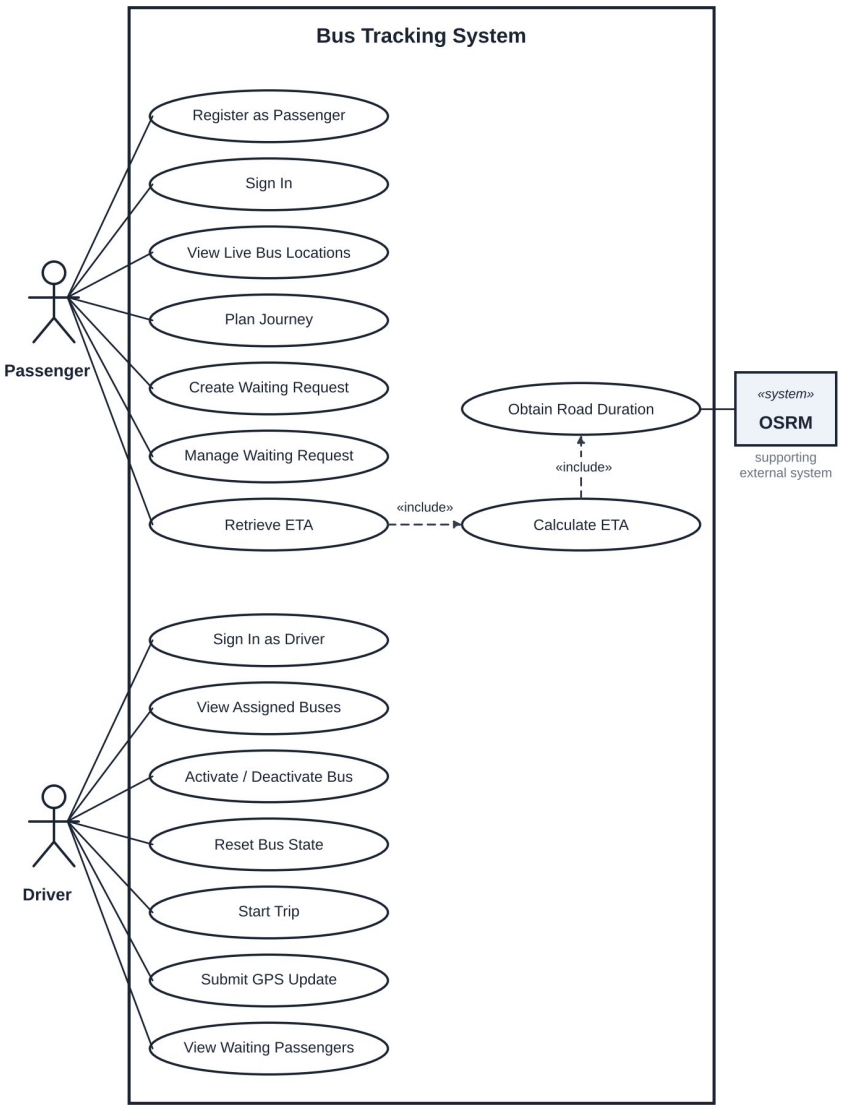


Figure 4.5.1.1: UML use-case diagram of the Bus Tracking System

4.5.2 Use-Case Summaries

Table 4.5.2.1 gives brief descriptions of the passenger use cases in the casual format described by Larman [4], with the part of the interface that supports each. The driver use cases (Start Trip, Submit GPS Update and View Waiting Passengers) belong to the driver dashboard and the server.

Table 4.5.2.1: Principal passenger use cases

Use case	Actor	Description	Interface part
View Active Buses	Passenger	The passenger opens the passenger page and sees the active buses as markers; a marker's popup gives its	Live map

Use case	Actor	Description	Interface part
		registration number, status and speed	
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	Plan Trip panel
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	Plan Trip panel
Manage Waiting Request	Passenger	The passenger marks the request as boarded, cancels it, or completes the trip	My Requests sheet
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	My Requests sheet
Register and Sign In	Passenger	The passenger creates an account or signs in so that requests can be made and owned	Account area of the Plan Trip panel

4.5.3 System Sequence Diagram

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 [4]. Figure 4.5.3.1 covers the scenario in which a passenger plans a journey, creates a waiting request and retrieves the ETA. OSRM appears in it because it provides the road duration that determines the response. Operation names are given at the level of intent and are derived from the endpoints of the implementation (Table 4.5.3.1).

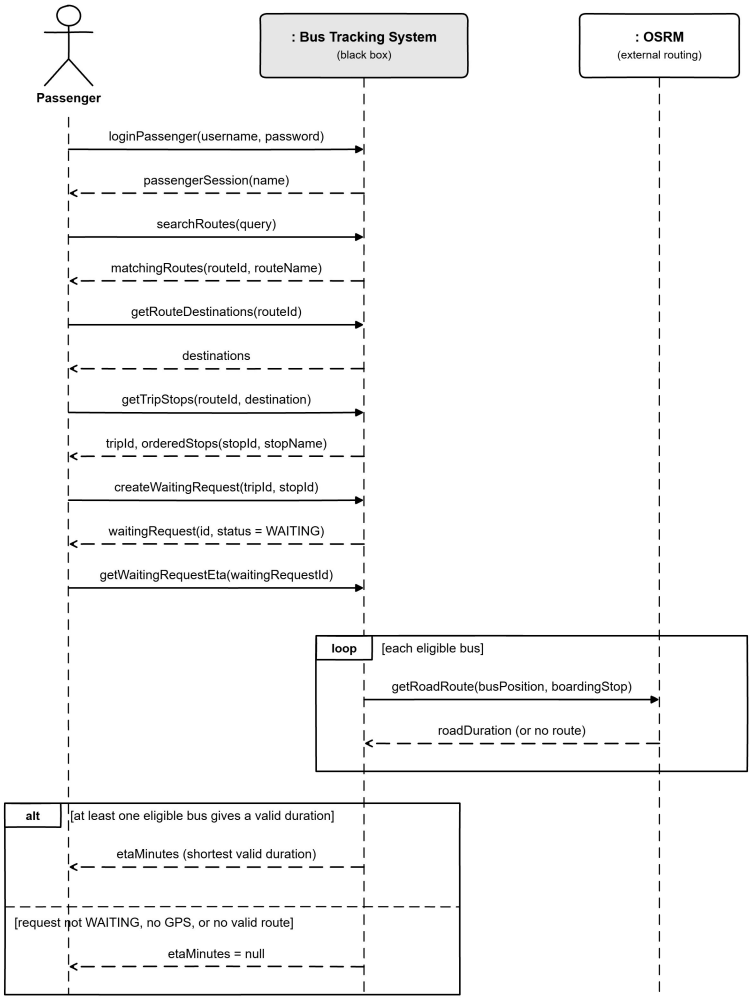


Figure 4.5.3.1: System sequence diagram: passenger plans a journey, creates a waiting request and retrieves the ETA

Table 4.5.3.1: System operations used by the passenger interface and their endpoints

System operation	Method	Endpoint (under /api/)
registerPassenger	POST	auth/register/
loginPassenger	POST	auth/login/
logoutPassenger	POST	auth/logout/
listBuses	GET	buses/
searchRoutes	GET	routes/search/?q=
getRouteDestinations	GET	routes/<route_id>/destinations/
getTripStops	GET	routes/<route_id>/stops/?destination=
createWaitingRequest	POST	waiting-requests/
listWaitingRequests	GET	waiting-requests/

System operation	Method	Endpoint (under /api/)
boardWaitingRequest	POST	waiting-requests/<id>/board/
cancelWaitingRequest	POST	waiting-requests/<id>/cancel/
completeWaitingRequest	POST	waiting-requests/<id>/complete/
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 communicate with the Django project over HTTP using JSON and a session cookie. The Django project handles URL routing, settings, session authentication and static files, and is served by gunicorn. 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. OSRM is an external service reached through HTTP. Password-reset e-mail is sent through Brevo.

