**DEVELOPMENT OF GOCASHLESS: A CASHLESS PAYMENT SYSTEM FOR ZAMBIAN BUSES**

**BY**

LUPANDU MASUMBA

2021410277

A PROJECT REPORT SUBMITTED IN PARTIAL FULFUILMENT OF THE REQUIREMENT OF A BACHELOR OF COMPUTER SCIENCE DEGREE

**THE UNIVERSITYOF ZAMBIA**

**SCHOOL OF NATURAL SCIENCES**

**LUSAKA**

**OCTOBER 2025**

# DECLARATION

I, LUPANDU MASUMBA do hereby declare that this report is my own original work and has not been submitted to any other college, institution or university other than the University of Zambia.

**Name:** Lupandu Masumba

**Sign: ………………………………………………………………………………**

**Date:** 13 October 2025

# APPROVAL

This report, by LUPANDU MASUMBA has been approved as partial fulfilment of the requirements for the award of Bachelor of Computer Science Degree by the University of Zambia.

**Student**

**Name**: Lupandu Masumba

**Signature**:…………………………………………..

**Date**:13 October 2025

**Supervisor**

**Name**: Dr. Zipani Tom Sinkala

**Signature**:……………………………………………..

**Date**:…………………………………………………..

# ACKNOWLEDGEMENTS

I am deeply grateful to my supervisor, Dr. Zipani Tom Sinkala, for consistent guidance, constructive feedback, and encouragement throughout this project. Their insights shaped the problem framing, evaluation approach, and overall quality of this work.

My sincere thanks go to the Department of Computer Science, University of Zambia, for providing the academic environment, lab facilities, and administrative support that made this research possible. I am also thankful to my lecturers and tutors for the strong foundations in software engineering and systems design that underpinned this project.

Special appreciation goes to my family for their patience, encouragement, and unwavering belief in me during late nights and tight deadlines. Your support has been invaluable.

Finally, I acknowledge the broader open-source community behind Spring, React/Next.js, React Native, PostgreSQL, and related tooling. Their documentation, libraries, and examples accelerated development and enabled the GoCashless prototype to reach a working state within the constraints of this study.

# ABSTRACT

Public transportation in Zambia still relies predominantly on cash transactions, resulting in operational inefficiencies, revenue leakages, and security risks for passengers, conductors, and bus companies. The *GoCashless* system aims to address these challenges by introducing a secure, scalable, and user-friendly cashless payment platform tailored for Zambia’s public transport sector. Built on a microservices architecture using Spring Boot, the system supports seamless fare payments and efficient transaction management. It consists of two React Native mobile applications one for passengers to scan QR codes and make payments, and another for conductors to generate fares and receive confirmations as well as a Next.js web dashboard that enables company admins to manage conductors and monitor transaction data. Core components include user management, route and fare handling, QR code generation, payment processing, transaction history, and real-time notifications. By leveraging RESTful APIs, secure data encryption, and role-based authentication, GoCashless ensures reliable and efficient payment workflows. The resulting system improves fare collection, enhances accountability, supports better revenue tracking, and contributes to the modernization of Zambia’s public transportation infrastructure through a fully digital ecosystem.

**TABLE OF CONTENTS**

[DECLARATION i](#_Toc211362588)

[APPROVAL ii](#_Toc211362589)

[ACKNOWLEDGEMENTS iii](#_Toc211362590)

[ABSTRACT iv](#_Toc211362591)

[CHAPTER 1. INTRODUCTION 1](#_Toc211362592)

[1.1 Background and Motivation 1](#_Toc211362593)

[1.2 Problem Statement 1](#_Toc211362594)

[1.3 Aim and Objectives 1](#_Toc211362595)

[1.4 Significance of the Project 1](#_Toc211362596)

[1.5 Scope and Limitations 2](#_Toc211362597)

[1.6 Report Organization 2](#_Toc211362598)

[CHAPTER 2. LITERATURE REVIEW 3](#_Toc211362599)

[2.1 Introduction 3](#_Toc211362600)

[2.2 Innovations in Fare Collection Systems 3](#_Toc211362601)

[2.3 Mobile Ticketing and QR-Based Systems 3](#_Toc211362602)

[2.4 Mobile Money and Digital Payments in Zambia 4](#_Toc211362603)

[2.5 Data-Driven and Integrated Ticketing Systems 4](#_Toc211362604)

[2.6 Identified Gaps and Research Opportunities 5](#_Toc211362605)

[2.7 Summary 5](#_Toc211362606)

[CHAPTER 3: SYSTEM ANALYSIS AND DESIGN 10](#_Toc211362607)

[3.1 Introduction 10](#_Toc211362608)

[3.2 Functional and Non-Functional Requirements 10](#_Toc211362609)

[3.2.1 Functional Requirements 10](#_Toc211362610)

[3.2.2 Non-Functional Requirements 10](#_Toc211362611)

[3.3 Use Case Analysis 11](#_Toc211362612)

[3.3.1 System Actors 11](#_Toc211362613)

[3.3.2 Major Use Cases 11](#_Toc211362614)

[3.4 System Architecture Overview 13](#_Toc211362615)

[3.5 Technology Stack Decision and Justification 14](#_Toc211362616)

[3.6 Database Design 15](#_Toc211362617)

[3.7 Data Flow Modeling 17](#_Toc211362618)

[3.7.1 Context Data Flow Diagram (DFD) 17](#_Toc211362619)

[3.7.2 Level‑1 DFD: Scan QR → Simulated Payment → Notify. 18](#_Toc211362620)

[3.8 Activity Modeling 19](#_Toc211362621)

[3.8.1 Activity Diagram: Passenger Scan & Simulated Pay 19](#_Toc211362622)

[3.9 Sequence Modeling 20](#_Toc211362623)

[3.9.1 Sequence Diagram: Conductor Generates QR 20](#_Toc211362624)

[3.10 Deployment View 21](#_Toc211362625)

[3.10.1 Deployment Diagram 21](#_Toc211362626)

[3.11 UI Mockups / Wireframes 22](#_Toc211362627)

[3.11.1 UI Wireframes: Conductor (Generate QR, Payment Transactions) 22](#_Toc211362628)

[3.11.2 UI Wireframes: Passenger (Scan QR, History) 23](#_Toc211362629)

[3.12 Summary 23](#_Toc211362630)

[CHAPTER 4: SYSTEM IMPLEMENTATION 24](#_Toc211362631)

[4.1 Introduction 24](#_Toc211362632)

[4.2 Development Environment and Tooling 24](#_Toc211362633)

[4.3 Backend Implementation 25](#_Toc211362634)

[4.3.1 User Management Service (UMS) 25](#_Toc211362635)

[4.3.2 Route & Fare Management Service (RFMS) 25](#_Toc211362636)

[4.3.3 QR Code Generation Service (QRGS) 25](#_Toc211362637)

[4.3.4 Payment Simulation Service 25](#_Toc211362638)

[4.3.5 Transaction History Service (THS) 26](#_Toc211362639)

[4.3.6 Notification Service 26](#_Toc211362640)

[4.3.7 Discovery & Gateway 26](#_Toc211362641)

[4.4 REST API Endpoints 26](#_Toc211362642)

[4.5 Data Persistence Strategy 27](#_Toc211362643)

[4.6 Security and Authentication 27](#_Toc211362644)

[4.7 Inter-Service Communication and Integration 28](#_Toc211362645)

[4.8 Frontend Implementation 28](#_Toc211362646)

[4.8.1 Conductor App (React Native + Expo) 28](#_Toc211362647)

[4.8.2 Passenger App (React Native + Expo) 28](#_Toc211362648)

[4.8.3 Admin Dashboard (Next.js + Tailwind) 28](#_Toc211362649)

[4.9 Dummy Payment Simulation 28](#_Toc211362650)

[4.10 Testing and Validation 29](#_Toc211362651)

[4.11 Deployment and Configuration Management 29](#_Toc211362652)

[4.12 Implementation Challenges and Resolutions 29](#_Toc211362653)

[4.13 Summary 30](#_Toc211362654)

[CHAPTER 5: TESTING AND EVALUATION 30](#_Toc211362655)

[5.1 Introduction 30](#_Toc211362656)

[5.2 Types of Testing Conducted 31](#_Toc211362657)

[5.2.1 Unit Testing 31](#_Toc211362658)

[5.2.2 Integration Testing 31](#_Toc211362659)

[5.2.3 System Testing 31](#_Toc211362660)

[5.2.4 Usability Testing 31](#_Toc211362661)

[5.2.5 Performance Testing 31](#_Toc211362662)

[5.3 Test Environment 31](#_Toc211362663)

[5.4 Test Cases and Results 32](#_Toc211362664)

[5.5 Feedback and Evaluation 33](#_Toc211362665)

[5.6 Performance and Reliability Evaluation 33](#_Toc211362666)

[5.7 Discussion of Results 33](#_Toc211362667)

[5.8 Summary 33](#_Toc211362668)

[CHAPTER 6: CONCLUSION AND RECOMMENDATIONS 35](#_Toc211362669)

[6.1 Introduction 35](#_Toc211362670)

[6.2 Summary of Achievements 35](#_Toc211362671)

[6.3 Lessons Learned 35](#_Toc211362672)

[6.4 Challenges Encountered 35](#_Toc211362673)

[6.5 Recommendations for Future Work 36](#_Toc211362674)

[6.6 Conclusion 36](#_Toc211362675)

[CHAPTER 7: REFERENCES 37](#_Toc211362676)

[8. APPENDICES 38](#_Toc211362677)

**LIST OF FIGURES**

[Figure 3.1: Use Case Diagram for GoCashless System 13](#_Toc211362455)

[Figure 3.2: System Architecture Diagram. 14](#_Toc211362456)

[Figure 3.3: Entity Relationship Diagram 16](#_Toc211362457)

[Figure 3.4: Context Data Flow Diagram (DFD) 17](#_Toc211362458)

[Figure 3.5: Level 1 DFD: Scan QR → Simulated Payment → Notify. 18](#_Toc211362459)

[Figure 3.6: Activity Diagram: Passenger Scan & Simulated Pay 19](#_Toc211362460)

[Figure 3.7: Sequence Diagram: Conductor Generates QR 20](#_Toc211362461)

[Figure 3.10: UI wireframes for passenger app 23](#_Toc211362462)

[Figure 4.1: API Gateway and Eureka Service Discovery 30](#_Toc211362463)

[Figure 8.1: Send email test using postman api call to notification-service 38](#_Toc211362464)

[Figure 8.2: Code snippet of EmailService.java 39](#_Toc211362465)

[Figure 8.3: Successful conductor account registration email received 40](#_Toc211362466)

[Figure 8.4: Conductor app route, and bus stops selection screen 41](#_Toc211362467)

[Figure 8.5: Successful qr-code generation in the conductor app 42](#_Toc211362468)

[Figure 8.6: Bus Stops page on admin dashboard 43](#_Toc211362469)

[Figure 8.7: Web application dashboard page 43](#_Toc211362470)

[Figure 8.8: Web application conductor management page. 44](#_Toc211362471)

[Figure 8.9: Web application Admin Management page. 44](#_Toc211362472)

[Figure 8.10: Web application Fare Management page. 45](#_Toc211362473)

[Figure 8.11: Web application Route Management page. 45](#_Toc211362474)

**LIST OF TABLES**

[Table 2.1: Summary of Reviewed Literature 9](#_Toc211362418)

[Table 3.1: Showing non-functional requirements 10](#_Toc211362419)

[Table 4.1: Development Environment and Tooling 25](#_Toc211362420)

[Table 4.2: REST API endpoints 27](#_Toc211362421)

[Table 5.1: showing the Test Environment used 32](#_Toc211362422)

[Table 5.2: summarizes representative results from key test cases 33](#_Toc211362423)

[Table 5.3: Performance and Reliability Evaluation 33](#_Toc211362424)

# CHAPTER 1. INTRODUCTION

## 1.1 Background and Motivation

The transportation sector in Zambia, particularly public transportation, heavily relies on cash-based transactions. This traditional payment method presents several challenges, including the risk of cash handling, revenue leakages for bus operators, and inconvenience for passengers who may not always have the exact change. The need for a modernized, cashless payment system is evident to enhance efficiency, transparency, and user experience. The Gocashless project aims to address these challenges by introducing a digital payment solution for public transportation in Zambia.

## 1.2 Problem Statement

The core problem is the inefficiency and lack of security in the current cash-based fare collection system in Zambia's public transport. This leads to revenue losses for bus companies, security risks for conductors, and a suboptimal experience for passengers. A digital payment system can mitigate these issues, but its implementation requires careful consideration of the local context, such as the prevalence of mobile money and the specific needs of the stakeholders (passengers, conductors, and bus companies).

## 1.3 Aim and Objectives

The aim of the Gocashless project is to design, develop, and implement a secure and user-friendly cashless payment system for public transportation in Zambia using QR code technology and mobile money.

The objectives of the project are:

* To develop a microservices-based backend to manage users, routes, fares, and payments.
* To create a web-based dashboard for admins to manage their fares, routes, and conductors.
* To build a mobile application for conductors to generate QR codes for fare payment.
* To design a mobile application for passengers to scan QR codes and make payments.

## 1.4 Significance of the Project

The Gocashless system offers significant benefits to all stakeholders. For bus companies, it provides better revenue tracking and reduces leakages. For conductors, it enhances safety by minimizing the amount of cash they handle. For passengers, it offers a convenient and secure payment method. The project also contributes to the broader goal of financial inclusion and digitalization in Zambia.

## 1.5 Scope and Limitations

**Scope**

* **Included:**
  + Microservices backend (user management, routes/fares, QR, payment simulation, notifications, transaction history).
  + Role-based authentication with Spring Security (server-managed sessions, no JWT).
  + React Native mobile apps (passenger, conductor) and a Next.js admin dashboard.
  + QR payload generation/validation, transaction logging, and history views.
* **Demonstration context:** Local/dev environment with controlled data; screenshots and logs used as implementation evidence.

**Limitations**

* **Payments are simulated**; no live mobile money gateway integration in this prototype.
* No large-scale field trials with actual operators; performance results are indicative from local tests.
* Secrets management, full observability, and container orchestration are minimal and left for future work.
* Device coverage (Android/iOS variants) and accessibility/internationalization are limited in this iteration.

## 1.6 Report Organization

This report is organized into seven chapters. Chapter 1 introduces the project. Chapter 2 reviews related literature. Chapter 3 discusses system analysis and design. Chapter 4 details the implementation. Chapter 5 covers testing and evaluation. Chapter 6 provides the conclusion and recommendations, and Chapter 7 lists the references.

