BusKoi
Bus Schedule Management System

By
Kamruzzaman Hridoy
213-15-4565

FINAL YEAR DESIGN PROJECT REPORT

This Report Presented in Partial Fulfillment of the Requirements for the **Degree of Bachelor of Science in Computer Science and Engineering**

Supervised by

Zakia Sultana
Lecturer
(SeniorScale)
Department of Computer Science and EngineeringDaffodil International University

Co-Supervised by

Dr. Md. Zahid Hasan
Associate Professor
Department of Computer Science and EngineeringDaffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

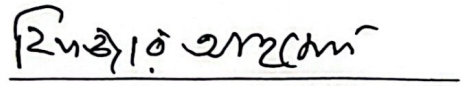
September 17, 2025

APPROVAL

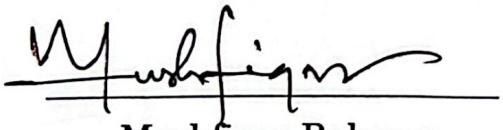
This Project titled "BusKoi Bus Schedule Management System," submitted by Kamruzzaman Hriody to the Department of Computer Science and Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 17 September, 2025.

BOARD OF EXAMINERS

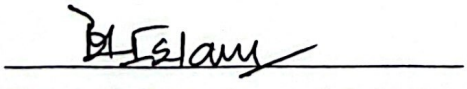
Dr. Arif Mahmud — **Chairman**
Associate Professor & Associate Head
Designation, Department of
CSE, FSIT Daffodil
International University



Dr. Fizar Ahmed — **Internal Examiner**
Associate Professor
Designation, Department of
CSE, FSIT Daffodil
International University



Mushfiqur Rahman — **Internal Examiner**
Assistant Professor
Designation, Department of
CSE, FSIT Daffodil
International University



Dr. Md. Manowarul Islam — **External Examiner**
Professor
Designation, Department of
CSE, FSIT Daffodil
International University

DECLARATION

We hereby declare that this project has been done by us under the supervision of **Zakia Sultana lecturer (senior scale)** Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this projectnor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

Zakia 17.09.25

Zakia Sultana
Lecturer (senior scale)
Department of Computer Science and EngineeringDaffodil International University

Co-Supervised by:

Md. Zahid Hasan
Associate Professor
Department of Computer Science and EngineeringDaffodil International University

Submitted by:

Kamruzzaman Hridoy 17.09.25

Kamruzzaman Hridoy
Student ID: 213-15-4565
Department of Computer Science and EngineeringDaffodil International University

ACKNOWLEDGEMENTS

This work would not have been possible without the support and contributions of many individuals over the past two semesters. We are deeply grateful to everyone who has assisted us in one way or another.

First, we express our heartfelt thanks and gratefulness to the almighty for His divine blessing making it possible for us to complete the **Final Year Design Project (FYDP)** successfully.

We are grateful and wish our profound indebtedness to **Zakia Sultana**, **Lecturer (senior scale)**, Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh Deep knowledge and keen interest of our supervisor in the field of **Artificial intelligence, Machine learning, deep learning, data mining** to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts, and correcting themat all stages have made it possible to complete this project.

We would like to express our heartfelt gratitude to the Head of the Department of Computer Science and Engineering, for his kind help in finishing our project and also to other faculty members and the staff of the Department of Computer Science and Engineering, Daffodil International University.

We would like to thank our entire course-mates at Daffodil International University, who took part in this discussion while completing the coursework.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

Transportation is a critical component for the smooth functioning of private organizations, schools, colleges, and universities, where thousands of individuals rely on shuttle buses or institutional vehicles for daily commuting. However, the traditional method of managing buses, drivers, and routes is still largely manual, relying on paper-based schedules, static notices, or simple spreadsheets. Such outdated practices often result in confusion, frequent scheduling conflicts, inefficient resource allocation, and miscommunication between administrators, drivers, and users. These issues collectively reduce punctuality, increase delays, and negatively impact the overall effectiveness of institutional transport systems.