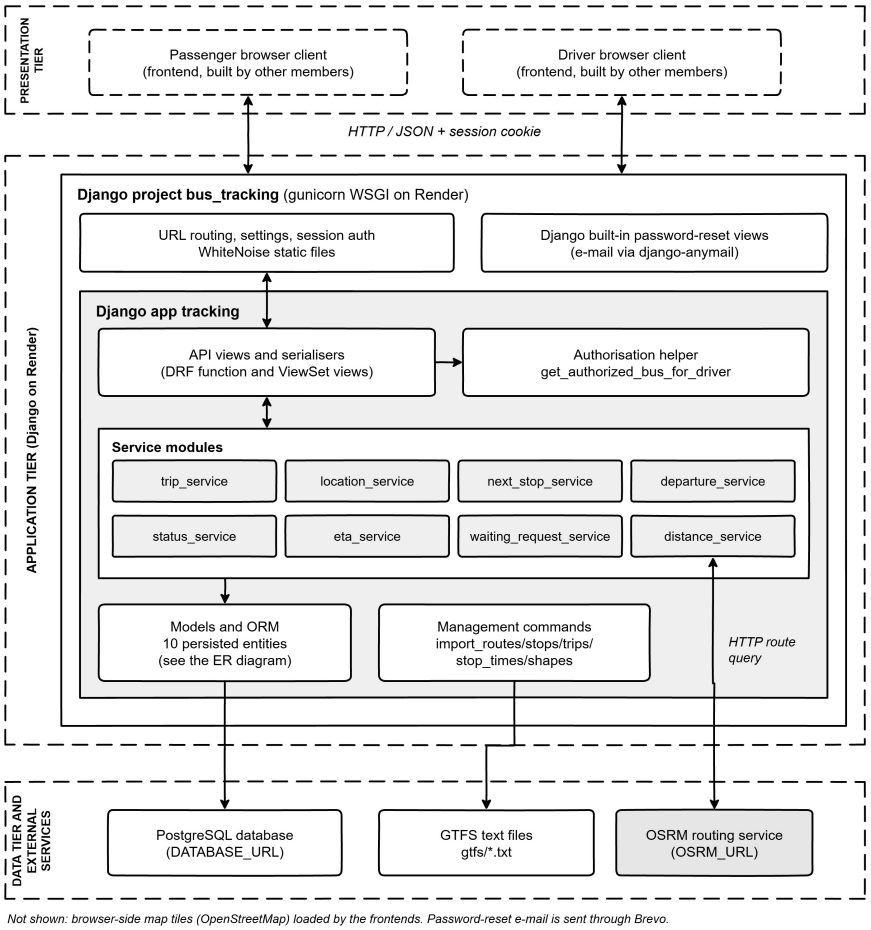


Figure 4.6.1: System architecture of the Bus Tracking System

The system is a client–server application with a three-tier structure. The passenger interface belongs to the **presentation tier**: it runs in the passenger's browser and is delivered as a page by the server. It never reads the database or calls OSRM directly. Every piece of information on the passenger page, from the bus markers to the arrival estimate, arrives as JSON from the API in the application tier. The map tiles are the one exception: Leaflet requests them directly from OpenStreetMap in the browser, and the Leaflet library itself is loaded from a content-delivery network.

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 [4]. 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*. 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.

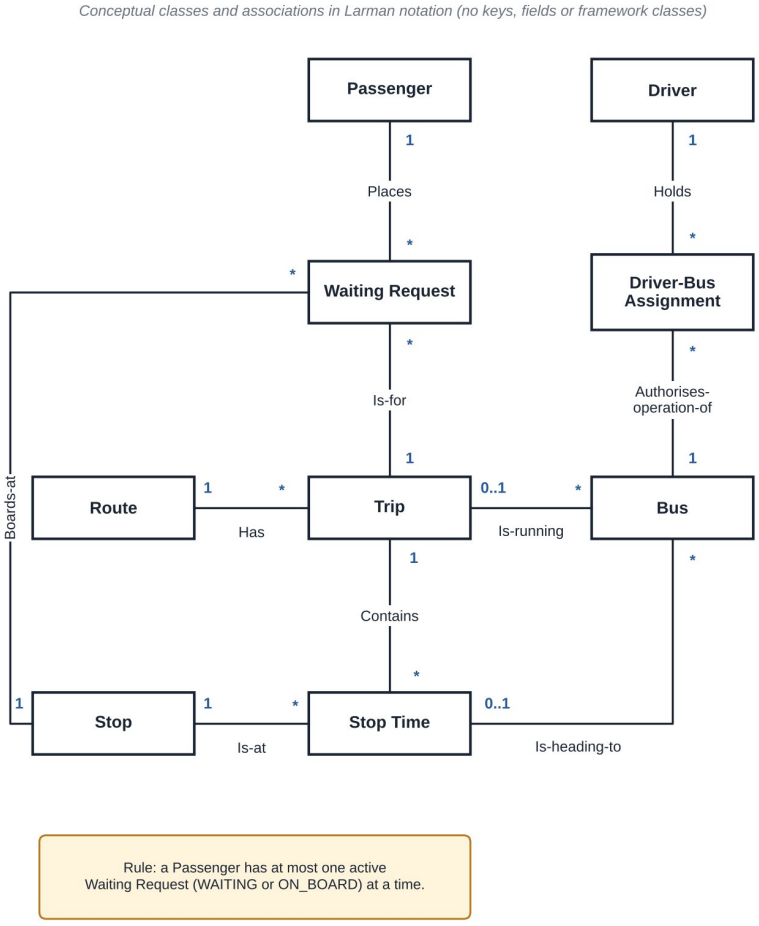


Figure 4.7.1.1: Domain model of the Bus Tracking System (conceptual)

4.7.2 Data Presented by the Passenger Interface

The passenger interface reads and writes a small part of the persisted data. Table 4.7.1 lists the entities it touches and the fields that appear on the passenger's screen. The full entity-relationship design, prepared by another member, is in Appendix C.

Table 4.7.1: Data held about passengers and waiting requests

Entity	Fields the interface uses	Rules
Bus	registration_number, status, current_lat, current_lng, speed, is_active	Read-only for the passenger; status is IDLE, IN_TRANSIT or AT_STOP; only the latest state is stored
Route	route_long_name	Searched by name; one route has many trips
Trip	trip_headsign, direction_id, route	The representative trip (identifier ending -08:00:00) identifies a route and direction
StopTime and Stop	stop_sequence, stop_name	Give the ordered stops of a trip that the passenger chooses among
Passenger	user (one-to-one), name	Created at registration; 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 client cannot set its own status or timestamps

The application stores only the current state of each bus. It has no history table for locations, so the passenger map can show only where a bus is now, not where it has been.

4.8 Interface / Interaction Design

This section is the author's principal design contribution. It describes the passenger interface from the repository code and the screenshots of the deployed system.

4.8.1 Overall Layout and Navigation

A landing page (Figure 4.8.1) lets the user choose the passenger or the driver role. The passenger page is organised around one idea: the map is always the base screen, and everything else opens over it. A blue header bar carries the title *Lusaka Bus Tracker* and four controls: **Active Buses**, **Plan Trip**, **My Requests** and a refresh control. A second, round refresh control floats at the lower right of the map. Table 4.8.1 lists the parts of the page.

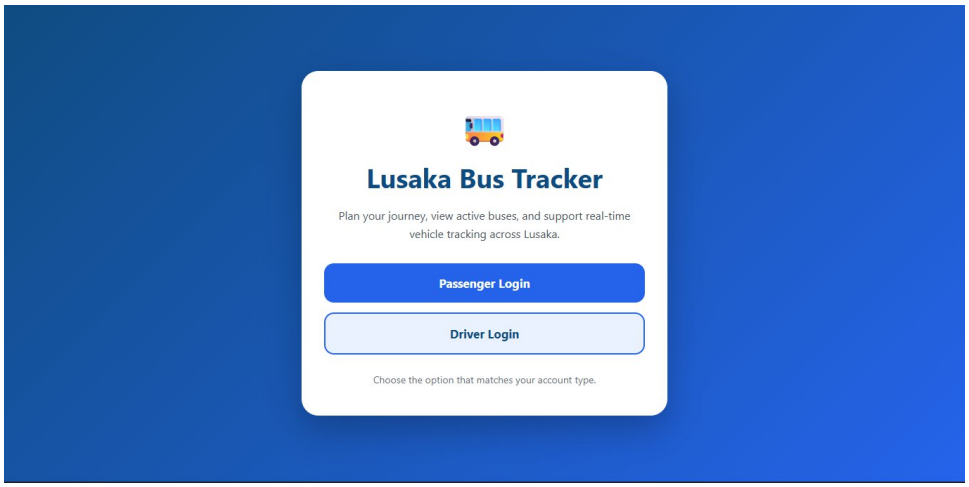


Figure 4.8.1: Landing page of the deployed system

Table 4.8.1: Parts of the passenger page and what each one does

Part	What the passenger does or sees	Requirements
Header bar	Title and the controls Active Buses, Plan Trip, My Requests and refresh	PR11
Live map	Leaflet map on OpenStreetMap tiles with zoom controls; a marker for each active bus; a popup with registration number, status and speed	PR1, PR2
Plan Trip panel	An overlay panel with the account area (register, login, forgot password, logout), the route search, the destination choice, the list of stops, and the Create Waiting Request and Cancel controls	PR3 to PR7
My Requests sheet	A sheet that rises from the bottom of the screen with the heading *My Waiting Requests* and one card per request	PR8 to PR10
Messages	Coloured banners at the top of the screen that confirm an action, for example that a boarding stop was selected or that a waiting request was created	PR6, PR7

4.8.2 The Live Map

Figure 4.8.2 shows the passenger map. An active bus appears as a green bus marker. The marker's popup gives the registration number, the status and the speed, which turns a bare position into information about the bus. The data is requested from the API when the page loads and again every 15 seconds. The refresh controls allow the passenger to request it at once. The map shows the buses only; routes and stops are presented as lists in the Plan Trip panel, not drawn on the map.