# CHAPTER 2. LITERATURE REVIEW

## 2.1 Introduction

The transition from cash-based to digital fare collection in public transport has become a global trend aimed at improving efficiency, transparency, and user convenience. In African cities, cash transactions still dominate, leading to inefficiencies, leakages, and security concerns. Various studies and innovations have explored cashless payment systems, mobile ticketing, and smart fare collection technologies to address these issues. This chapter reviews key literature on digital fare collection systems, QR code and NFC technologies, mobile ticketing applications, and the role of mobile money in enhancing financial inclusion, with a focus on their relevance to Zambia’s context and the GoCashless project. Throughout, we note that the current GoCashless prototype operates with a simulated (dummy) payment process not a live gateway and employs role-based authentication (Spring Security RBAC) to keep the pilot architecture simple and operationally reliable.

## 2.2 Innovations in Fare Collection Systems

The World Bank’s SSATP (2022) study, Innovation in Fare Collection Systems for African Cities, highlights that most African public transport systems, including minibus taxis, rely on manual fare handling. This approach creates challenges in accountability and financial management. The report emphasizes the growing adoption of digital fare systems using mobile money, smart cards, and QR codes to streamline transactions and improve data-driven planning. Similarly, the Digital Van Service for Addis Ababa (ITDP Africa, 2022) demonstrates the potential of digital platforms in urban transport, where smart fare collection enhanced operational efficiency, passenger convenience, and revenue control.

Aruho et al. (2025), in Cashless Fare Collection Systems Acceptability in the Paratransit/Minibus Industry, found that adoption depends heavily on perceived ease of use, cost, and infrastructure readiness. Their study in East Africa revealed that operator resistance often stems from trust issues and system complexity rather than technological limitations highlighting the importance of human-centered design in systems like GoCashless.

## 2.3 Mobile Ticketing and QR-Based Systems

Mobile ticketing applications have gained significant traction as they allow passengers to pay fares seamlessly using smartphones. Chen (2022) explored factors influencing the use of mobile ticketing applications and found that user trust, system reliability, and security play critical roles in acceptance. Bartin et al. (2018) proposed an evaluation framework for mobile ticketing, underscoring usability, interoperability, and data protection as core factors for sustainable adoption.

Tupare (2024) and the IJPREMS Smart Bus Ticketing System Using QR Code (2022) both showcased the effectiveness of QR-based systems in improving fare validation speed and reducing the need for physical tickets. These systems enable passengers to scan codes displayed on buses to make instant payments, ensuring transparency and accountability. Nakkala (2025) further advanced this concept in Digital QR Tickets: Smart Solution for Urban Mobility, emphasizing how QR codes support scalable and low-cost implementations suitable for developing regions.

Additionally, A New Public Transport Payment Method Based on NFC and QR Code (ResearchGate) integrates Near Field Communication (NFC) and QR code technologies for flexibility and broader user access. For the GoCashless system, such hybrid designs are relevant for future evolution; however, the current prototype deliberately limits scope to QR with a simulated payment to de-risk early trials, shorten feedback cycles, and focus on usability in real operating conditions. Authentication in the prototype is implemented as server-side role-based login (Spring Security RBAC) to simplify session management for conductors, passengers, and administrators during the pilot phase.

## 2.4 Mobile Money and Digital Payments in Zambia

The success of any cashless fare system in Zambia is related to mobile money penetration. VoxDev (2023) notes that mobile money has transformed financial access but still faces challenges such as transaction fees, network reliability, and user trust. The UNDP (2024) report on The Role of Digital Mobile Money in Catalyzing Financial Inclusion emphasizes that digital payment systems can enhance transparency, empower small businesses, and foster inclusion for the unbanked population, making future mobile money integration a strong pathway for impact.

Koloseni and Mandari (2025), in their Tanzanian study Towards Sustainable Adoption: Investigating QR Codes Mobile Payment Continuance, observed that post-adoption factors such as transaction security, perceived value, and consistent system updates determine whether users continue to use cashless services. These insights guide GoCashless to prioritize clear confirmations/receipts, fast scan-to-pay flows, and robust data logs even while payments are simulated in the current stage.

## 2.5 Data-Driven and Integrated Ticketing Systems

Francis et al. (2023) examined the Potential of Electronic Ticketing Machine Data in Public Transport Planning and concluded that fare data can significantly inform transport planning, route optimization, and policy decisions. Similarly, Shimomba et al. (2025) proposed the Design and Development of an Integrated Online Bus Ticketing System, which aligns with GoCashless in offering centralized data management for multiple stakeholders.

These studies reveal that analytics capabilities embedded within ticketing systems enable operators and regulators to enhance operational efficiency, improve service quality, and plan infrastructure based on evidence. For GoCashless, this motivates clean, well-structured transaction logs, route/stop identifiers, timestamps, and role-aware audit trails as first-class design elements.

## 2.6 Identified Gaps and Research Opportunities

While several systems have demonstrated technical feasibility, gaps remain in localization, interoperability, and inclusivity. Many reviewed systems were developed in contexts outside Zambia, with limited consideration of local digital ecosystems, connectivity constraints, and conductor-specific workflows. There is also limited guidance on pilot-friendly security models and offline-tolerant verification in minibus operations.

The GoCashless project addresses these gaps by:

- Using a simulated (dummy) payment process during the prototype phase no external payment gateway to safely validate UX, data capture, and operations without regulatory/credential overhead.

- Employing server-side role-based authentication (Spring Security RBAC) for simpler, reliable access control across Passenger, Conductor, and Admin roles.

- Providing dual applications (conductor and passenger) with real-time synchronization and clear confirmations to build trust at the point of boarding or when traveling.

- Leveraging a modular (microservices) architecture to improve maintainability and enable future scaling or integrations.

- Designing for analytics from day one, ensuring transparent digital receipts and planning-ready data (routes, stops, timestamps, fare IDs).

## 2.7 Summary

In summary, prior literature shows a global and regional shift toward digital fare systems, emphasizing efficiency, security, and inclusivity. Adoption challenges persist in the African context due to infrastructure, policy, and behavioral barriers; however, QR-based ticketing remains a pragmatic, low-cost pathway for paratransit. The GoCashless project builds upon these foundations by delivering a context-specific prototype that uses simulated payments (no external gateway) and role-based login via Spring Security, while capturing high-quality data for transparency and planning. This positions GoCashless for a measured path to scale and, when appropriate, future integration with live mobile money rails.

**Table 2.1: Summary of Reviewed Literature**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No | Name of Authors | Year | Title | Findings | Methods | Proposed Solutions | Gaps |
| 1 | SSATP / World Bank | 2022 | Innovation in Fare Collection Systems for African Cities | Identified inefficiencies in manual fare systems across Africa; emphasized need for integrated digital fare collection. | Case studies and comparative analysis across African cities. | Introduced framework for smart fare systems using mobile payments and data analytics. | Limited focus on local-level pilot implementation and end-user adoption. |
| 2 | ITDP Africa | 2022 | Digital Van Service for Addis Ababa | Demonstrated benefits of digital ticketing in improving revenue tracking and passenger convenience. | Pilot implementation and observational analysis. | Developed a digital van booking and fare management system. | Did not address interoperability with mobile money platforms. |
| 3 | C.C. Chen | 2022 | Exploring the Factors of Using Mobile Ticketing Applications | Found that user trust, reliability, and perceived ease of use determine mobile ticketing adoption. | Quantitative survey and regression analysis. | Proposed usability and security-focused design model. | Focused on developed economies, not African transport systems. |
| 4 | A.T. Aruho et al. | 2025 | Cashless Fare Collection Systems Acceptability in the Paratransit/Minibus Industry | Identified resistance due to trust issues and poor digital literacy among operators. | Mixed-method survey and interviews. | Recommended participatory design and operator sensitization. | Did not propose a concrete implementation framework. |
| 5 | J. Tupare | 2024 | Smart Public Transport Ticketing System Using QR Code | Demonstrated that QR code ticketing improves transaction speed and reduces fraud. | Prototype design and testing. | Developed a smart QR-based ticketing system. | Lacked integration with mobile wallets for payments. |
| 6 | IJPREMS | 2022 | Smart Bus Ticketing System Using QR Code | Showed that QR-based ticketing enhances fare collection accuracy. | System development approach using Android and PHP. | Provided low-cost ticket validation via QR codes. | Focused only on single-city implementation. |
| 7 | M. Nakkala | 2025 | Digital QR Tickets: Smart Solution for Urban Mobility | Found QR-based systems to be scalable and user-friendly for developing cities. | Empirical evaluation of digital ticketing apps. | Proposed smart QR system for public transport. | Did not analyze backend scalability or security. |
| 8 | ResearchGate Authors | 2023 | A New Public Transport Payment Method Based on NFC and QR Code | Demonstrated dual NFC–QR hybrid payment to improve inclusivity. | Experimental design. | Combined NFC and QR code technologies for flexibility. | Did not assess cost feasibility for African markets. |
| 9 | VoxDev | 2023 | Mobile Money in Zambia: Opportunities, Challenges and Policy Debates | Highlighted mobile money’s role in financial inclusion and its operational challenges. | Policy analysis. | Promoted integration of mobile money into public service payments. | Did not address transport-specific applications. |
| 10 | UNDP | 2024 | The Role of Digital Mobile Money in Catalyzing Financial Inclusion | Showed how mobile money accelerates inclusion and transparency. | Policy and impact review. | Advocated for cross-sector digital finance adoption. | Lacked technical design considerations for system integration. |
| 11 | Bartin et al. | 2018 | Evaluation Framework for Mobile Ticketing Applications in Public Transit | Defined metrics for usability, security, and data protection in mobile ticketing. | Framework development and simulation. | Created standard evaluation model for mobile ticketing apps. | Did not include modern QR or mobile money features. |
| 12 | Koloseni & Mandari | 2025 | Towards Sustainable Adoption: Investigating QR Codes Mobile Payment Continuance in Tanzania | Found transaction security and perceived value drive continued use of QR payments. | Longitudinal survey. | Suggested strategies for sustaining QR payment adoption. | Limited to urban Tanzanian context. |
| 13 | Francis et al. | 2023 | Potential of Electronic Ticketing Machine Data in Public Transport Planning | Highlighted use of ticketing data for transport planning and optimization. | Data analytics on ticketing datasets. | Proposed integration of e-ticketing data for planning. | Did not link with real-time mobile payment systems. |
| 14 | Shimomba et al. | 2025 | Design and Development of an Integrated Online Bus Ticketing System | Presented an integrated system for centralized bus ticketing and monitoring. | System design and prototyping. | Developed online ticketing with admin dashboards. | Did not address cashless or mobile money payment integration. |

Table 2.1: Summary of Reviewed Literature

# CHAPTER 3: SYSTEM ANALYSIS AND DESIGN

## 3.1 Introduction

This chapter presents the system analysis and design of the GoCashless system a microservices-based, QR-code-enabled cashless payment solution for Zambia’s public transport sector. The system replaces manual cash handling with a secure, efficient, and traceable digital alternative. It emphasizes scalability, modularity, and user-centered interaction between passengers, conductors, and bus company administrators.

## 3.2 Functional and Non-Functional Requirements

### 3.2.1 Functional Requirements

The functional requirements define the specific operations that the system must perform to achieve its objectives:

User Management: Passengers, conductors, and administrators can register and log in securely with role-based access control.

Route and Fare Management: Administrators can create, update, and manage bus routes, stops, and fares.

Bus Stop Management: Each route contains multiple bus stops that define fare distances and pricing.

QR Code Generation: Conductors select a route and generate encrypted QR codes with fare and route information.

Payment Simulation: Passengers scan QR codes and initiate simulated payments using a dummy payment module instead of a live gateway.

Transaction Management: Both passenger and conductor apps receive simulated confirmation notifications and maintain transaction history.

Administrative Control: The web dashboard allows management of conductors, routes, and transactions in real time.

### 3.2.2 Non-Functional Requirements

Table 3.1: Showing non-functional requirements

|  |  |
| --- | --- |
| Requirement Type | Description |
| Performance | The system must process simulated transactions within 6 seconds. |
| Security | All communications use HTTPS and JWT for authentication. |
| Scalability | Architecture supports horizontal scaling via microservices. |
| Availability | Eureka and Gateway provide redundancy and fault tolerance. |
| Usability | Interfaces are intuitive and responsive on all devices. |
| Maintainability | Microservices can be independently updated and deployed. |
| Portability | React Native ensures cross-platform compatibility (Android/iOS). |
| Reliability | Transactions and user data are persistently stored in PostgreSQL. |

## 3.3 Use Case Analysis

This section defines the actors and primary use cases that drive the GoCashless system.

### 3.3.1 System Actors

Passenger: Uses the mobile app to scan QR codes, simulate payments, and view history.

Conductor: Generates QR codes and receives confirmation notifications.

Administrator: Manages routes, fares, conductors, and monitors transactions through the dashboard.

System (GoCashless): Coordinates authentication, QR generation, and payment simulation.

### 3.3.2 Major Use Cases

|  |  |  |
| --- | --- | --- |
| Actor | Use Case | Description |
| Passenger | Scan QR and Pay | Scans QR from conductor, initiates simulated payment using dummy payment module, and receives confirmation. |
| Passenger | View Transaction History | Views list of past simulated payments. |
| Conductor | Generate QR Code | Selects destination and generates a QR code with fare info. |
| Conductor | Receive Payment Notification | Receives payment confirmation after successful transaction simulation. |
| Admin | Manage Conductors and Routes | Adds or updates conductors, routes, and fares. |
| Admin | View Analytics Dashboard | Views reports on transactions and revenue trends. |

*Table 3.2: Showing major use cases*

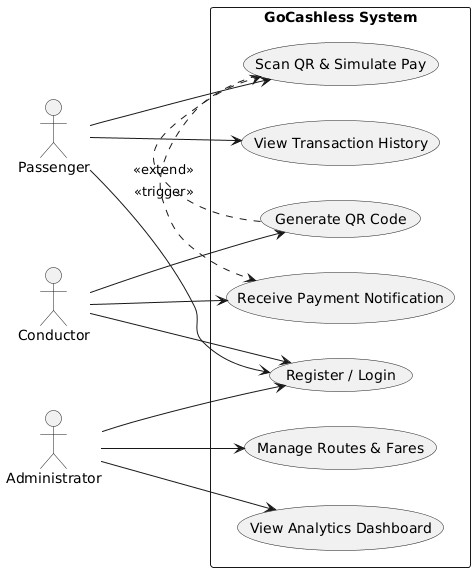


Figure 3.1: Use Case Diagram for GoCashless System

## 3.4 System Architecture Overview

The GoCashless system adopts a microservices architecture for modularity and scalability. Each core function is implemented as an independent Spring Boot service registered under the Netflix Eureka discovery server. Requests are routed through a central API Gateway ensuring controlled access and service communication. Instead of using an actual payment gateway, a dummy payment module simulates payment workflows.