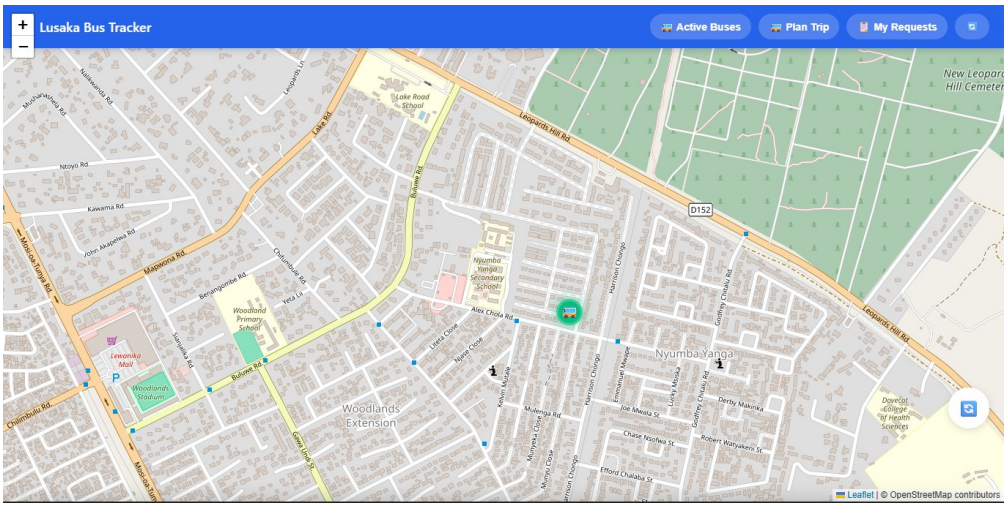


Figure 4.8.2: Passenger map with the header controls and an active bus marker

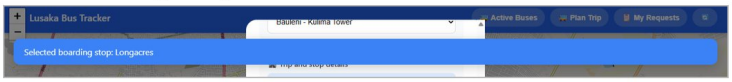
4.8.3 Trip Planning and Boarding Stop Selection

The Plan Trip panel guides the passenger through four steps, which correspond to the Journey Planning feature and to the ride-hailing concepts adapted in Table 3.3.3.1. First the passenger registers or signs in. Second, the passenger types part of a route name in the *Search route* field, and the matching routes appear in a selection list. Third, the panel displays the chosen route under *Trip and stop details* with the prompt *Select a destination*, and the passenger touches one of the destination buttons. Fourth, the panel lists the ordered stops of the trip, and the passenger touches the stop where they will board; a banner confirms the choice (*Selected boarding stop: …*). The *Create Waiting Request* button then submits the request. Figure 4.8.3 shows the route search, the boarding-stop confirmation and the submission control.

(a) Route search and selection



(b) Boarding-stop selection confirmed



(c) Waiting-request submission control

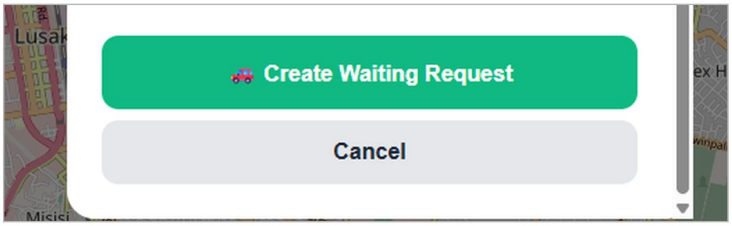


Figure 4.8.3: Passenger interface screens for route search, boarding-stop selection and waiting-request creation

Two decisions shaped this design. First, a passenger chooses a *destination* and not a *direction*, because the destination is the form in which a passenger thinks of a journey; the server turns the route and destination into one representative trip (Section 3.3.4). Second, the stops are offered as a list in travel order, so that the passenger can see which stops lie ahead of the one they choose.

4.8.4 Waiting Requests

The *My Waiting Requests* sheet lists the passenger's requests, newest at the bottom in the checks reported here. Each card gives the route name in blue, the status at the right in a colour that distinguishes it, and a second line with the boarding stop and the time of the request. The card of the request in progress is outlined. For a request in the WAITING state the card adds the line *Estimated arrival at …: about N minutes* and two controls, a green **Mark as Boarded** button and a grey **Cancel Trip** button (Figure 4.8.4). After boarding, the request is in the ON_BOARD state and the card offers the control that completes the trip (Figure 4.8.5). Requests that were cancelled stay in the list with their status and no controls.

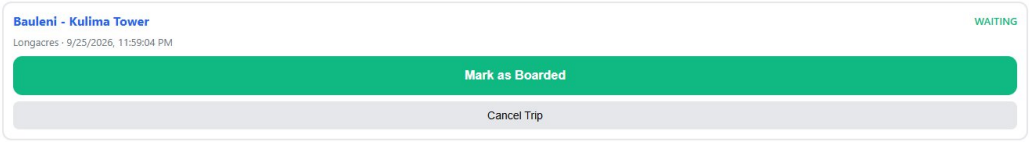


Figure 4.8.4: A waiting request in the WAITING state

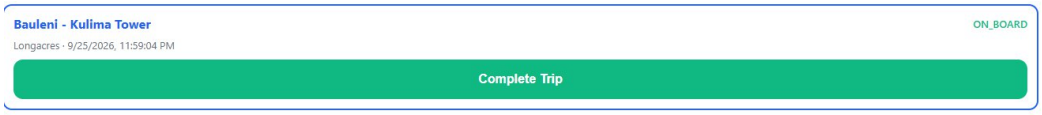


Figure 4.8.5: A waiting request in the ON_BOARD state

4.9 Algorithms / Models / Technical Design

The passenger interface contains little computation of its own: the server holds the rules. Its logic is the sequencing of calls and the choice of what to show. This section states that logic.

4.9.1 Map Refresh

- When the page loads, request the bus records from the API and draw a marker for each active bus that has a position.
- Attach a popup to each marker giving the registration number, status and speed.
- Repeat the request every 15 seconds and update the markers with the new data.
- Repeat the request at once when the passenger presses a refresh control.

The interval is a compromise between currency and load. It is not tuned, and the position the passenger sees can be older than 15 seconds because the driver's device also sends updates at its own rate.

4.9.2 Journey-Planning Sequence

- When the passenger types a route name, request the routes that match and offer them in the selection list.
- When the passenger selects a route, request its destinations and present them as buttons.
- When the passenger selects a destination, request the ordered stops of the trip for that route and destination, and present them as a list.
- When the passenger selects a stop, confirm the choice with a banner and keep the choice for the request.
- When the passenger presses Create Waiting Request, send the chosen trip and stop. On success, confirm with a banner and add the request to the list.

4.9.3 Waiting-Request Lifecycle and the Controls Shown

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, an operation limited to

administrators. A passenger cannot create a request while another is WAITING or ON_BOARD. The server enforces these rules, so a client cannot set a status directly. The interface offers only the controls that suit the current status, as Table 4.9.1 shows.

Table 4.9.1: Controls offered for each waiting-request status

Status	Meaning	Controls and information on the card
WAITING	The passenger is waiting at the boarding stop	Estimated arrival (when available); Mark as Boarded; Cancel Trip
ON_BOARD	The passenger has boarded	Control to complete the trip
COMPLETED	The journey is finished	None; kept in the list as a record
CANCELLED	The passenger cancelled	None; kept in the list as a record (Figure 6.3.4)
EXPIRED	An administrator ended a waiting request	None; kept in the list as a record

4.9.4 The Arrival Estimate as Presented

The estimate is calculated on the server. It is requested for a request in the WAITING state. The server selects the active buses on the same trip with valid coordinates that have not passed the stop, asks OSRM for the road duration of each to the stop, and returns the shortest valid duration in minutes, or null if no bus qualifies or routing fails [8]. The passenger interface shows the figure as *about N minutes* when the server returns one and has nothing to show when it returns null; it never substitutes a value of its own. This follows the principle of Section 2.2.2 that a wrong arrival time is worse than none. The duration is not adjusted for live traffic or for dwell time at stops.

4.10 Chapter Summary

The analysis defines three features and, for the passenger interface, eleven requirements linked to the group's nineteen. The design shows the passenger interface as the presentation tier of a three-tier system, organised around a live map with overlay panels for trip planning and for waiting requests. Its logic is a short sequence of calls to the API, a fixed refresh interval and a rule that the controls on a request card follow its status. Chapter 5 documents the implementation.

CHAPTER 5: IMPLEMENTATION / DEVELOPMENT

5.1 Introduction

This chapter describes how the passenger interface was implemented and how it fits into the rest of the system. 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; it was implemented by Kenneth Kaunda Musonda. The second is the driver dashboard, a browser page for bus operations and GPS tracking, implemented by Madalitso Nyirenda. The third is the passenger interface, a browser page with a map of active buses and a trip-planning panel, implemented by the author. The sections that follow document the passenger interface in detail and the other parts only at the points where it meets them.

5.2 Development Environment

The passenger interface was developed on Windows in Visual Studio Code, using HTML, CSS and JavaScript, and tested in a web browser against the Django development server. Git and GitHub were used for source control, and the integration branch is the final state of the system. The deployed system, on which the passenger flows were checked, runs on Render with a PostgreSQL database. The passenger page is a Django template served by the project; it uses Leaflet 1.9.4 for the map, loaded from the unpkg content-delivery network, and OpenStreetMap tiles for the map background.

5.3 System Components / Modules

Table 5.3.1: Components used by the passenger interface and their responsibilities

Component	Responsibility	Developer
Passenger page template	The page structure, styles and scripts: header bar, map container, Plan Trip panel, My Requests sheet and message banners	Author
Map module (Leaflet)	Creates the map, draws and updates the bus markers and popups, and handles zoom	Author
Refresh logic	Requests bus data on load, every 15 seconds and when a refresh control is pressed	Author
Account area	Register, login, forgot-password link and logout controls in the Plan Trip panel	Author
Route and stop selection	Route search, destination buttons and the list of stops of the trip	Author
Waiting-request controls	Creation of a request, the request list, and the board, cancel and complete controls with their status-based display	Author
ETA display	Shows the estimate with a waiting request when the server returns one	Author

Component	Responsibility	Developer
Landing page	Lets the user choose the passenger or driver role	Group (shared page)
JSON API under /api/	Authentication, routes, destinations, trip stops, bus records, waiting requests and ETA	Kenneth Kaunda Musonda
Service modules and models	Bus state, waiting-request lifecycle, OSRM distance and ETA	Kenneth Kaunda Musonda
Driver dashboard	Bus activation, trip start, GPS reporting and waiting counts	Madalitso Nyirenda

5.4 Key Implementation Details

5.4.1 Delivery and Structure of the Page

The Django project serves the passenger page as an HTML template. Its styles and scripts are written inline and no front-end framework is used. The page has one full-screen map beneath a fixed header, and the panels that the header controls open are layered over the map. The scripts call the JSON API with the browser's Fetch API and send the CSRF token with each request that changes data. The passenger's session is held in the session cookie that the server sets at sign-in. Because the page needs no build step and no installation, a passenger needs only a browser and a network connection.

5.4.2 The Live Bus Map

The map is created with Leaflet and OpenStreetMap tiles. For each active bus that has a position, the page draws a bus marker. The marker's popup gives the registration number, the status (IDLE, IN_TRANSIT or AT_STOP) and the speed, as supplied by the server. The page requests the bus data when it loads and repeats the request every 15 seconds; the header refresh control and the floating refresh control request it immediately. This is polling over ordinary HTTP. The proposal had planned Firebase for real-time delivery, but the final system uses no WebSockets (Table 3.2.4.1), and polling was sufficient for a prototype. The speed value is displayed as the server supplies it; the unit is not specified in the server code, so the report assigns none.

5.4.3 Account Area

At the top of the Plan Trip panel is the account area, with fields for the account details and the controls *Register*, *Login*, *Forgot password?* and *Logout*. Registering and signing in call the passenger authentication endpoints. The server applies the configured password validators at registration and limits sign-in to five attempts a minute for each client address, so the page does not implement those rules itself. The *Forgot password?* link leads to the password-reset page that the server provides. A waiting request needs a signed-in passenger, so the account area comes first in the panel.

5.4.4 Route Search and Destination Choice

Typing in the *Search route* field requests the routes whose names match the text, and the matches appear in a selection list (Figure 4.8.3 and Figure 6.3.2). In the test run, the text *bauleni* returned the route *Bauleni – Kulima Tower*. Selecting a route requests its destinations. Under the heading *Trip and stop details* the page then shows the route name and the prompt *Select a destination*, with one button for each destination; for this route the buttons were *Bauleni Junction* and *Kamwala Market – Independence Ave*.

5.4.5 Boarding Stop Selection

When the passenger selects a destination, the page requests the ordered stops of the trip and lists them, one button per stop, in the order the bus visits them (Figure 6.3.3). The passenger touches a stop, which is highlighted, and a blue banner confirms the choice, for example *Selected boarding stop: Civic Center*. The list is long on routes with many stops, up to 38 in the data (Table 3.3.4.1), so it scrolls inside the panel.

5.4.6 Creating and Managing Waiting Requests

The green *Create Waiting Request* button sends the chosen trip and stop to the server, which creates the request in the WAITING state for the signed-in passenger, and a green banner confirms it (Figure 6.3.4). The one-active-request rule and the checks that the stop is served by the trip are enforced by the server. The *My Requests* control opens the sheet that lists the passenger's requests. Each card shows the route name, the boarding stop, the time and the status. For a WAITING request the card shows the board and cancel controls; after boarding it shows the control that completes the trip (Table 4.9.1). Each control calls the matching server operation and the list is updated with the new status. Because the server filters the list by the signed-in passenger, the page shows no one else's requests.

5.4.7 Displaying the Arrival Estimate

For a request in the WAITING state the page asks the server for the estimate and shows it on the card as *Estimated arrival at …: about N minutes*. In the test run the estimate was about 10 minutes (Figure 6.3.4). This is the road-route duration of the nearest eligible bus as calculated by OSRM; it was not compared with an observed arrival time. When the server has no valid estimate it returns null, and the page has no figure to show.

5.4.8 Feedback to the Passenger

The page confirms the important steps with coloured banners at the top of the screen. A blue banner confirms the selected boarding stop and a green banner confirms the created waiting request. The status of each request is shown in the list in a distinguishing colour, so that waiting and cancelled requests can be told apart at a glance (Figure 6.3.4).

5.5 Integration

Integration took place at three boundaries from the passenger interface's side. First, the page calls the JSON API under /api/ built by the server developer; Table 5.5.1 lists each passenger action and the call it triggers. Second, the passenger's waiting requests are consumed by the driver dashboard, which shows the number of passengers waiting at each stop of the bus's current trip, so a request created on the passenger page is what the driver sees (Figure 6.3.5). Third, the page relies on the driver dashboard indirectly, because the bus markers appear on the passenger map only after a driver has activated a bus, started a trip and sent a GPS update. The integration branch was used as the single source of truth.

Table 5.5.1: Passenger-interface actions and the API calls they trigger

Passenger action	Method	Endpoint (under /api/)
Register	POST	auth/register/
Login	POST	auth/login/
Logout	POST	auth/logout/
Open map, refresh (every 15 seconds)	GET	buses/
Type in Search route	GET	routes/search/?q=
Select a route	GET	routes/<route_id>/destinations/
Select a destination	GET	routes/<route_id>/stops/?destination=
Create Waiting Request	POST	waiting-requests/
Open My Requests	GET	waiting-requests/
Mark as Boarded	POST	waiting-requests/<id>/board/
Cancel Trip	POST	waiting-requests/<id>/cancel/
Complete the trip	POST	waiting-requests/<id>/complete/
Show arrival estimate	GET	waiting-requests/<id>/eta/

5.6 Challenges and Solutions

- **Giving a coordinate meaning.** A bus marker alone does not tell a passenger which bus it is or whether it is moving. The marker popup gives the registration number, status and speed, and the trip-planning panel relates buses to routes and stops through lists.
- **Choosing among many routes.** The GTFS data has 68 routes in two directions. Searching by typed name and then choosing a destination narrows the choice to one representative trip without asking the passenger to understand directions.

- **Keeping the map current without a push connection.** The page polls every 15 seconds and offers two refresh controls, so a passenger who doubts the display can update it at once.
- **Not misleading with the arrival time.** The estimate is shown only for a waiting request and only when the server returns a value, because a wrong estimate is worse than none.
- **Fitting several functions on one screen.** The map is the base screen and the trip planning and the request list open over it, so the passenger does not lose sight of the buses while planning.
- **Working while the server was changing.** The endpoints were developed in parallel with the page. The author and the server developer worked from a shared list of endpoints (Table 4.5.3.1), and the page was integrated against the final branch.
- **Change of the waiting-request model.** The request moved from a route-based to a trip-based design during development (Section 3.2). The page was adjusted so that a request names a route direction and a boarding stop.

5.7 Individual Contribution (Group Projects)

Ngosa Muleba designed and implemented the passenger interface: the passenger page with its live bus map, route search, boarding stop selection and waiting requests. He took part, jointly with the other members, in the problem analysis and requirements. He did not implement the server, the deployment configuration or the driver dashboard. 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
Passenger page: live bus map, markers, popups and 15-second refresh	Primary (author)	Sections 4.8.2, 5.4.2
Passenger page: route search and destination choice	Primary (author)	Sections 4.8.3, 5.4.4
Passenger page: boarding stop selection	Primary (author)	Sections 4.8.3, 5.4.5
Passenger page: waiting-request creation, list, controls and ETA display	Primary (author)	Sections 4.8.4, 5.4.6, 5.4.7
Passenger page: account area and feedback banners	Primary (author)	Sections 5.4.3, 5.4.8
Manual checks of the passenger flows on the deployed system	Primary (author), with the group	Section 6.3
Problem analysis, requirements and Yango interface analysis	Joint (group)	Section 3.3; all members used Yango as passengers
Literature review	Joint (group); adapted by the	Chapter 2 draws on the

Activity	Contribution	Notes
	author for this report	review prepared for the group report
Use-case diagram, system sequence diagram, architecture and domain model	Joint (group)	Shared analysis artefacts, reproduced from the group work
Django and DRF server, models, GTFS import, driver authorisation, GPS handling, waiting-request services, OSRM and ETA, automated tests, deployment configuration	Other group member (Kenneth Kaunda Musonda)	Not claimed
Driver dashboard	Other group member (Madalitso Nyirenda)	Not claimed

5.8 Chapter Summary

The passenger interface is a single browser page that shows the active buses on a Leaflet map refreshed every 15 seconds, guides the passenger from a route search through a destination and a boarding stop to a waiting request, and lists the passenger's requests with their status, the controls that suit it and the arrival estimate when there is one. It depends on the server for every rule and every figure it displays. Chapter 6 reports how it was checked.