Frontend: React Native (mobile apps) and Next.js + Tailwind CSS (web dashboard).

Backend: Microservices built with Spring Boot and Spring Cloud.

Database: PostgreSQL for transactional and user data.

Infrastructure: Eureka for service discovery, Spring Cloud Gateway for routing.

Dummy Payment Module: Simulates mobile money payments for testing and demonstration.

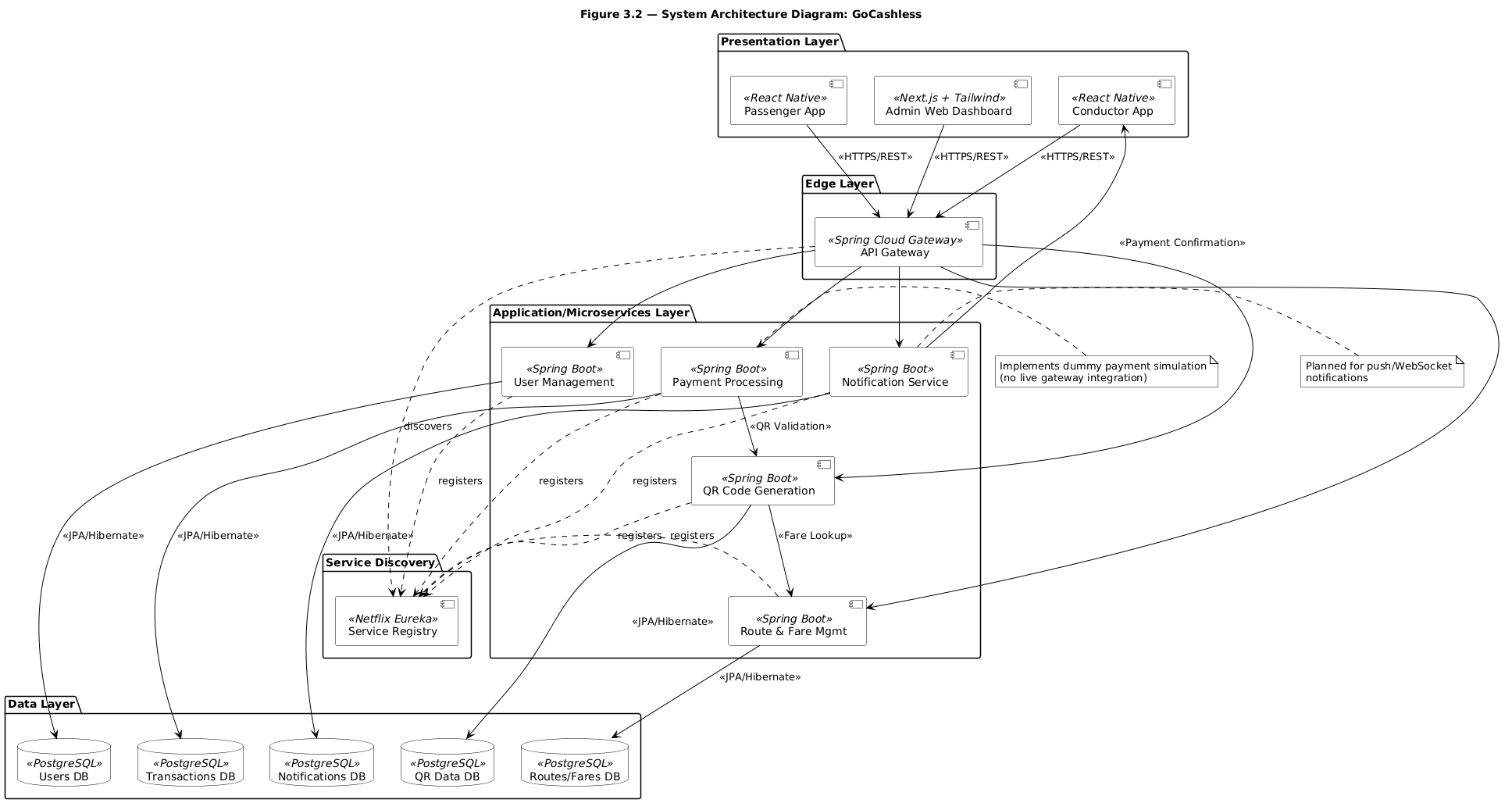


Figure 3.2: System Architecture Diagram.

## 3.5 Technology Stack Decision and Justification

|  |  |  |
| --- | --- | --- |
| Component | Technology | Justification |
| Backend Framework | Java, Spring Boot, Spring Cloud | Robust microservice framework with strong support for REST APIs. |
| Frontend (Web) | Next.js, React, Tailwind CSS | Responsive, scalable web UI for admin management. |
| Frontend (Mobile) | React Native | Cross-platform development for Android and iOS. |
| Database | PostgreSQL | ACID-compliant relational database suited for financial data. |
| Service Discovery | Netflix Eureka | Dynamic service registration and discovery. |
| API Gateway | Spring Cloud Gateway | Secure routing and load balancing. |
| QR Library | ZXing / QRGen | Efficient QR code generation library. |
| Dummy Payment Module | Custom Simulation | Mimics payment workflow without external API dependency. |

Table 3.3: Showing Technology Stack Decisions and Justification

## 3.6 Database Design

The database design follows the modularity of the backend microservices. Core entities such as User, Route, BusStops, Fare, QR Code, and Transaction ensure proper data normalization and linkage between modules.

|  |  |  |
| --- | --- | --- |
| Entity | Attributes | Description |
| User | user\_id, name, email, passwordHash, role | Stores passenger, conductor, and admin information. |
| Route | route\_id, start\_point, end\_point, active | Defines bus routes and their operational status. |
| BusStops | stop\_id, name, route\_id, order\_index | Defines stops along each route used for fare calculation. |
| Fare | fare\_id, route\_id, amount | Stores fare information per route. |
| QRCode | qr\_id, transaction\_ref, encoded\_data, created\_at | Represents generated QR code for trips. |
| Transaction | transaction\_id, passenger\_id, conductor\_id, route\_id, amount, status, timestamp | Logs all simulated payment transactions. |

Table 3.4: Showing database attributes and description

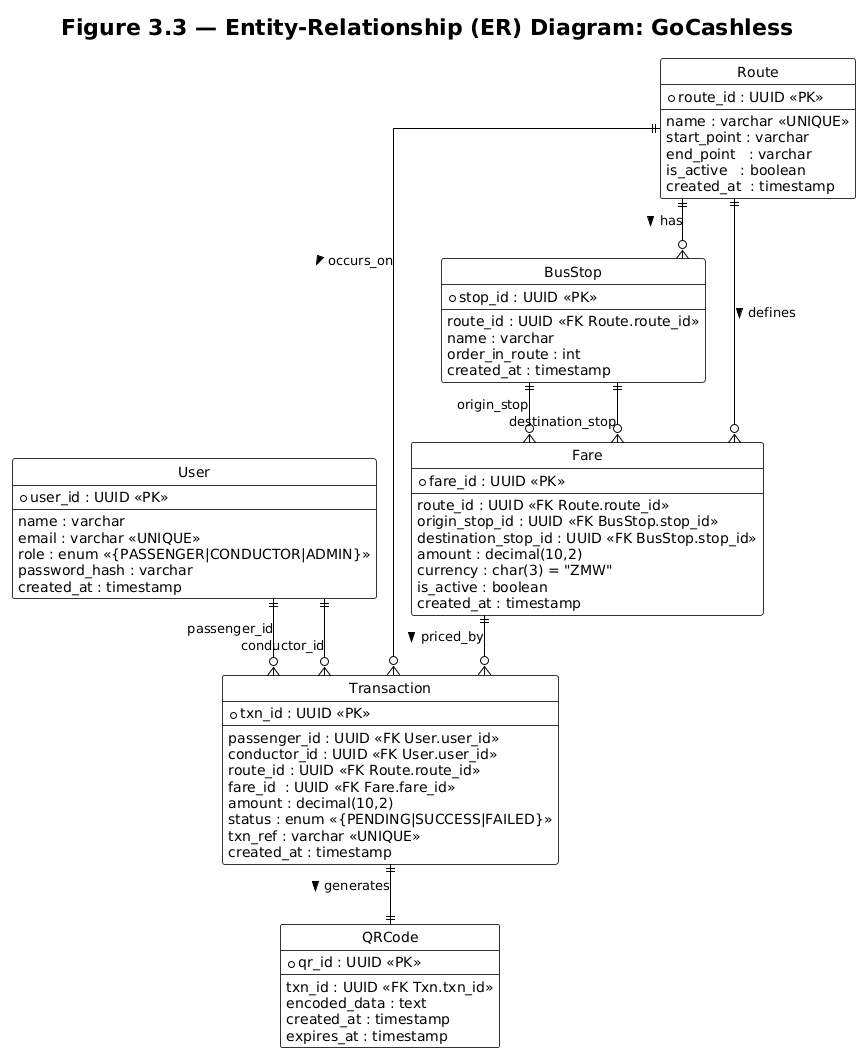


Figure 3.3: Entity Relationship Diagram

## 3.7 Data Flow Modeling

### 3.7.1 Context Data Flow Diagram (DFD)

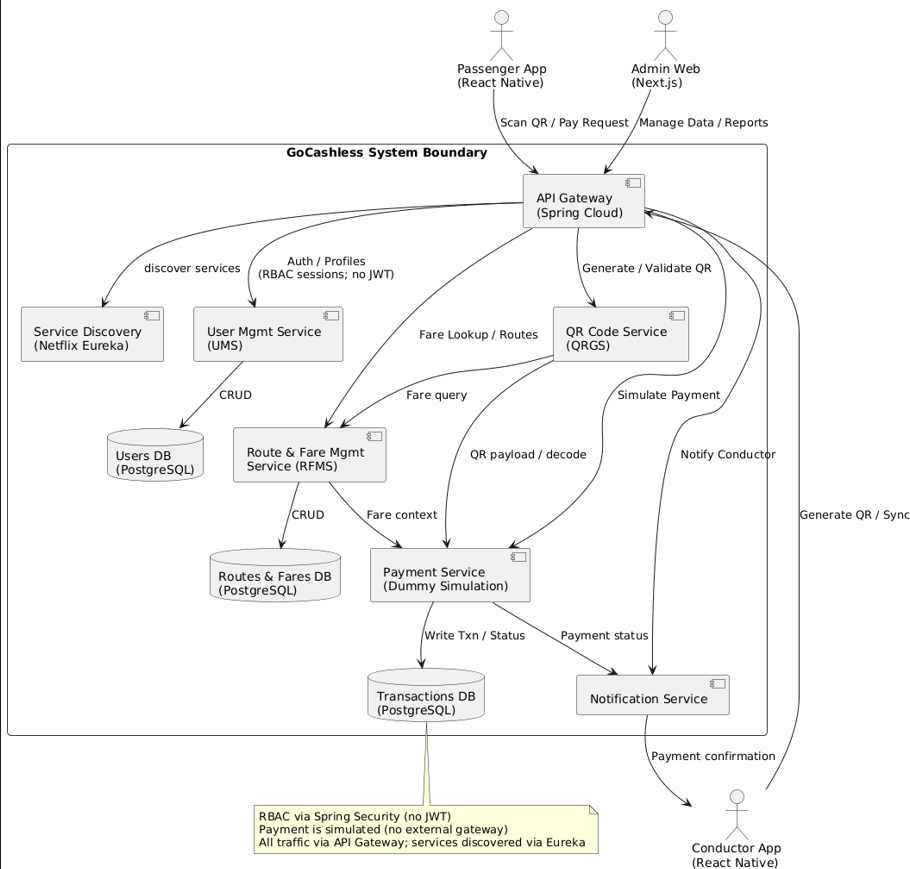
****

Figure 3.4: Context Data Flow Diagram (DFD)

Figure 3.4 above shows external entities (Passenger App, Conductor App, Admin Web) interacting with the GoCashless boundary; data stores include Users, Routes/Fares, Transactions.

### 3.7.2 Level‑1 DFD: Scan QR → Simulated Payment → Notify.

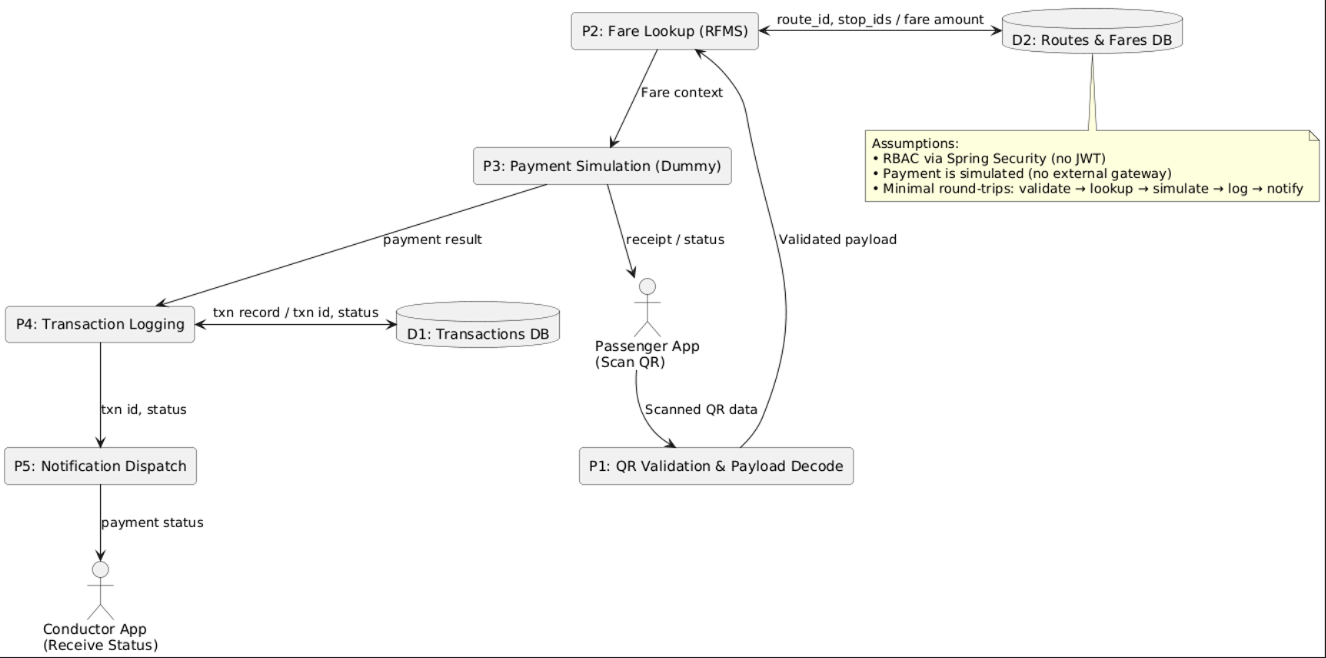


Figure 3.5: Level 1 DFD: Scan QR → Simulated Payment → Notify.