CHAPTER 6: TESTING, RESULTS AND EVALUATION

6.1 Introduction

This chapter reports the testing evidence for the passenger interface and evaluates it against the objectives of Section 1.4, within its limits. Testing was conducted at two levels. The first was manual functional and integration checks of the deployed passenger interface working together with the driver dashboard and the server. The second was the server's automated tests, which verify the rules the passenger page relies on. The passenger interface itself has no automated tests, so the manual checks are the evidence for it. The testing was not exhaustive acceptance testing, and no performance benchmark, user study or field trial was carried out, so none is claimed. Where the evidence is absent the chapter says so.

6.2 Testing Strategy

The manual checks followed the passenger's path through the system: sign in, view the map, search a route, choose a destination and a boarding stop, create a waiting request, read its arrival estimate, use the controls on the request, and confirm that the driver dashboard shows the request. They were made on the deployed system using a test passenger account, a test driver account assigned to a test bus, and the location reported by the device used for the checks, which was not an operating bus. The screenshots in this chapter were taken during the checks. Each shows the outcome of a flow at one moment; they are not a table of expected and actual results with pass or fail status.

The server's automated tests use Django's test framework with Django REST Framework's APITestCase. Each test builds its own data, authenticates a test user and calls the real endpoint, and 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. They were written by the server developer. They are reported here because they cover the behaviour behind what the passenger sees: the arrival estimate, the scoping of requests to their owner and the access rules.

6.3 Test Cases and Results

6.3.1 Manual Checks of the Passenger Flows

Table 6.3.1 lists the passenger flows that were checked, the outcome observed and the figure that shows it. The word *Observed* means that the outcome was seen on the deployed system and is recorded in the figure; it is not a formal pass mark. The last rows state the flows for which no screenshot evidence exists.

Table 6.3.1: Manual checks of the passenger flows on the deployed system

No.	Flow checked	Outcome observed	Evidence
M1	Choose the passenger role	The landing page offers the passenger and driver sign-in	Figure 4.8.1

No.	Flow checked	Outcome observed	Evidence
	on the landing page		
M2	Sign in with the test passenger account	The passenger sign-in screen was shown and used	Figure 6.3.1
M3	View the active bus on the map	A bus marker appeared at the position reported by the driver's device	Figure 4.8.2
M4	Search a route by typing *bauleni*	The route Bauleni – Kulima Tower was offered in the selection list	Figure 6.3.2
M5	Choose a destination	The destinations Bauleni Junction and Kamwala Market – Independence Ave were offered as buttons	Figure 6.3.2
M6	Choose a boarding stop	The ordered stops were listed; choosing Civic Center produced the message *Selected boarding stop: Civic Center*	Figure 6.3.3
M7	Create a waiting request	A confirmation banner appeared and the request was listed in the WAITING status	Figure 6.3.4
M8	Read the arrival estimate	The WAITING request showed an estimate of about 10 minutes; not compared with an observed arrival	Figure 6.3.4
M9	Use the request controls	WAITING requests offered Mark as Boarded and Cancel Trip; earlier requests were listed as CANCELLED; an ON_BOARD request offered the control to complete the trip	Figures 6.3.4, 4.8.4, 4.8.5
M10	See the request from the driver's side	The driver dashboard listed one passenger waiting at Civic Center, the stop chosen by the passenger	Figure 6.3.5
M11	Not evidenced by a screenshot	Registering a new account, the password-reset link, the 15-second refresh interval, a request whose estimate is unavailable, the completed state, and use on different browsers and devices were not recorded	None

6.3.2 Walkthrough of the Passenger Flow

The passenger map (Figure 4.8.2) shows the bus reported by the driver's device as a marker. Figure 6.3.1 shows the passenger sign-in, which precedes any request. Figure 6.3.2 shows the route search, in which the text *bauleni* returned the route Bauleni – Kulima Tower, and the destinations offered for it. Figure 6.3.3 shows the choice of a boarding stop, with the confirmation banner. Figure 6.3.4 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 6.3.5 shows the same request with the driver dashboard alongside, which lists one passenger waiting at Civic Center.