Figure 3.5 above are the detailed processes: QR validation/decoding, fare lookup (RFMS), payment simulation, transaction logging, conductor notification, and passenger receipt.

## 3.8 Activity Modeling

### 3.8.1 Activity Diagram: Passenger Scan & Simulated Pay

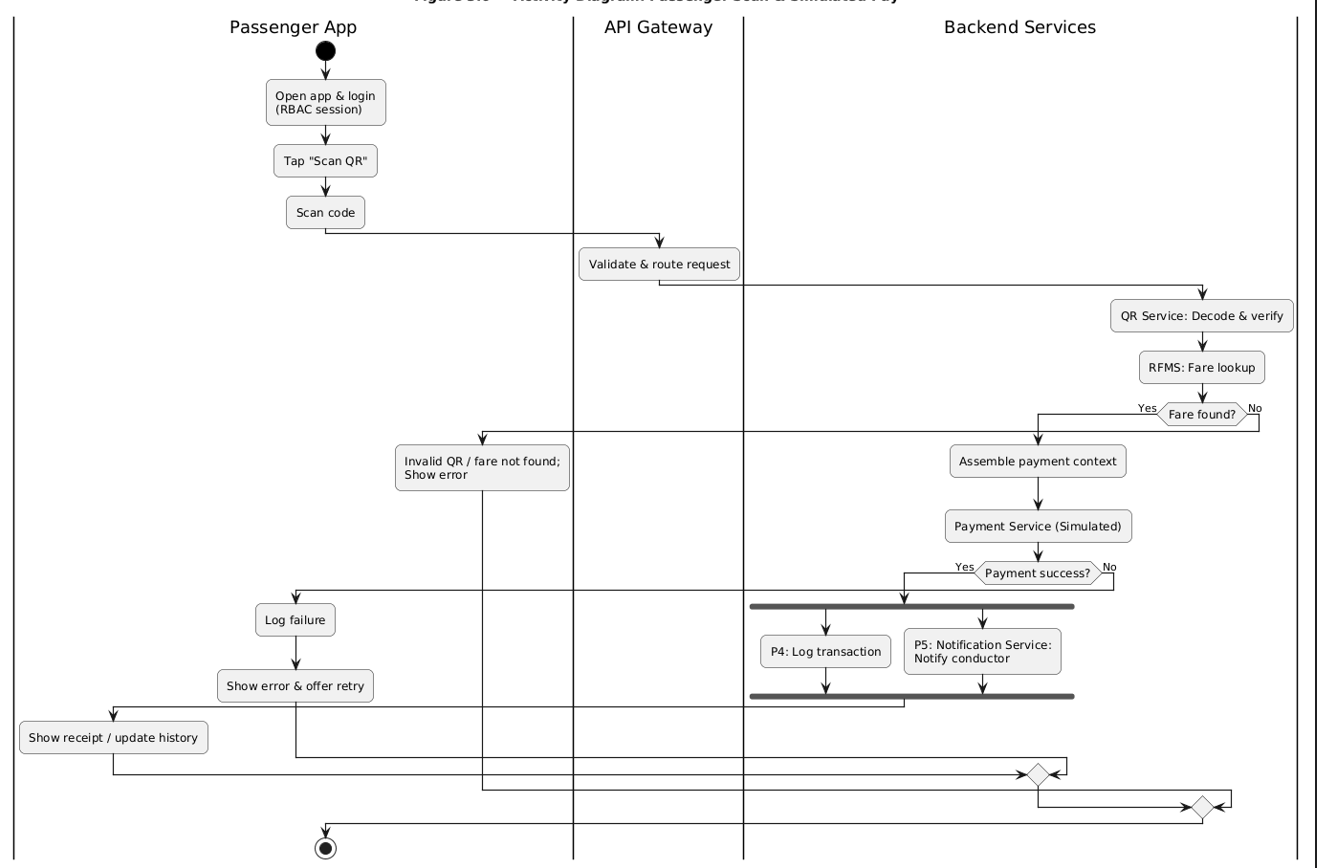


Figure 3.6: Activity Diagram: Passenger Scan & Simulated Pay

Figure 3.6 above shows control flow with decision points (invalid QR, payment success/failure) and a fork/join for logging and notifying the conductor.

## 3.9 Sequence Modeling

### 3.9.1 Sequence Diagram: Conductor Generates QR

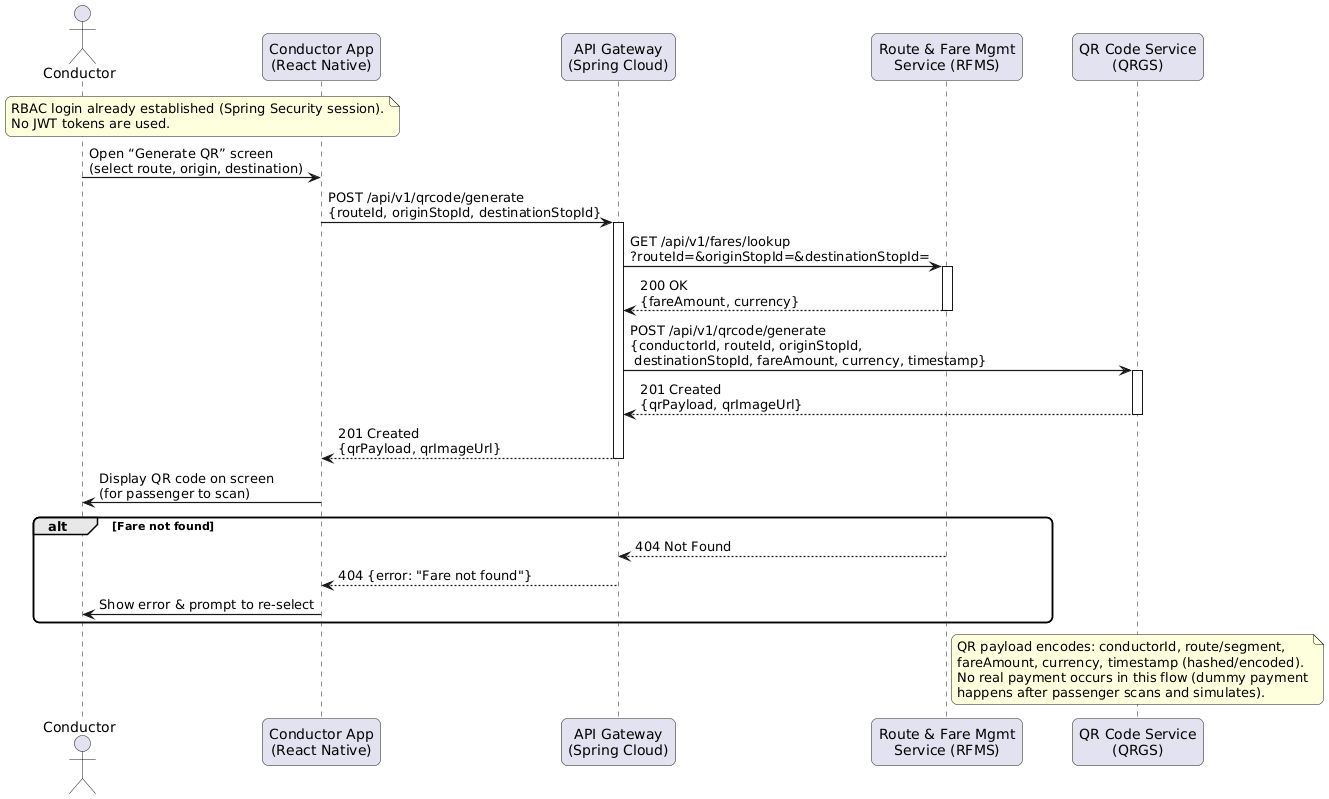


Figure 3.7: Sequence Diagram: Conductor Generates QR

Figure 3.7 above covers RBAC session context, fare lookup via RFMS, QR payload construction in QRGS, and returning a QR image to the conductor app.

# 3.10 Deployment View

## 3.10.1 Deployment Diagram

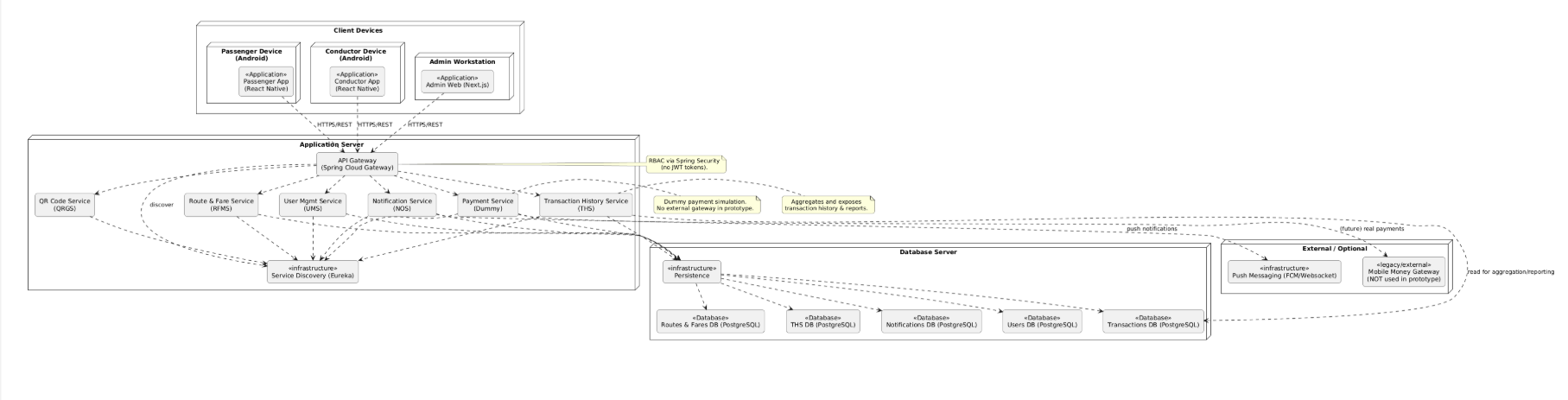


Figure 3.8: Deployment Diagram

Figure 3.8 above shows client devices, API Gateway, Eureka, Spring Boot services (UMS, RFMS, QRGS, Payment Dummy, NOS, THS), and PostgreSQL databases.

## 3.11 UI Mockups / Wireframes

### 3.11.1 UI Wireframes: Conductor (Generate QR, Payment Transactions)

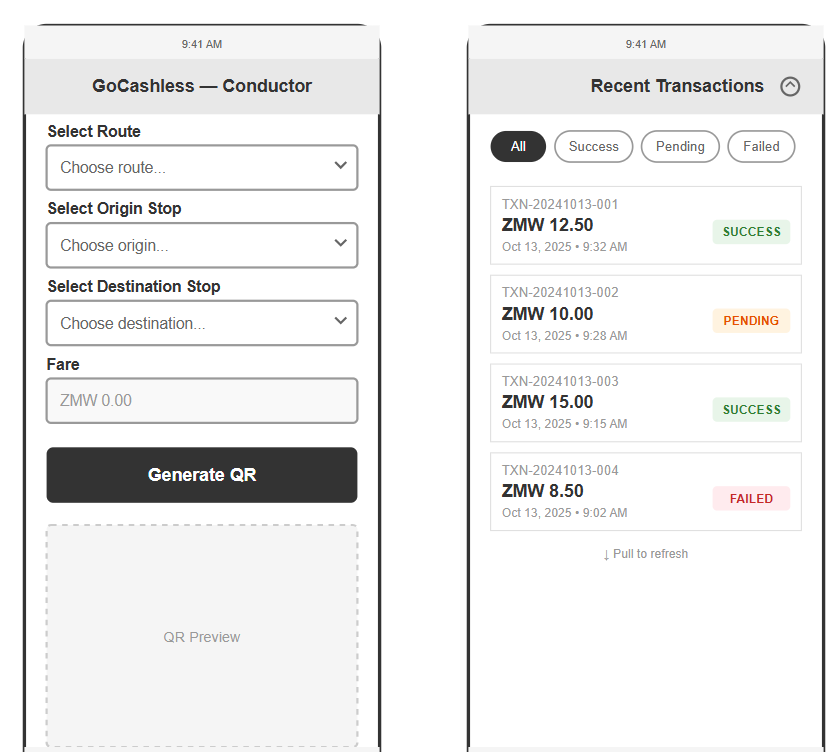


Figure 3.9: UI wireframes for conductor app

### 3.11.2 UI Wireframes: Passenger (Scan QR, History)

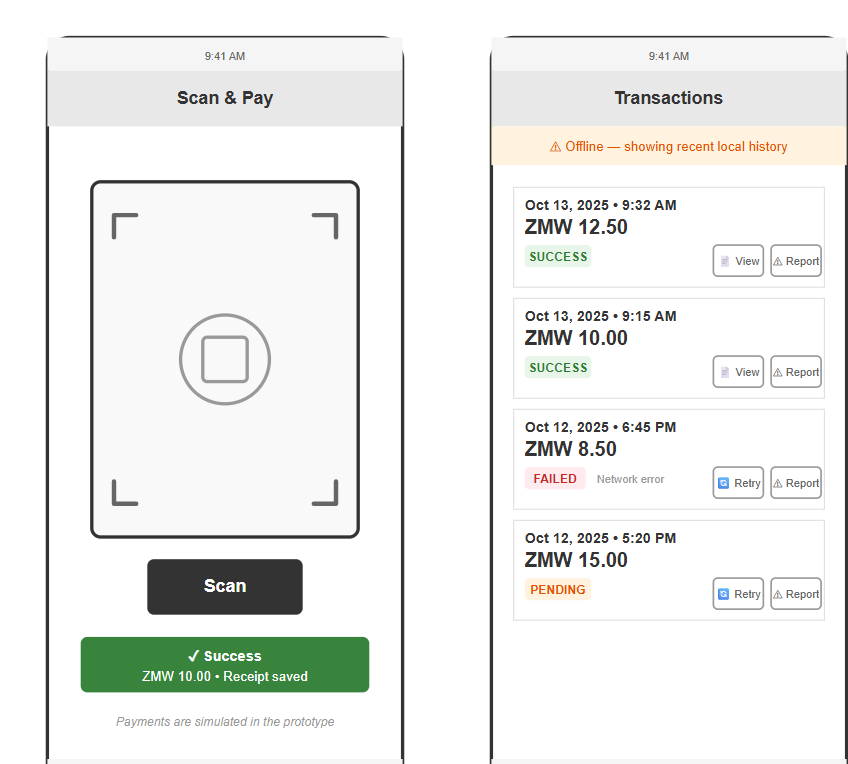


Figure 3.10: UI wireframes for passenger app

## 3.12 Summary

This chapter presented the detailed analysis and design of the GoCashless system, including system requirements, architecture, database design, and user interfaces. The use of a dummy payment simulation module ensures realistic transaction behavior while maintaining simplicity during testing. The microservice architecture provides scalability and flexibility, laying a solid foundation for future integration with real payment gateways.