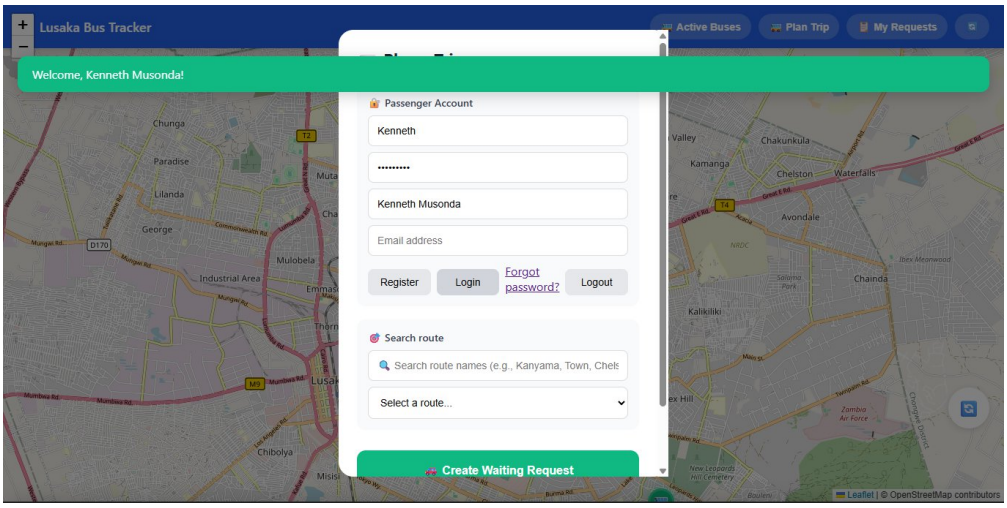


Figure 6.3.1: Passenger sign-in

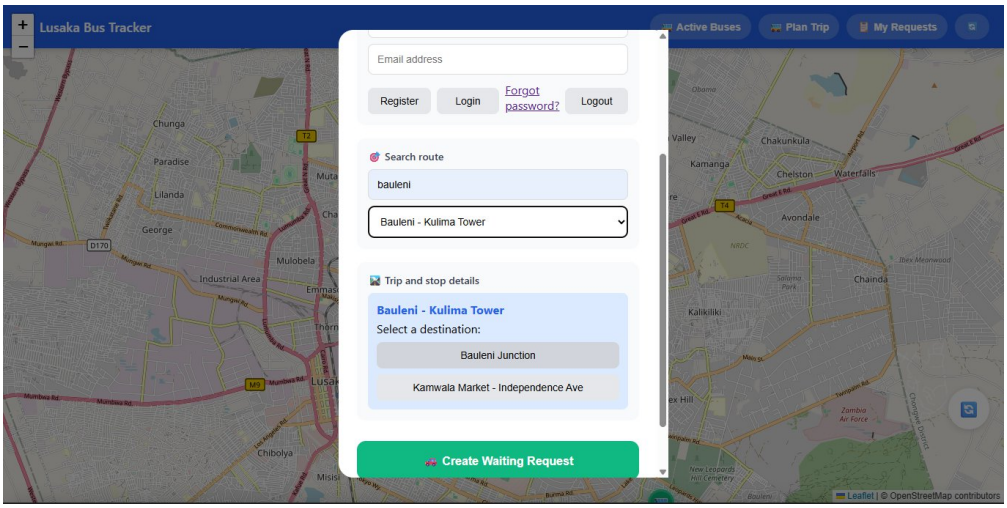


Figure 6.3.2: Route search and destinations offered

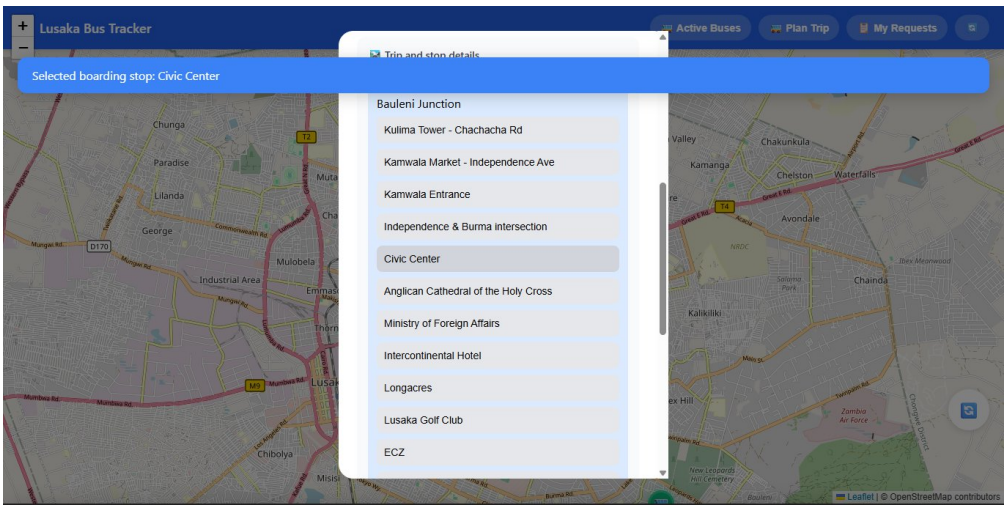


Figure 6.3.3: Boarding-stop selection

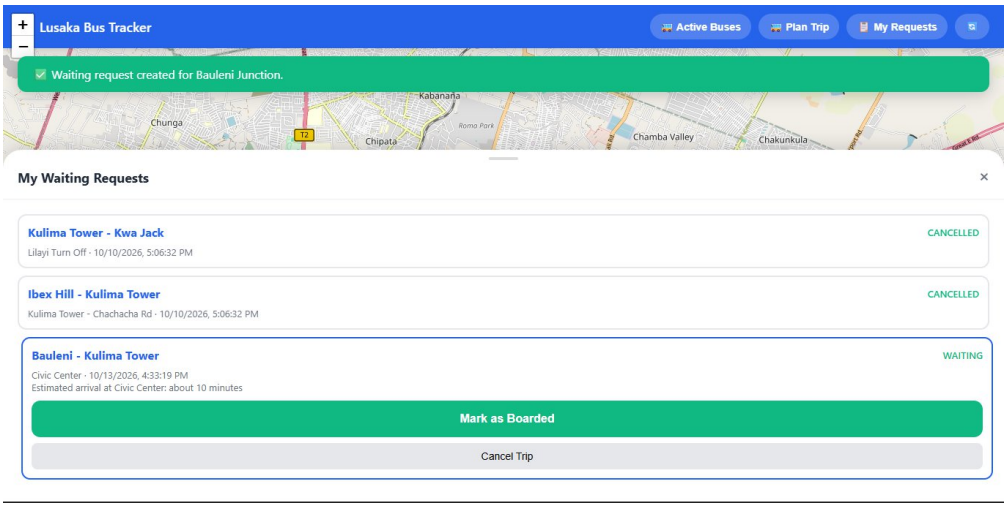


Figure 6.3.4: Waiting request created, with the arrival estimate and the options to board or cancel

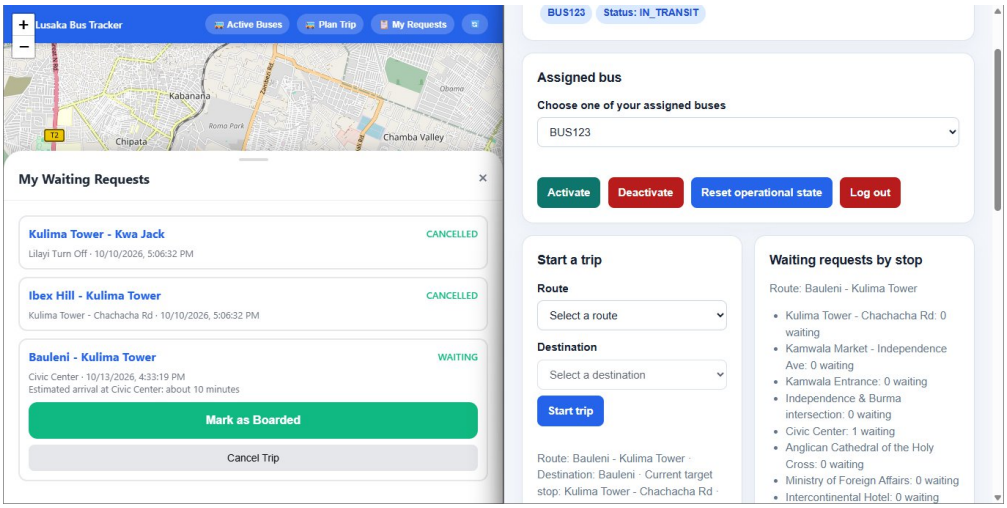


Figure 6.3.5: Passenger waiting request (left) and the driver dashboard showing one passenger waiting at Civic Center (right)

6.3.3 Automated Server Tests that Support the Passenger Flows

The server developer ran python manage.py test on the deployed working copy. Django found 35 tests, ran all 35 and reported OK. Table 6.3.2 lists the tests that bear on what the passenger sees and does. The run does not show that untested modules are correct, and it does not test the passenger page.

Table 6.3.2: Automated server tests that support the passenger flows

Behaviour	Condition and asserted outcome	Test class	Result
Shortest valid duration is chosen	Two eligible buses with different mocked durations: the smaller duration, in minutes, is returned	WaitingRequestEtaTests	PASS
No ETA when GPS is unavailable	A bus without coordinates: eta_minutes is null	WaitingRequestEtaTests	PASS

Behaviour	Condition and asserted outcome	Test class	Result
Passed stop is excluded	A bus beyond the requested stop: ETA null	WaitingRequestEtaTests	PASS
Non-WAITING requests have no ETA	Requests in other statuses: ETA null	WaitingRequestEtaTests	PASS
Passenger scoping	A different passenger asks for the request: HTTP 404	WaitingRequestEtaTests	PASS
Unavailable routing response	The mocked routing function finds no route: ETA null	WaitingRequestEtaTests	PASS
Public reads kept	Routes, stops, buses, route search, destinations and trip stops return 200 without signing in	ApiAccessSecurityTests	PASS
Default permission	Anonymous requests to the waiting-request endpoints return 403	ApiAccessSecurityTests	PASS
Waiting-request integrity	Update and delete return 405; client-supplied status, passenger and timestamps are ignored; another passenger cannot board, cancel or complete a request	ApiAccessSecurityTests	PASS
Passenger account required	Anonymous and non-passenger users receive 403 on every waiting-request path; reads stay limited to the owner	ApiAccessSecurityTests	PASS

These cases confirm, within the fixtures that the tests build, that the estimate returned to the passenger page is null when its inputs are missing or invalid, that it is the shortest valid duration otherwise, and that a request is visible only to its owner. They support the design decision that the page needs to show nothing of its own when no estimate exists.

6.4 Experimental Results / Performance Results

No performance experiment was carried out. There is no measurement of page load time, of the time between a driver's update and its appearance on the passenger map, or of behaviour with many buses or many passengers, 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. By design, a position reaches the passenger map no sooner than the next 15-second refresh.

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 manual checks give some confidence that the passenger interface, the driver dashboard and the server work together on the deployed system, and that the main path from the map to

a waiting request with an estimate can be followed. The server's automated tests give moderate confidence in the rules behind the estimate and the scoping of requests. Neither shows that the estimates are accurate for real buses, because the routing results are mocked in the tests, the manual run used one test device, and no observed arrival times were collected. Table 6.6.1 lists the limitations of the passenger interface found by reviewing its behaviour against the requirements. The severity is the author's assessment.

Table 6.6.1: Limitations of the passenger interface

ID	Finding	Impact	Severity and status
P1	No usability evaluation or user acceptance test was carried out	It is not known whether passengers can complete the flow unaided	Medium; open
P2	Buses are shown on the map but routes and stops are lists; route shapes are imported but unused	A passenger cannot see where a stop lies relative to a bus on the map	Low; open
P3	Data reaches the page by polling every 15 seconds	A position can be older than 15 seconds; no push updates	Low; open
P4	The passenger page has no automated tests	Faults in the page would not be detected automatically	Medium; open
P5	The map library is loaded from a content-delivery network and the tiles from OpenStreetMap	Without access to these services the map does not display	Low; open
P6	Only representative trips are used, so time of day is not modelled	The passenger chooses a route and destination, not a departure time	Low; open
P7	The accuracy of the displayed estimate has not been established	An estimate could differ from the real arrival	Medium; open
P8	A passenger may hold only one active request	A journey with a change of bus cannot be recorded in advance	Low; open (design rule)
P9	Small-screen layout, accessibility and browser compatibility were not evaluated	Use on other devices and by users with different needs is unverified	Medium; open

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	The passenger map shows the bus reported by the driver's device (M3, Figure 4.8.2); no field trial	Implemented; not field-tested
Group objective 2: passenger interface to view buses and routes	Marker with registration, status and speed, refreshed every 15 seconds; route, destination and stop lists (M3 to M6)	Met at prototype level
Group objective 3: Arrival Time	Estimate displayed with the waiting request (M8);	Implemented;

Objective	Evidence	Status
Prediction	server tests of its rules (Table 6.3.2); accuracy not evaluated	accuracy not established
Group objective 5: drivers see passengers at pickup points	The request created on the passenger page appears on the driver dashboard (M10)	Met at prototype level
Individual objective 1: live map with refresh	Section 5.4.2; M3	Achieved at prototype level
Individual objective 2: register, sign in and sign out	Section 5.4.3; M2; registration and the password-reset link not recorded (M11)	Partly evidenced
Individual objective 3: route search, destination and stops	Sections 5.4.4, 5.4.5; M4 to M6	Achieved
Individual objective 4: boarding stop and waiting request	Section 5.4.6; M6, M7; one-active-request rule enforced by the server	Achieved
Individual objective 5: view and manage own requests	Section 5.4.6; M9; completed state not recorded	Achieved; partly evidenced
Individual objective 6: show the arrival estimate	Section 5.4.7; M8	Achieved; accuracy not established
Individual objective 7: verify the flows on the deployed system	Table 6.3.1; no automated interface tests, no user study	Partly achieved

6.8 Individual Results

The results attributable to the author are the passenger interface and its manual verification (Table 5.7.1). The verified evidence is the set of manual checks in Table 6.3.1 and the screenshots that record them, together with the server tests in Table 6.3.2, which were written by another member and are cited as supporting evidence only. No individual result is presented as a performance, accuracy or usability outcome.