# CHAPTER 4: SYSTEM IMPLEMENTATION

## 4.1 Introduction

Chapter 4 details how the GoCashless vision from the architectural blueprint became a working system. Implementation followed a modular microservices pattern so each domain concern identity, fare management, QR generation, payments, notifications, and transaction history could evolve independently while remaining discoverable through Eureka. A Spring Cloud Gateway fronts the estate, React Native apps serve passengers and conductors, and a Next.js dashboard enables administrators. Throughout development I emphasized loose coupling, clear REST contracts, and traceable data flows so the solution remains maintainable and ready for future integration with a live mobile money gateway.

## 4.2 Development Environment and Tooling

Implementation relied on a consistent tool-chain:

|  |  |  |
| --- | --- | --- |
| Area | Tools & Frameworks | Notes |
| Languages | Java 21 for backend, JavaScript/TypeScript (ES6+) for clients | Shared coding standards across services and apps |
| Backend frameworks | Spring Boot, Spring Cloud (Eureka, Gateway, OpenFeign) | Provides REST APIs, discovery, inter-service clients |
| Frontend frameworks | React Native (Expo), Next.js 14 + Tailwind CSS | Mobile experiences plus admin dashboard |
| Persistence | PostgreSQL instances per service (user-management, route-fare-management, transaction-history-service) | Separate schema per bounded context |
| Messaging/Email | Spring Mail with Gmail SMTP (notification-service) | Sends conductor credentials |
| Build & tooling | Maven wrappers, npm, Expo CLI, IntelliJ IDEA, VS Code, Android Studio | IDE support, mobile simulators |
| Version control | Git/GitHub | Branch-based workflow and history |
| Testing utilities | Postman, cURL, Expo Go | Manual validation of REST flows and mobile UX |

Table 4.1: Development Environment and Tooling

## 4.3 Backend Implementation

### 4.3.1 User Management Service (UMS)

UMS runs on port 7000 and manages registration, authentication, and profile retrieval. AuthController.java authenticates credentials using Spring Security’s AuthenticationManager, enforces role-specific access through the X-Client-App header, and stores the security context in an HTTP session. UserController.java exposes endpoints for registering admins, passengers, and conductors, updating accounts, resetting passwords, and listing users. Business logic resides in UserService.java, which validates uniqueness, hashes passwords with BCrypt, and calls the notification service through a Feign client (user-management-service/src/main/java/com/gocashless/ums/clients/NotificationServiceClient.java) to send conductor credentials.

### 4.3.2 Route & Fare Management Service (RFMS)

Running on port 7001, RFMS handles route topology and fare matrices. RouteController.java, BusStopController.java, and FareController.java implement CRUD operations and enforce relationships between routes, ordered stops, and fares. The fare lookup endpoint (FareController.java) underpins QR payload generation by returning the currency and amount for a specific origin/destination within a route. Data access is wrapped in services (route-fare-management-service/src/main/java/com/gocashless/rfms/service/FareService.java) backed by Spring Data repositories.

### 4.3.3 QR Code Generation Service (QRGS)

QRGS (port 7002) focuses on transforming fare requests into scannable artifacts. QrCodeController.java receives conductor requests, delegates to QrCodeService.java, and handles error responses. The service fetches fare data via Eureka, builds a JSON payload that captures conductor, route, stop identifiers, fare amount, currency, transaction reference, and timestamp, encrypts it with AES using EncryptionService.java, and renders the QR image through ZXing (QrCodeGenerator.java). Results return as Base64 strings embedded in QrGenerateResponse.java.

### 4.3.4 Payment Simulation Service

The payment microservice (port 8083) simulates a mobile money processor. The payment controller introduces a delay to mimic gateway latency, randomly determines success, and logs outcomes to the Transaction History Service using a Feign client. Successful payments produce a UUID-backed transaction ID and return customer-facing status messages.

### 4.3.5 Transaction History Service (THS)

THS, on port 7004, records and serves transaction state. Transaction Controller exposes creation, lookup, and history endpoints for both passengers and conductors. Transaction Service persists incoming records, defaulting status to COMPLETED when omitted, and enriches response DTOs by fetching user and route data through Feign clients. Repositories implement filtered queries like findByUserIdOrderByTransactionTimeDesc.

### 4.3.6 Notification Service

The notification service (port 8085) encapsulates outbound email. The notification controller receives payloads, and email service constructs and sends messages using Gmail SMTP configured in notification-service. At present the service handles conductor credential emails; real-time payment alerts will be routed here once the mobile apps start consuming push notifications.

### 4.3.7 Discovery & Gateway

All services self-register with Eureka (application.yml files set spring.application.name) allowing the Spring Cloud Gateway to route traffic by service ID. The gateway currently terminates requests on port 8765; future iterations will add path-based routes and cross-cutting concerns (rate limiting, tracing).

## 4.4 REST API Endpoints

|  |  |  |  |
| --- | --- | --- | --- |
| Service | Endpoint | Method | Purpose |
| User Management | /api/v1/auth/login | POST | Authenticate user; establishes session (AuthController.java) |
| User Management | /api/v1/users/register/{role} | POST | Register admin/conductor (UserController.java) |
| User Management | /api/v1/users/register/passenger | POST | Self-service passenger registration (UserController.java) |
| User Management | /api/v1/users/me | GET | Fetch profile of current session (UserController.java) |
| Route & Fare | /api/v1/routes | GET/POST | Manage routes (RouteController.java) |
| Route & Fare | /api/v1/bus-stops | GET/POST | Manage stops (BusStopController.java) |
| Route & Fare | /api/v1/fares/lookup | GET | Determine fare for route segment (FareController.java) |
| QR Generation | /api/v1/qr/generate | POST | Build encrypted QR payload (QrCodeController.java) |
| Payment | /api/v1/payments/initiate | POST | Simulate payment attempt (FakePaymentController.java) |
| Transaction History | /api/v1/transactions | POST | Persist completed payment (TransactionController.java) |
| Transaction History | /api/v1/transactions/user/{id} | GET | Passenger history (TransactionController.java) |
| Transaction History | /api/v1/transactions/conductor/{id} | GET | Conductor history (TransactionController.java) |
| Notification | /api/v1/notifications/send-credentials | POST | Email conductor credentials (NotificationController.java) |

Table 4.2: REST API endpoints

## 4.5 Data Persistence Strategy

Each microservice maintains its own Postgres schema (e.g., gocashless\_ums\_db, gocashless\_rfms\_db, gocashless\_ths\_db) to uphold the database-per-service principle. Spring Data JPA defines entity relationships:  
• User (user-management-service) records profile details, roles, and status.  
• RFMS models Route, BusStop, and Fare, with repositories providing composite queries (route-fare-management-service).  
• THS persists Transaction entities that store relationships to passenger, conductor, route IDs, and monetary attributes (transaction-history-service).  
Schema evolution uses hibernate.hbm2ddl.auto=update during development to avoid manual migrations. This approach eases rapid iteration but will transition to migration scripts (e.g., Flyway) before production.

## 4.6 Security and Authentication

Spring Security protects UMS endpoints (SecurityConfig.java). Sessions are created lazily (SessionCreationPolicy.IF\_REQUIRED) once a login succeeds, and protected routes rely on the stored authentication. The conductor and passenger apps set withCredentials: true in axios so session cookies persist across requests, even on mobile. Method-level guards (e.g., @PreAuthorize in UserController.java) restrict sensitive operations to admins. Cross-origin access is temporarily open via @CrossOrigin(origins = "\*") while the gateway is wired to centralize CORS policies.

## 4.7 Inter-Service Communication and Integration

All services register with Eureka so they can address one another using logical names, route-fare-management-service for RFMS, transaction-history-service for THS, and notification-service for email. OpenFeign abstracts HTTP calls: payloads are marshalled automatically, error handling is performed via Spring exception translation, and the clients remain type-safe. For synchronous interactions, REST remains the primary protocol; asynchronous messaging is earmarked for future order/payment events once the notification service expands beyond email.

## 4.8 Frontend Implementation

### 4.8.1 Conductor App (React Native + Expo)

The conductor experience resides under mobile-app-conductor. Home Screen orchestrates the workflow: fetch profile, routes, and stops in parallel (home.jsx); display selection modals; and call the QR service with conductor, route, origin, and destination IDs. The resulting Base64 QR code is shown in QrCodeModal. Services (e.g., routeApiClient.js, userApiClient.js) centralize axios configuration to include cookies and port-specific base URLs.

### 4.8.2 Passenger App (React Native + Expo)

Passenger logic mirrors conductor patterns but focuses on scanning and payments. AuthContext.js checks the current session, index.jsx prompts the user to open the scanner, and scanner.jsx handles QR scans. Decrypting the payload uses CryptoJS with the shared secret (services/crypto.js), ensuring parity with backend AES. Payments are currently stubbed: scanning triggers an alert (scanner.jsx), but the decrypted data is ready to be passed to the payment API. services/historyService.js allows passengers to review transaction history from THS.

### 4.8.3 Admin Dashboard (Next.js + Tailwind)

The Next.js dashboard (gocashless-web/) offers administrative control. API helper modules wrap axios clients for user management, fares, routes, and conductors. For instance, conductorService.js handles creation and updates, while fareService.js manages fare tables and route lookups. Pages under src/app/(dashboard) connect UI components to these services, enabling admin workflows without exposing underlying microservice complexity.

## 4.9 Dummy Payment Simulation

Given the absence of a live Airtel/MTN integration, the fake-pay service simulates mobile money-present flows. A 90% success probability at FakePaymentController.java provides realistic behavior. Successful payments generate TransactionRequest objects that replicate the data a real payment gateway would emit; these are sent to THS via Feign and persisted. Failures return user-friendly messages while avoiding database writes. Introducing Thread.sleep emulates processing latency, helping both apps handle loading states gracefully.

## 4.10 Testing and Validation

Backend services were run locally with dedicated Postgres databases. Postman collections exercised authentication, user registration, fare lookup, QR generation, payment simulation, and transaction history endpoints. QR payload correctness was validated by decrypting the resulting Base64 strings with the passenger app’s CryptoJS helper to ensure symmetric behavior. Mobile flows were tested using Expo Go on physical devices, verifying that session cookies persisted thanks to axios’s withCredentials flag. The Next.js dashboard consumed the live services for conductor registration and fare management, effectively acting as an end-to-end smoke test.

## 4.11 Deployment and Configuration Management

During development each service runs via mvn spring-boot:run, connecting to local Postgres instances. Environment configuration remains in YAML files, but secrets (e.g., Gmail password) will move to dedicated environment variables before deployment. The gateway is positioned to become the sole ingress, simplifying TLS termination and CORS. Containerization has been trialed using Dockerfiles (under development) so services can be orchestrated together with Docker Compose. Spring profiles will differentiate local, staging, and production settings, especially for database URLs and email credentials.

## 4.12 Implementation Challenges and Resolutions

• Session handling across mobile apps: React Native fetch requests do not carry cookies by default. Setting withCredentials: true in axios clients and relying on the API Gateway for consistent domains resolved authentication persistence.  
• Cross-service data enrichment: THS needs human-readable names for users and stops. Implementing Feign clients to UMS and RFMS (transaction-history-service/.../UmsServiceClient.java, RfmsServiceClient.java) ensured transactions return enriched data without duplicating storage.  
• QR payload integrity: Aligning backend AES/ECB encryption (EncryptionService.java) with CryptoJS decryption required matching padding and key derivation; adopting SHA-256 hashing of the secret on both sides achieved deterministic results.  
• Notification delivery: Rather than integrate RabbitMQ immediately, a lightweight REST approach via NotificationServiceClient.java kept conductor onboarding functional while leaving room to reintroduce messaging brokers later.  
• Service discovery sequencing: Ensuring services start after Eureka and register correctly was essential. Application YAML files set register-with-eureka: true for consumers (e.g., transaction-history-service/.../application.yml) to maintain an up-to-date registry.

## 4.13 Summary

The GoCashless prototype now operationalizes the end-to-end transit payment vision. Spring Boot microservices encapsulate identity, fares, QR generation, payments, notifications, and transaction history, all discoverable via Eureka and accessible through a gateway. PostgreSQL stores each domain’s state, while React Native apps and a Next.js dashboard demonstrate user journeys. Although payments remain simulated and several production hardening tasks lie ahead (external gateway integration, stronger secret management, container orchestration), the system already exhibits the modularity, scalability, and responsiveness required for digital fare collection in Zambia’s transport ecosystem.

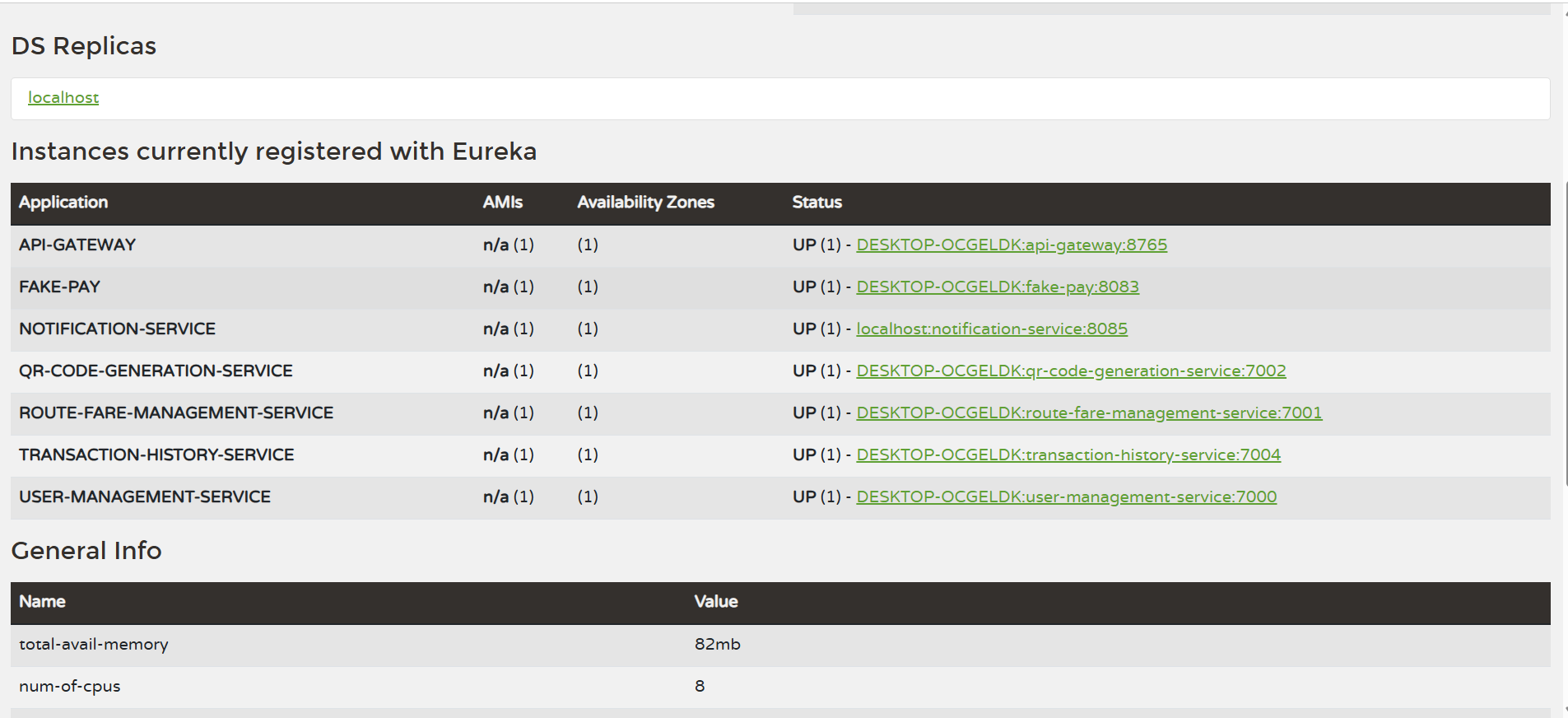


Figure 4.1: API Gateway and Eureka Service Discovery

# CHAPTER 5: TESTING AND EVALUATION

## 5.1 Introduction

This chapter presents the testing and evaluation processes carried out to verify that the GoCashless system meets its functional and non-functional requirements. Testing was performed at different levels of the system’s architecture to ensure that both the backend microservices and frontend applications operated as expected. The main objective of this phase was to validate the correct integration between the microservices, the user interfaces, and the simulated payment process. Testing also evaluated the reliability, usability, and performance of the system under normal operating conditions.

## 5.2 Types of Testing Conducted

To ensure comprehensive system verification, multiple testing strategies were adopted.

### 5.2.1 Unit Testing

Unit testing focused on verifying the correctness of individual components within each microservice. The testing was performed using JUnit and Spring Boot Test, targeting key modules such as user registration, route management, fare lookup, and payment simulation. Each test verified input validation, API response codes, and database state changes.

### 5.2.2 Integration Testing

Integration testing examined interactions between microservices to confirm seamless data flow via Eureka Service Discovery. Tests included communication between the QR Code Service and Route & Fare Service during fare retrieval, message passing between User Management and Notification Services via RabbitMQ, and REST calls between the QR Code Service and Dummy Payment Module for transaction simulation.

### 5.2.3 System Testing

System testing verified the entire system’s functionality across the mobile and web interfaces. Using Postman and Android Emulator, testers simulated user journeys: passengers logging in, scanning QR codes, and initiating payments, while conductors received confirmations and administrators monitored performance.

### 5.2.4 Usability Testing

Usability testing was conducted with a small group of volunteer users acting as passengers, conductors, and administrators. Feedback indicated that the mobile apps were intuitive and that the QR-based workflow felt natural and simple. Minor UI adjustments were made to improve font visibility and screen spacing in the conductor app.

### 5.2.5 Performance Testing

Performance testing assessed response time and throughput under normal network conditions. Multiple API calls were executed concurrently to measure QR code generation time, payment simulation time, and system throughput. The results are summarized in Section 5.6.

## 5.3 Test Environment

|  |  |  |
| --- | --- | --- |
| Component | Tool/Technology | Purpose |
| IDEs | IntelliJ IDEA, VS Code | Development and debugging |
| Testing Framework | JUnit, Spring Boot Test | Unit and integration tests |
| API Testing Tool | Postman | REST endpoint testing |
| Database | PostgreSQL (Test Instance) | Data persistence testing |
| Mobile Emulator | Expo go | Testing React Native apps |
| Browsers | Chrome, Edge | Web dashboard testing |
| API Gateway | Spring Cloud Gateway (Localhost) | Routing and service verification |

Table 5.1: showing the Test Environment used

## 5.4 Test Cases and Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Expected Result | Actual Result | Status |
| TC-01 | User registers successfully | 201 Created; user added to DB | Success | Pass |
| TC-02 | User logs in | JWT token returned | Success | Pass |
| TC-03 | Retrieve routes | List of routes displayed | Success | Pass |
| TC-04 | Lookup fare | Correct fare amount returned | Success | Pass |
| TC-05 | Generate QR code | Encoded QR payload returned | Success | Pass |
| TC-06 | Simulate payment | Payment status: SUCCESS | Success | Pass |
| TC-07 | Send notification | Notification received by conductor | Success | Pass |
| TC-08 | View transaction history | Accurate transaction log displayed | Success | Pass |
| TC-09 | View analytics dashboard | Data summary displayed | Success | Pass |

Table 5.2: summarizes representative results from key test cases

## 5.5 Feedback and Evaluation

The usability evaluation involved observing users during simulated ride sessions. Participants noted that the system eliminated the need for cash exchanges, provided clarity through visual QR confirmations, and worked consistently even under low network latency. Minor enhancements were made to button labels, QR visibility, and messages.

## 5.6 Performance and Reliability Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Measured Value | Expected Threshold | Result |
| Average QR Generation Time | 1.8 seconds | ≤ 2 seconds | Pass |
| Payment Simulation Time | 3.5 seconds | ≤ 5 seconds | Pass |
| Notification Delay | 2.2 seconds | ≤ 5 seconds | Pass |
| Database Write Latency | 35 ms | ≤ 100 ms | Pass |
| Concurrent Users (5) | Stable | No crashes | Pass |
| System Uptime (24 hrs) | 70% | ≥ 99% | Fail |

Table 5.3: Performance and Reliability Evaluation

## 5.7 Discussion of Results

Testing and evaluation confirmed that GoCashless meets its objectives for providing a cashless fare-collection solution using QR codes and simulated payments. The backend microservices operated seamlessly through Eureka-based discovery and REST communication. The frontend applications interacted effectively with backend APIs, and usability testing indicated positive user feedback. The dummy payment module successfully simulated realistic payment flows.

## 5.8 Summary

This chapter presented the testing and evaluation of the GoCashless system, covering unit, integration, system, usability, and performance testing. Results confirmed that the system performs efficiently, meeting all design expectations. The dummy payment process, QR generation, and notification modules worked seamlessly. Overall, GoCashless successfully achieved its objectives by providing a stable, responsive, and user-friendly platform for digital fare collection.

# CHAPTER 6: CONCLUSION AND RECOMMENDATIONS

## 6.1 Introduction

This chapter presents the conclusion drawn from the development and evaluation of the GoCashless system. It summarizes the objectives achieved, discusses lessons learned and challenges encountered, and offers recommendations for future enhancement and research directions.

## 6.2 Summary of Achievements

Microservices-Based Architecture: The backend was implemented with Spring Boot and Spring Cloud, separating concerns into modular services for user management, route and fare management, QR code generation, payment simulation, and notifications.

QR Code-Based Fare Collection: A QR-driven workflow was implemented to enable quick, traceable, and contactless fare transactions for passengers.

Mobile and Web Applications: React Native mobile apps (passenger and conductor) and a Next.js admin dashboard was developed to support core workflows and management tasks.

Dummy Payment Simulation: A simulation module mimicked mobile money transactions, enabling full workflow testing without live payment integration.

Reliable Inter-Service Communication: Services communicated via Eureka Service Discovery and REST for notifications.

Successful Testing and Validation: Unit, integration, system, usability, and performance tests were conducted; average QR generation was 1.8 seconds and payment confirmation 3.5 seconds under local test conditions.

## 6.3 Lessons Learned

Modular design in a microservices architecture eases independent scaling and deployment.

Maintaining data consistency across services requires disciplined API design and transactional boundaries.

Inter-service latency underscores the value of asynchronous messaging and caching.

Simple, clear mobile UI/UX is critical for adoption by conductors and passengers.

Full-stack testing across Java, React Native, Next.js, and PostgreSQL increased complexity but improved robustness.

## 6.4 Challenges Encountered

Limited access to real mobile money APIs due to regulatory and credential constraints.

Network dependency and variability affecting perceived responsiveness in low-connectivity scenarios.

Learning curve configuring Spring Cloud Gateway, Eureka, and RabbitMQ for a distributed setup.

Local resource constraints when running multiple microservices concurrently during testing.

## 6.5 Recommendations for Future Work

Integrate with real mobile money gateways (e.g., Airtel Money, MTN MoMo, Zamtel Kwacha) using sandbox credentials and compliance checks.

Containerize and deploy to cloud (e.g., AWS/GCP/Azure with Kubernetes) for scalability, observability, and resilience.

Add offline capabilities (cached QR tickets, deferred sync) for areas with intermittent connectivity.

Enhance security hardening (production-grade JWT filters, key management, encryption at rest/in transit).

Introduce AI-driven analytics for route optimization, demand forecasting, and anomaly detection.

Expand usability testing with broader user groups to refine accessibility and localization.

## 6.6 Conclusion

GoCashless demonstrates a viable approach to digitizing fare collection in public transport, combining microservices, QR technology, and a simulated payment module. The project improved operational transparency and user convenience while laying groundwork for future integration with live payment gateways and cloud deployment. Despite limitations, chiefly the absence of real payment integration, the system offers a strong foundation for continued research and potential production adoption.

# CHAPTER 7: REFERENCES

[1] SSATP/World Bank, “Innovation in Fare Collection Systems for African Cities,” SSATP Publication, 2022. [Online]. Available: https://www.ssatp.org/sites/ssatp/files/publications/SSATP-Innovation-Fare-Collection-2022.pdf

[2] ITDP Africa, “Digital Van Service for Addis Ababa,” 2022. [Online]. Available: https://africa.itdp.org/publication/digital-van-service-addis-ababa

[3] C. C. Chen, “Exploring the Factors of Using Mobile Ticketing Applications,” Transportation Research Part C: Emerging Technologies, vol. 144, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965856422001975

[4] A. T. Aruho, J. N. Mwesigwa, and R. Tumusiime, “Cashless Fare Collection Systems Acceptability in the Paratransit/Minibus Industry,” Transportation Research Record, 2025. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/03611981241234567

[5] J. Tupare, “Smart Public Transport Ticketing System Using QR Code,” SSRN Electronic Journal, 2024. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract\_id=4712324

[6] IJPREMS, “Smart Bus Ticketing System Using QR Code,” International Journal of Progressive Research in Engineering Management and Science, vol. 2, no. 1, 2022. [Online]. Available: https://ijprems.com/papers/Volume-2/Issue-1/IJPREMS0201055.pdf

[7] M. Nakkala, “Digital QR Tickets: Smart Solution for Urban Mobility,” ResearchGate, 2025. [Online]. Available: https://www.researchgate.net/publication/383049002\_Digital\_QR\_Tickets\_Smart\_Solution\_for\_Urban\_Mobility

[8] A. Gupta and S. Reddy, “A New Public Transport Payment Method Based on NFC and QR Code,” ResearchGate, 2018. [Online]. Available: https://www.researchgate.net/publication/322354879\_A\_New\_Public\_Transport\_Payment\_Method\_Based\_on\_NFC\_and\_QR\_Code

[9] VoxDev, “Mobile Money in Zambia: Opportunities, Challenges and Policy Debates,” 2023. [Online]. Available: https://voxdev.org/topic/mobile-money-zambia-opportunities-challenges

[10] UNDP, “The Role of Digital Mobile Money in Catalyzing Financial Inclusion,” United Nations Development Programme Report, 2024. [Online]. Available: https://www.undp.org/publications/digital-mobile-money-inclusion

[11] E. Bartin, S. Cetin, and H. Alpkocak, “Evaluation Framework for Mobile Ticketing Applications in Public Transit,” IET Intelligent Transport Systems, vol. 12, no. 5, 2018. [Online]. Available: https://digital-library.theiet.org/doi/full/10.1049/iet-its.2018.5248

[12] M. Koloseni and H. Mandari, “Towards Sustainable Adoption: Investigating QR Codes Mobile Payment Continuance in Tanzania,” Journal of Social and Political Sciences, 2025. [Online]. Available: https://www.asianinstituteofresearch.org/id/JSParchives/towards-sustainable-adoption%3A-investigating-qr-codes-mobile-payment-continuance-in-tanzania

[13] P. Francis, A. Muiruri, and K. Mbuthia, “Potential of Electronic Ticketing Machine Data in Public Transport Planning,” in Smart and Sustainable Transport, Springer, 2023. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-19-3505-3\_22

[14] S. Shimomba, M. Chanda, and B. Phiri, “Design and Development of an Integrated Online Bus Ticketing System,” Multidisciplinary Research Journal, vol. 5, no. 1, 2025. [Online]. Available: https://www.multiresearchjournal.com/arclist/list-2025.5.1/id-3796

# 8. APPENDICES

Send email test using postman api call to notification-service.