6.9 Chapter Summary

The available evidence shows by manual checks that a passenger can follow the path from the live map to a waiting request with an arrival estimate on the deployed system, and that the request reaches the driver. The server’s tests support the rules behind the estimate and the scoping of requests. The evidence does not support claims about accuracy, performance, usability or field operation. The main limitations are the absence of user, field and performance evidence and the open 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

Real-Time Bus Tracking (group objective 1). The passenger map displays the buses that the server holds, each with its registration number, status and speed. The checks show that a bus reporting from a driver's device appears on the map at the reported position. The method works for the implemented design, but the display is only as current as the last 15-second refresh and the driver's last update, and the thresholds behind the status have not been calibrated against field data.

Journey planning with GTFS data (group objective 2). The passenger interface offers the flow on top of the imported data. Imported routes, trips and ordered stops 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 the selection unambiguous. It does so by treating a route and a direction as the unit of planning, not a timetable, which suits a first prototype and limits what a passenger can plan.

Arrival time (group objective 3). The interface shows the estimate that the server calculates as the shortest road duration from OSRM among eligible buses, and nothing when none is valid. This keeps the interface honest, in line with the caution of Liu and Miller that real-time information does not automatically perform well [5]. Whether the numbers match real arrivals is not known.

Waiting passengers (group objective 5). A request created on the passenger page appears as a count on the driver dashboard. The passenger interface is therefore the source of the demand information that the driver sees, which is the part of the project in which passengers and drivers interact.

The interface as the point of use. The literature treats the effect of real-time information as dependent on how it is delivered [5]. The design choices of the passenger interface are attempts to deliver it well: the map as the permanent base screen, a short ordered sequence from route to request, confirmation messages at the key steps and controls that follow the status of the request. This report cannot say whether those choices work for passengers, because no usability study was done.

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 to passengers through a web interface, but it is far smaller; it has one passenger-facing channel, no multichannel delivery and no study of its effect on waiting time [3]. Unlike a ride-hailing application such as Yango, which was used as an interaction reference, the passenger does not request a vehicle for a private trip but registers at a stop on a fixed route. 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. The author's own contribution, set out in Section 5.7 and Appendix A, is the passenger interface and the analysis it rests on: the live map with refresh, the route search and destination choice, the boarding stop selection, the creation and management of waiting requests, the display of the arrival estimate, and the manual verification of these flows on the deployed system. A further contribution is the documentation in this report, including the user guide in Appendix B and the explicit list of limitations.

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 positions on the map could not be verified as those of an operating bus and live service changes could not be received. The passenger interface was verified against its design but not validated against an operating bus service. The findings apply to a prototype checked functionally on one test device. They do not apply to a city-wide, continuously operating service, and without user, field and performance evidence no claim can be made about usability, ETA accuracy or the effect on passengers. The passenger interface was verified only by manual checks, not by automated tests. The GTFS feed is a snapshot with a stated validity period, and keeping data current is an operational task that the project did not address.

7.6 Chapter Summary

The passenger-facing objectives were met at prototype level, with the arrival estimate implemented but its accuracy not established. 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, waiting requests, OSRM integration and ETA logic. The author's part was the passenger interface, which she designed and implemented, and the joint problem analysis and requirements. The system was verified by manual checks of the deployed interfaces and by the server's automated tests.

8.3 Conclusions

- A browser page can turn the state held by the server into usable passenger information: active buses on a map with their status and speed, a route-and-stop selection sequence built on GTFS data, and waiting requests that the driver can see (group objectives 1, 2 and 5).
- The arrival estimate calculated by the server can be presented to the passenger with a waiting request, and the interface shows it only when it exists (group objective 3).
- The three intended features, Real-Time Bus Tracking, Journey Planning and Arrival Time Prediction, are implemented at prototype level and reach the passenger through the passenger interface.
- The project does not show city-wide operation, ETA accuracy, usability, user acceptance or performance.

8.4 Achievement of Objectives

The passenger interface objective was met at prototype level, with bus locations viewed as map markers and routes as lists. Of the author's individual objectives, the live map, route search, stop selection, waiting requests, request management and estimate display were achieved at prototype level; account handling was only partly evidenced, and verification was partly achieved because the interface has no automated tests and no user study was done. Table 6.7.1 gives the evidence for each objective.

8.5 Recommendations

- Carry out a usability evaluation with passengers, using task-based tests and the System Usability Scale planned in the proposal, before any operational use (P1).
- Add automated tests of the passenger page, covering the route-to-request sequence, the controls shown for each status and the handling of a missing estimate (P4).
- Evaluate the page on a range of phones and browsers, and for accessibility (P9).
- Collect observed arrival times and compare them with the displayed estimates before the estimate is relied on (P7).
- Record the checks of registration, password reset, request completion and a missing estimate that were not captured in this report (M11).

8.6 Future Work

Future work may include: drawing the route and its stops on the map, using the imported shape data (P2); delivering updates by push instead of polling (P3); self-hosting the map library and handling the loss of tile services (P5); time-of-day schedules using all GTFS trips (P6); notifications when a bus is near the boarding stop; use of the passenger's own position to suggest the nearest stop; support for journeys with more than one bus (P8); and field trials with measured ETA accuracy. 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. [Online]. Available: https://gtfs.org/documentation/realtime/

[7] GTFS, "General Transit Feed Specification overview," General Transit Feed Specification Documentation, gtfs.org. [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)." [Online]. Available: https://digitaltransport4africa.org/

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

[13] V. Agafonkin, "Leaflet: an open-source JavaScript library for interactive maps," version 1.9.4. [Online]. Available: https://leafletjs.com/

[14] OpenStreetMap contributors, "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org/

APPENDICES

APPENDIX A: INDIVIDUAL CONTRIBUTION STATEMENT (GROUP PROJECTS)

Purpose of this Statement

This statement documents the individual contribution of Ngosa Muleba 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	Ngosa Muleba
Computer Number	2022044371
Supervisor	Dr. Lighton Phiri
Group Size	3 students
Group Members	Kenneth Kaunda Musonda (2020056283); Madalitso Nyirenda (2021502597); 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 passenger-interface developer: the passenger page, with its live bus map, route search, boarding stop selection and waiting requests, and, jointly with the other members, the problem analysis and requirements. The server, the deployment configuration and the driver dashboard were not part of this role.

Detailed Individual Contribution

The following were implemented by the author and are described in Chapters 4 to 6:

- the live bus map, with bus markers, popups showing registration number, status and speed, and refresh every 15 seconds and on request;
- the account area of the passenger page: register, login, forgot-password link and logout;
- the route search and the selection of a route from the matches;
- the destination buttons and the list of the ordered stops of the trip;
- the selection of a boarding stop, with its confirmation message;
- the creation of a waiting request and the confirmation message;
- the list of the passenger's requests with route, stop, time and status;
- the controls to mark a request as boarded, cancel it and complete it, and their display according to status; and
- the display of the arrival estimate with a waiting request.

Individual Deliverables

- **Passenger interface** — the passenger page of the Django project in the final branch.
- **Manual verification record** — the checks of Table 6.3.1 and the screenshots that record them.
- **Analysis artefacts in this report** — the passenger requirements (Table 4.3.1), the passenger use-case summaries, the interface design of Section 4.8 and the user guide in Appendix B.

Contribution to the Main System or Research Output

The passenger interface is the component through which the three features of the final system reach the passenger. It displays the buses that the server holds, presents the journey-planning data, creates the waiting requests that the driver dashboard counts, and shows the arrival estimate. The server and the driver dashboard, which the interface depends on, were developed by the other group members and are not claimed.

Individual Implementation and Technical Work

The technical work consisted of a single browser page written in HTML, CSS and JavaScript as a Django template; a Leaflet map on OpenStreetMap tiles with marker popups and a 15-second refresh; a trip-planning panel with route search, destination and stop selection; a request list with status-dependent controls; and the calls to the thirteen API operations listed in Table 5.5.1 using the Fetch API with the CSRF token. Chapter 5 describes each in detail.

Individual Testing and Evaluation Contribution

The author checked the passenger flows on the deployed system together with the driver dashboard and the server (Section 6.3). The server's automated tests were written by another

member and are cited as supporting evidence only. No user testing, field trial or performance measurement was conducted, and the report makes no claim about them.

Collaboration with Other Group Members

The passenger interface calls the server built by Kenneth Kaunda Musonda, so the behaviour of his endpoints was a dependency for the author's screens, and the buses on the passenger map are those that Madalitso Nyirenda's driver dashboard reports. The problem analysis, the Yango interface analysis, the requirements analysis and the literature review were shared, and the literature review was adapted for this report. The supervisor's review feedback led to the replacement of simulated GPS with real GPS.

Contribution Summary

(1) **Work primarily completed individually:** the passenger interface, its manual verification, and the interface design and user guide in this report.

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

(3) **Work primarily completed by other group members but used in or alongside the individual work:** the server, the deployment configuration, the automated tests and the driver dashboard.

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	Contributed	Shared with the group
Literature Review	Adapted for this report	Prepared jointly for the group report
Requirements Analysis	Contributed; passenger requirements	Yango analysis shared
System/Research Design	Passenger interface design	Use-case, sequence, architecture and domain diagrams shared; server design by another member
Implementation/Development	Passenger interface	Server and driver dashboard by other members
Testing and Evaluation	Manual checks of the passenger flows	Automated server tests by another member
Documentation	This individual report	Group report prepared jointly
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: Ngosa Muleba

Computer Number: 2022044371

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 — THE PASSENGER INTERFACE

This guide describes how a passenger uses the passenger page. It describes the controls of the deployed version shown in the figures of this report.

Opening the page

- Open the address of the deployed system in a web browser. On the landing page choose the passenger option (Figure 4.8.1).
- The map appears with the header bar *Lusaka Bus Tracker*. The header has four controls: Active Buses, Plan Trip, My Requests and a refresh control.

Watching the buses

- Active buses appear as bus markers. Touch or click a marker to see its registration number, status and speed.
- The map refreshes by itself every 15 seconds. Press the refresh control, in the header or at the lower right of the map, to refresh at once.
- If no marker appears, no bus is active: a bus appears only after its driver has activated it, started a trip and sent a position.

Registering and signing in

- Press Plan Trip. In the account area enter your details and press Register to create an account, or Login if you already have one.
- If you forget your password, press *Forgot password?* and follow the instructions sent to your e-mail address.
- Press Logout when you have finished.

Making a waiting request

- In Plan Trip, type part of a route name in *Search route* and choose the route from the list.
- Under *Trip and stop details* touch the destination you are travelling towards.
- From the list of stops, touch the stop where you will board. A message confirms *Selected boarding stop*.
- Press *Create Waiting Request*. A message confirms the request. You may have only one active request at a time.

Managing your requests

- Press My Requests to see your requests. Each shows the route, the boarding stop, the time and the status.

- While the status is WAITING you can see the estimated arrival, when available. Press *Mark as Boarded* when you board, or *Cancel Trip* if you no longer need the bus.
- When you are on board (ON_BOARD), use the control to complete the trip when you arrive.
- The estimate appears only when a bus on your trip is active, has a valid position and has not yet passed your stop. When no estimate is shown, none could be calculated.

APPENDIX C: ADDITIONAL SYSTEM DESIGNS / DIAGRAMS

Figure C.1 shows the entity-relationship diagram of the persisted data, prepared by the server developer. The passenger interface reads and writes the Passenger, WaitingRequest, Route, Trip, StopTime, Stop and Bus entities (Table 4.7.1).

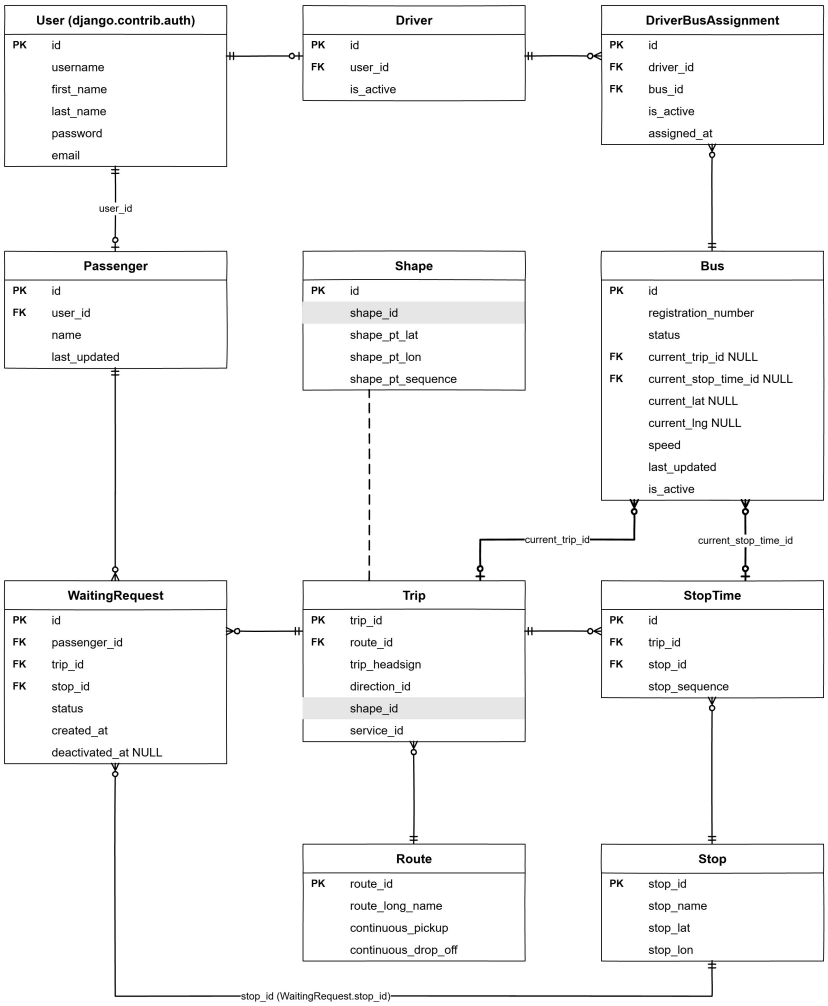


Figure C.1: Entity-relationship diagram of the persisted data

APPENDIX D: ADDITIONAL TEST CASES AND RESULTS

Table D.1 gives a script for the manual check of the passenger interface. The column *Status* states whether the case is recorded in the evidence of Chapter 6. Cases marked *To be run* were not captured in a screenshot and should be run and recorded before the interface is relied on.

Table D.1: Manual passenger test script

ID	Steps	Expected result	Status
D1	Open the landing page and choose the passenger option	The passenger page opens with the map and the header bar	Recorded (Figure 4.8.1)
D2	With a driver reporting a position, open the map	A bus marker appears; its popup shows registration number, status and speed	Recorded (Figure 4.8.2)
D3	Wait 15 seconds after the driver sends a new position	The marker moves without the page being reloaded	To be run
D4	Press each refresh control	The bus data is refreshed at once	To be run
D5	Register a new account in Plan Trip	The account is created and the passenger is signed in	To be run
D6	Sign in with the test passenger account	The passenger is signed in	Recorded (Figure 6.3.1)
D7	Press *Forgot password?*	The password-reset page opens	To be run
D8	Search *bauleni*	The route Bauleni – Kulima Tower is offered	Recorded (Figure 6.3.2)
D9	Select the route	Its destinations are offered as buttons	Recorded (Figure 6.3.2)
D10	Select a destination	The ordered stops of the trip are listed	Recorded (Figure 6.3.3)
D11	Select the stop Civic Center	The message *Selected boarding stop: Civic Center* appears	Recorded (Figure 6.3.3)
D12	Press Create Waiting Request	A confirmation appears and the request is listed as WAITING	Recorded (Figure 6.3.4)
D13	Press Create Waiting Request while a request is WAITING	The server refuses a second active request	To be run
D14	Open My Requests with an active bus on the trip	An estimated arrival in minutes is shown	Recorded (Figure 6.3.4)
D15	Open My Requests with no active bus on the trip	No estimate is shown	To be run
D16	Press Mark as Boarded	The status changes to ON_BOARD and the complete control appears	Recorded (Figure 4.8.5)
D17	Press the control to complete the trip	The status changes to COMPLETED	To be run
D18	Press Cancel Trip on a WAITING	The status changes to CANCELLED	Recorded (Figure

ID	Steps	Expected result	Status
	request		6.3.4)
D19	On the driver dashboard, view the waiting counts	One passenger is counted at the stop chosen	Recorded (Figure 6.3.5)
D20	Repeat D8 to D12 on a phone and on a second browser	The panels remain usable	To be run

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 lists the API endpoints that the passenger interface calls. All routes are under /api/. “Auth” states whether the endpoint requires a signed-in user in the final version of the system. The complete list for the whole system and the environment variables are in the group report and the project repository.

Table F.1: API endpoints used by the passenger interface

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/logout/	POST	Yes	Passenger sign-out
buses/	GET	No	Read-only list of bus records
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
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>/eta/	GET	Yes	ETA for own request

APPENDIX G: OTHER SUPPORTING MATERIAL

None.