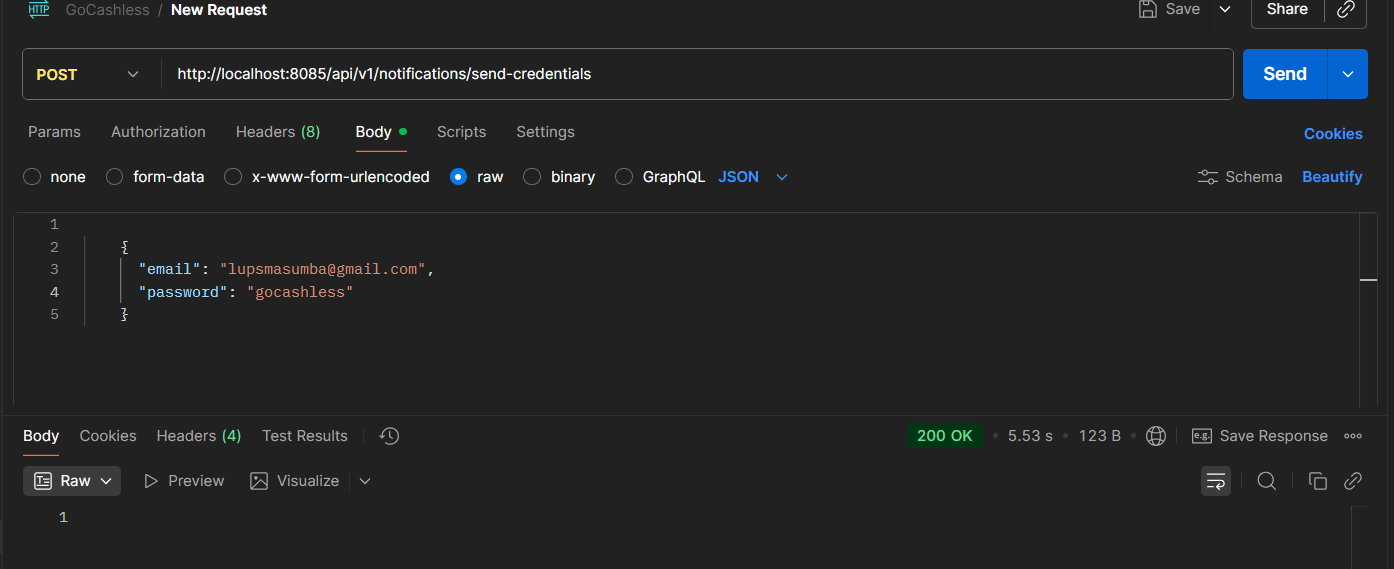


Figure 8.1: Send email test using postman api call to notification-service

Code snippet of EmailService.java

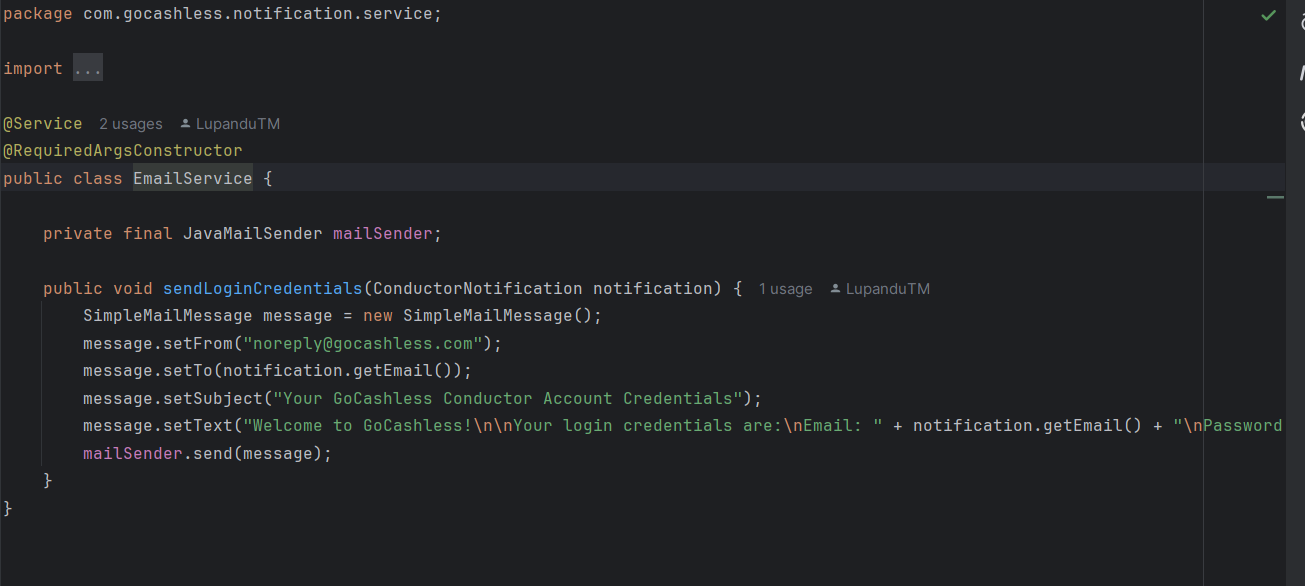


Figure 8.2: Code snippet of EmailService.java

Success, email received.

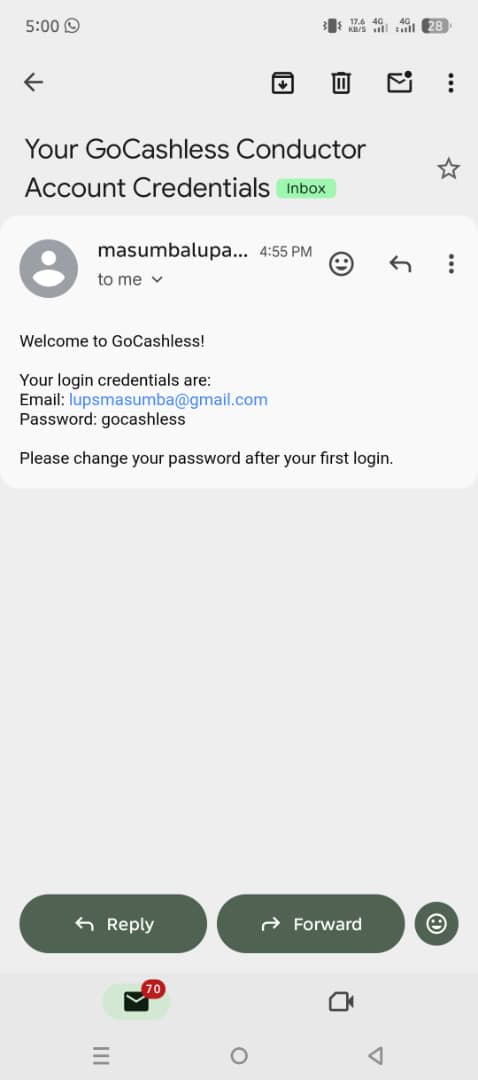


Figure 8.3: Successful conductor account registration email received

Conductor app route, and bus stops selection screen

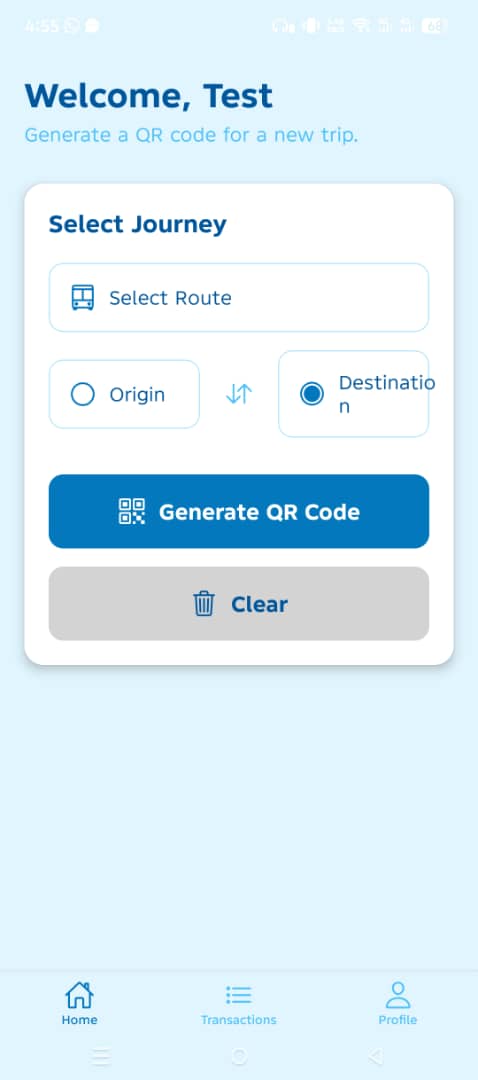


Figure 8.4: Conductor app route, and bus stops selection screen

Successful qr-code generation in the conductor app

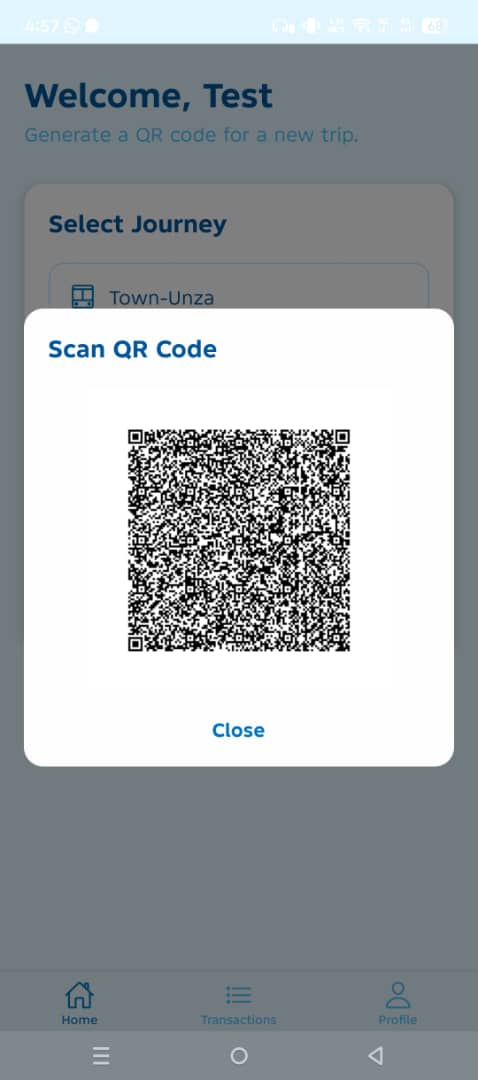


Figure 8.5: Successful qr-code generation in the conductor app

Bus Stops management page on admin dashboard.

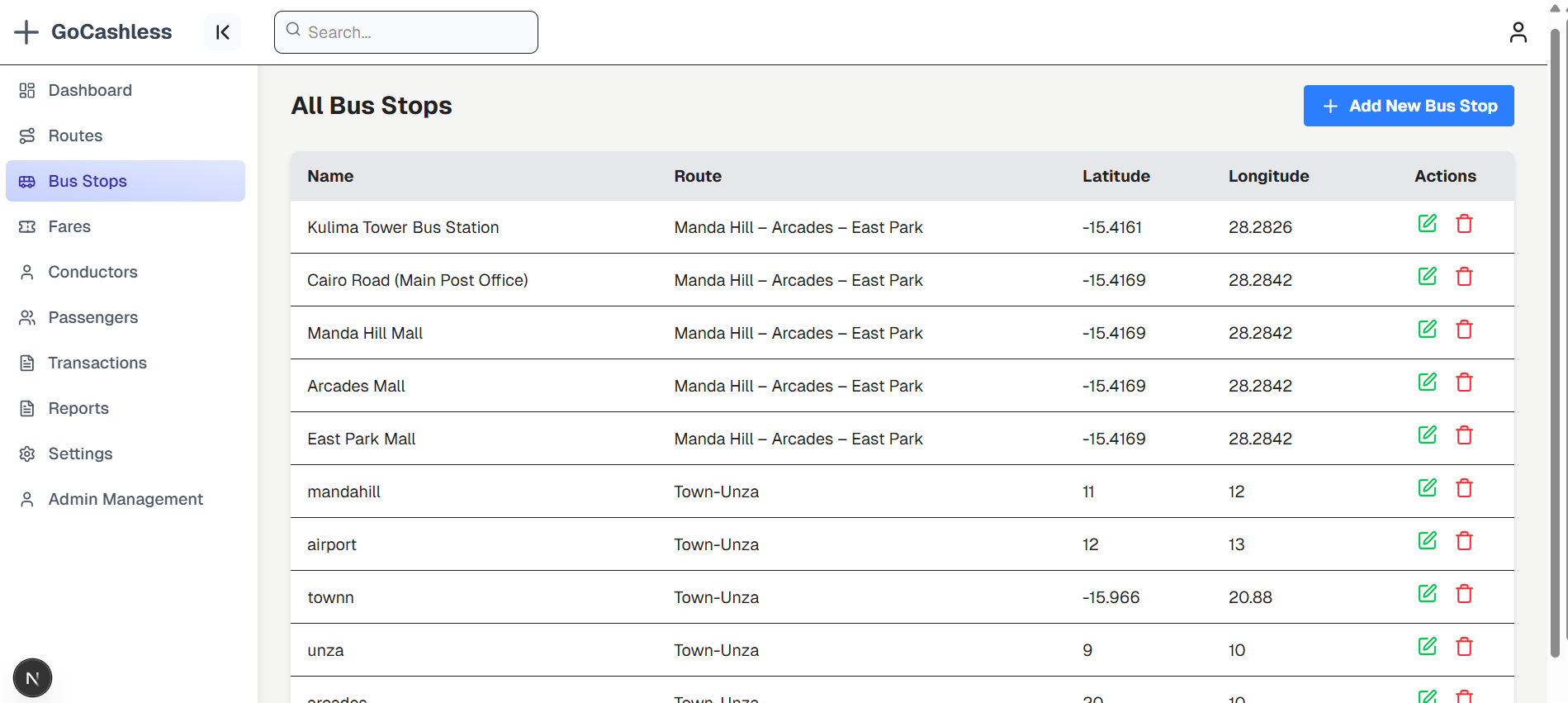


Figure 8.6: Bus Stops page on admin dashboard

Web application dashboard page

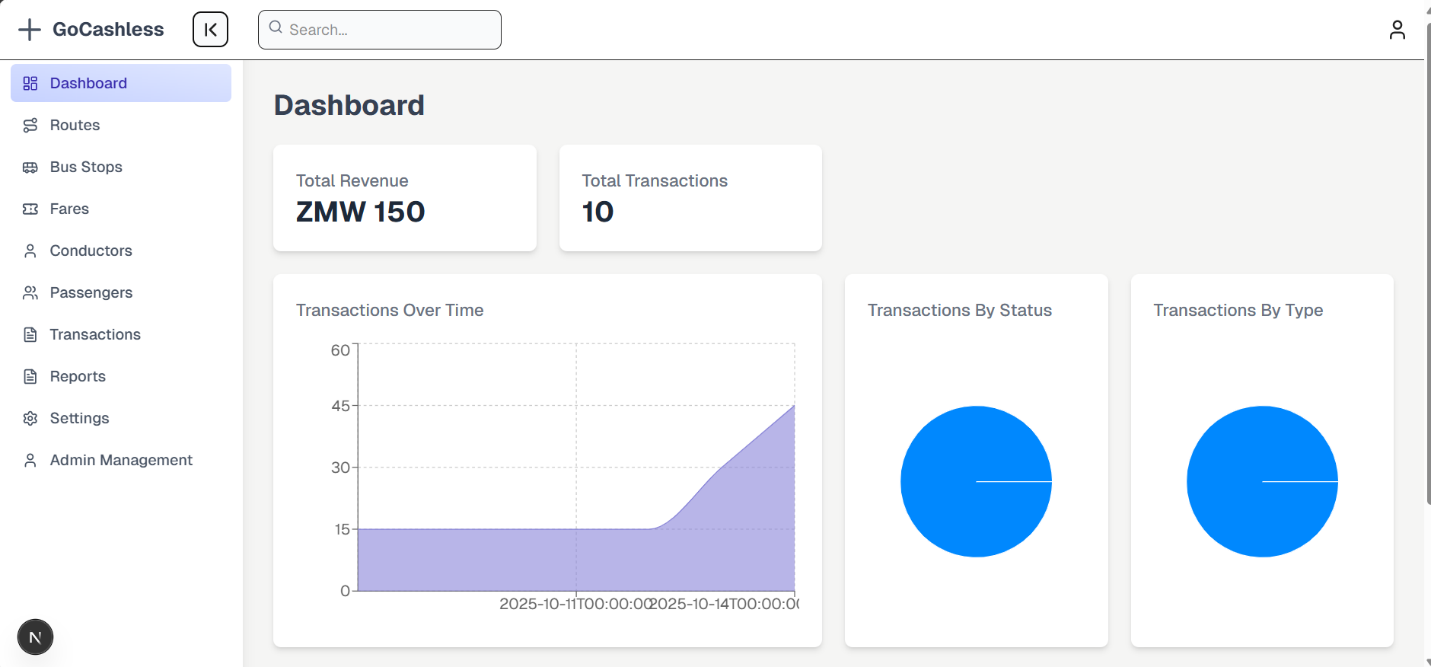


Figure 8.7: Web application dashboard page

Web application conductor management page.

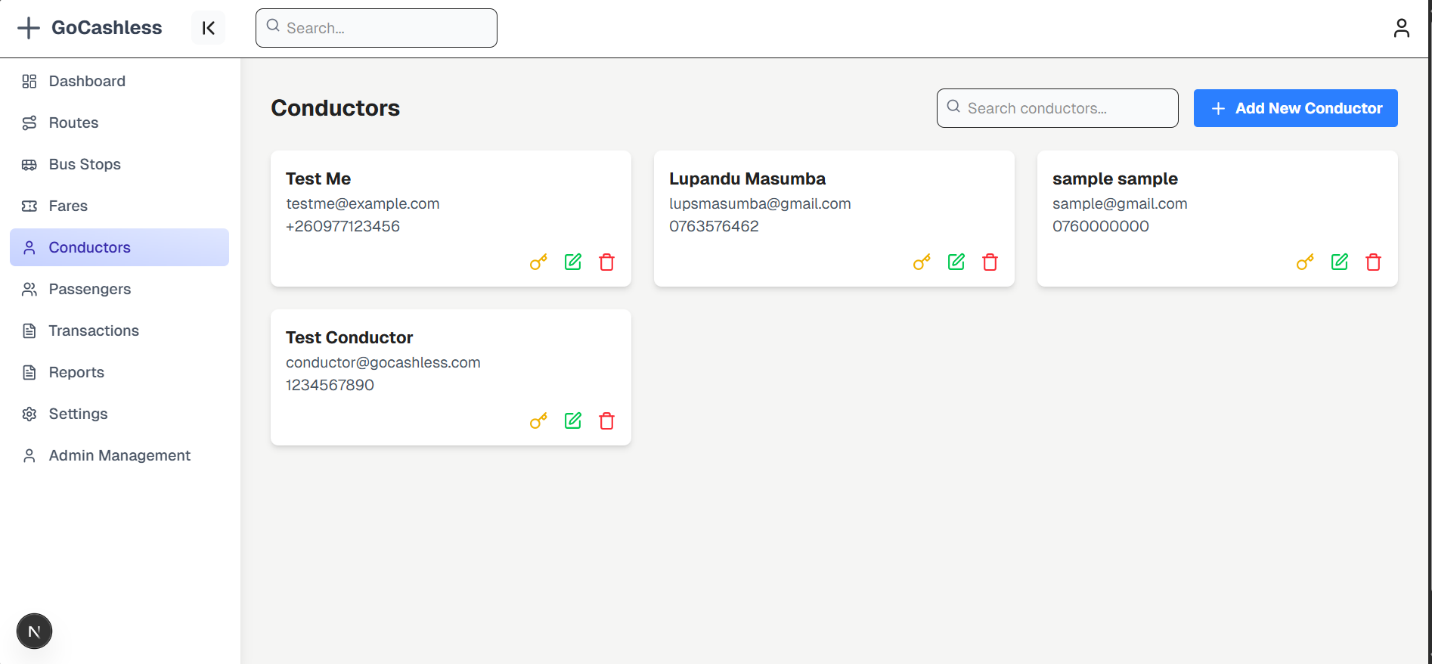


Figure 8.8: Web application conductor management page.

Web application Admin Management page.

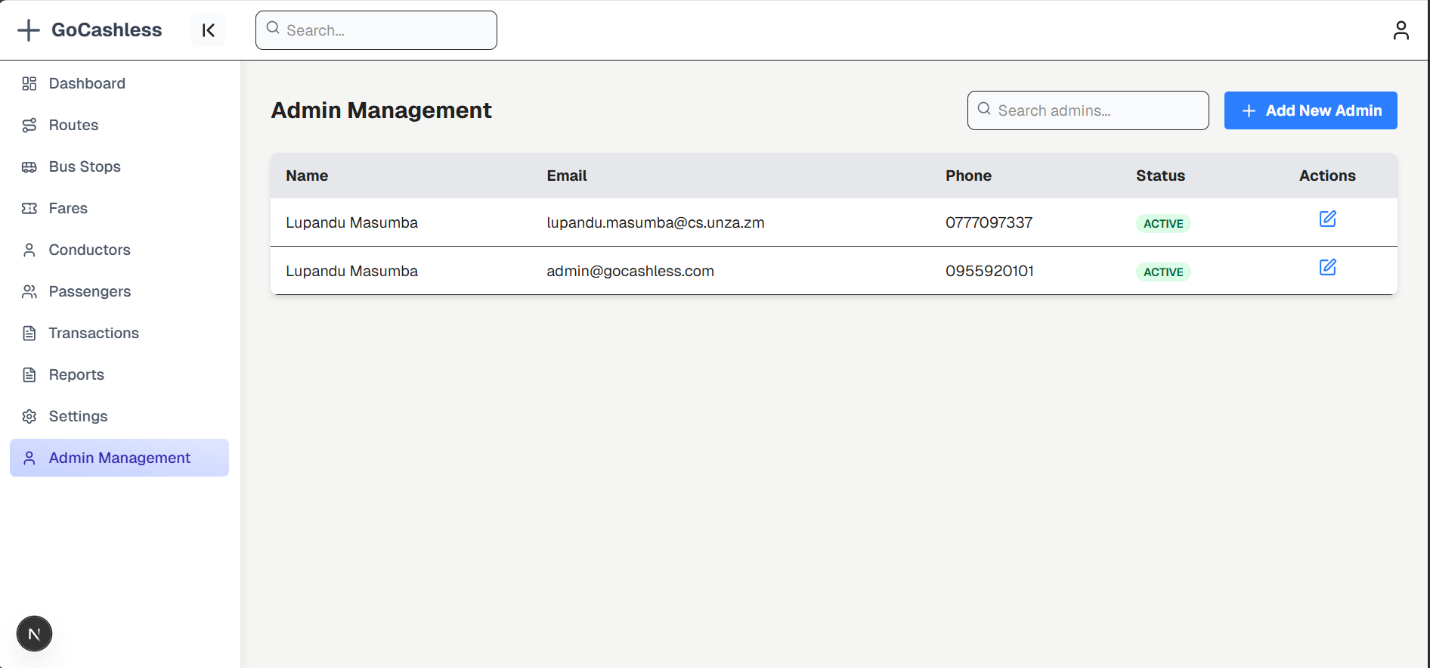


Figure 8.9: Web application Admin Management page.

Web application Fare Management page.

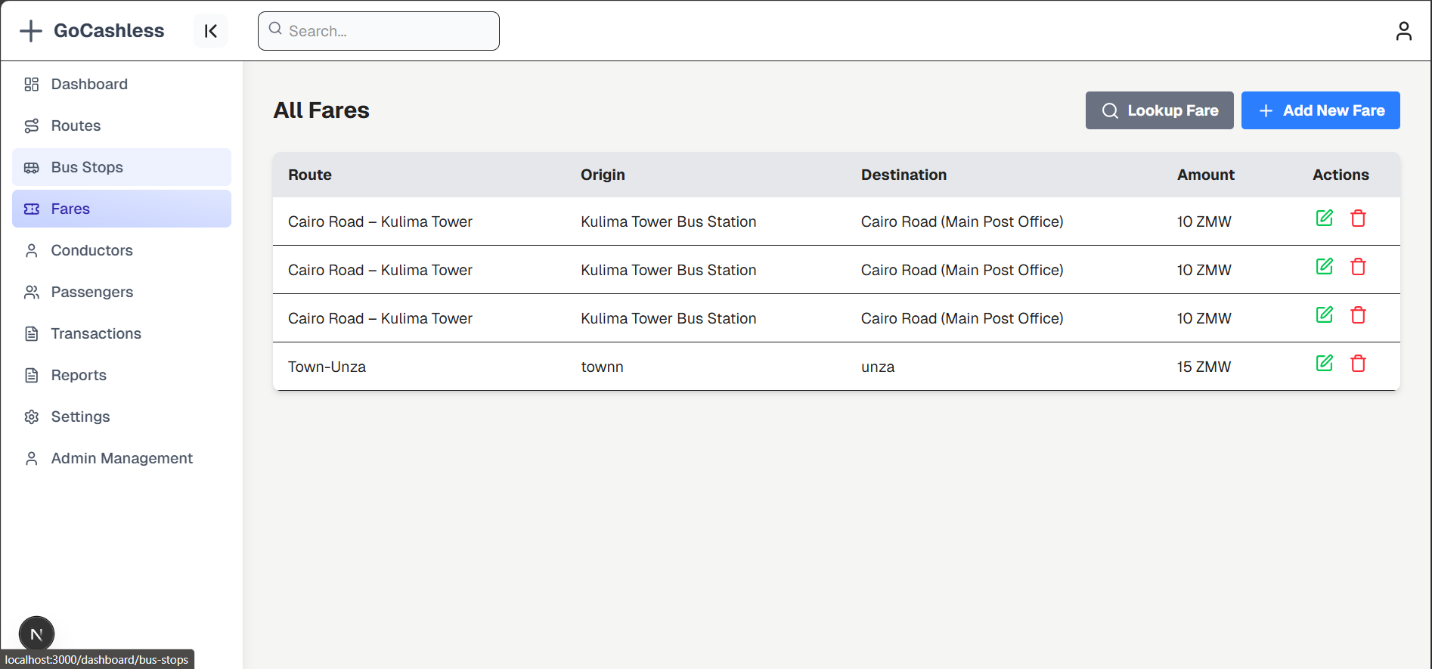


Figure 8.10: Web application Fare Management page.

Web application Route Management page.

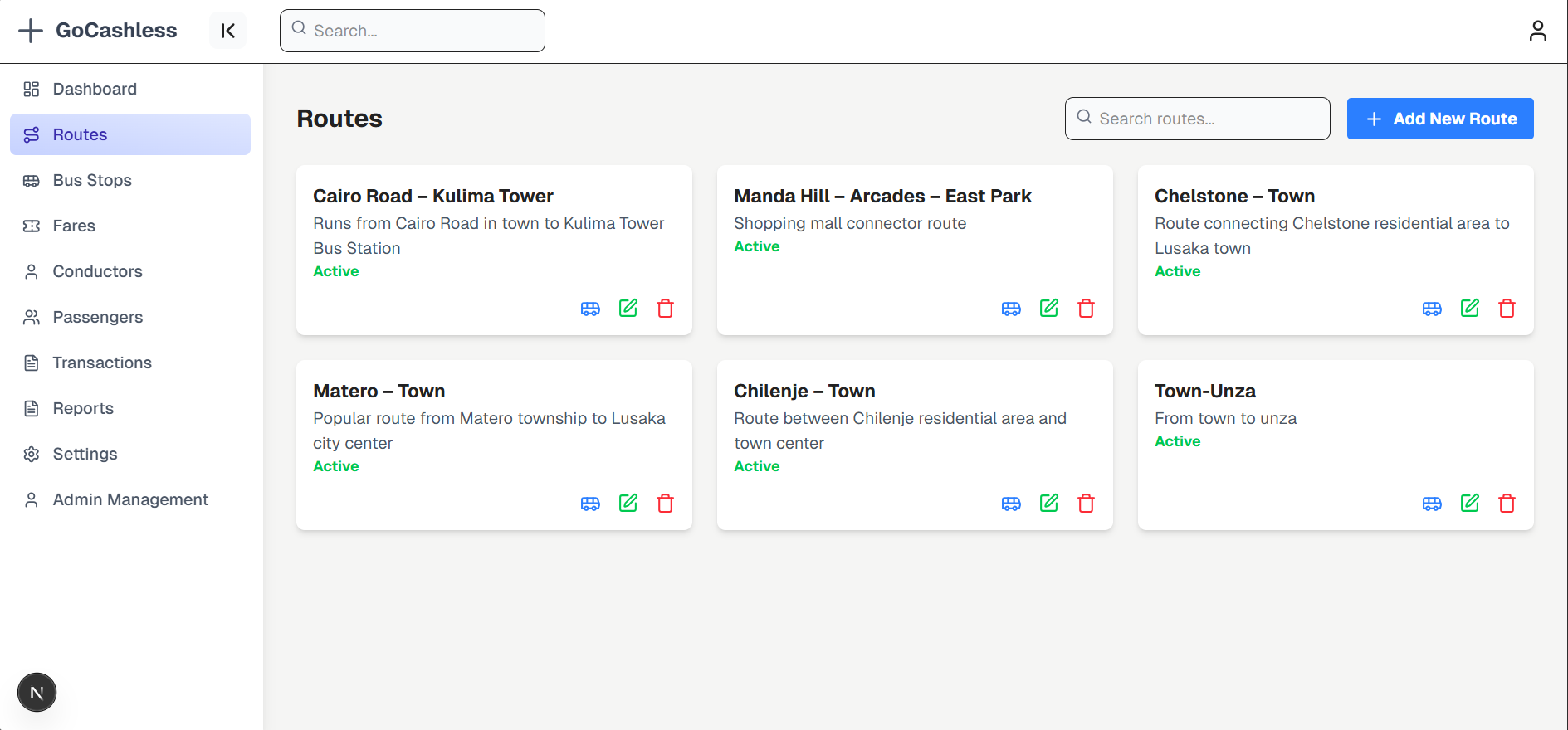


Figure 8.11: Web application Route Management page.