To address these challenges, this project introduces **BusKoi**, a comprehensive web-based bus schedule management system that digitizes and streamlines the entire transportation process. Unlike manual systems, BusKoi enables **real-time schedule updates**, ensuring that any changes made by administrators are instantly visible to both drivers and users. The system is designed with **role-based dashboards**: administrators can create and manage buses, routes, drivers, and schedules; drivers can access their assignments, update live location links, and send alerts when delays occur; and students or staff can view schedules, receive notifications, and request adjustments.

The backend of the system has been developed using **Python Flask**, chosen for its lightweight yet flexible architecture, and integrated with **MySQL** through **SQLAlchemy ORM** for efficient data management. **Flask-Socket.IO** powers the real-time communication module, ensuring that updates and notifications are delivered instantly to all stakeholders. On the frontend, **HTML, CSS, JavaScript, and Tailwind CSS** were used to build a responsive, modern, and user-friendly interface that adapts to desktops, tablets, and mobile devices.

Key features of BusKoi include bus and route management, driver assignments, CRUD (Create, Read, Update, Delete) operations for schedules, live tracking through driver-provided links, and automated notifications. These features not only improve efficiency and reduce administrative workload but also minimize student waiting times, reduce driver confusion, and enhance overall transport reliability.

By implementing BusKoi, institutions can benefit from a **centralized, scalable, and cost-effective solution** that overcomes the limitations of manual scheduling. The system also supports future enhancements such as GPS integration for automated tracking, mobile application development for easier access, and advanced analytics for resource optimization.

In conclusion, BusKoi contributes significantly to institutional transportation management by providing a **real-time, role-based, and user-friendly platform**. It enhances communication between administrators, drivers, and users, improves punctuality, reduces delays, and promotes sustainability by optimizing bus usage. This makes BusKoi not only a technological innovation but also a practical and

impactful solution for improving daily commuting experiences in educational and organizational contexts.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Transportation is one of the fundamental services that every large institution must provide in order to ensure smooth operations. In universities, schools, and large private organizations, shuttle buses are often the most common form of transportation for students, faculty, and staff. However, managing such a transportation system is rarely simple. Administrators are required to manage dozens of buses, drivers, and routes simultaneously. Without a proper digital system, even minor issues such as a driver being late, a bus breaking down, or a sudden route change can cause serious confusion.

In many universities in Bangladesh, as well as in other countries, bus schedules are still handled manually. Typically, administrators prepare paper-based schedules or maintain simple Excel spreadsheets. While this may seem manageable at first, such systems become increasingly ineffective as the institution grows in size. Students may be left waiting for buses without knowing if the bus is delayed or cancelled, drivers may not be properly informed of changes in their assignments, and administrators may struggle to make last-minute updates.

The impact of this inefficiency is not limited to wasted time. Missed buses can lead to late arrivals in classes or examinations, resulting in significant academic disruptions. For staff and faculty, delays can reduce productivity and create dissatisfaction. For institutions as a whole, poor transportation management reflects negatively on organizational reputation and efficiency.

BusKoi was developed as a solution to these challenges. It is a web-based bus schedule management system that aims to provide a **reliable, user-friendly, and real-time platform** for managing all aspects of institutional bus transportation. With BusKoi, administrators can create and update schedules instantly, drivers can access and update their assignments, and students can view live schedules and receive real-time notifications. By digitizing and centralizing the process, BusKoi reduces confusion, saves time, and ensures smoother transport service across the institution.

1.2 Motivation

The primary motivation for creating BusKoi comes from observing the daily difficulties faced by students, staff, and administrators in managing university transportation. During our experience, we noticed the following recurring problems:

1. **Frequent Miscommunication:** Students often missed buses because they were not aware of sudden schedule changes or delays. Information was usually passed verbally, through word of mouth, or through physical notice boards, which were neither efficient nor reliable.
2. **Manual Scheduling Burden:** Administrators struggled to assign buses and drivers efficiently. In the absence of a centralized digital system, they had to spend hours preparing schedules manually, and even then mistakes were common.
3. **Lack of Real-Time Updates:** Once a schedule was printed or shared, it could not be updated quickly. If a bus broke down or a driver was absent, there was

no quick way to inform hundreds of students and staff waiting for transportation.

4. **Driver Communication Issues:** Drivers often had no direct channel to receive updates. If they needed to inform administrators about delays or problems, it was usually through phone calls, which caused further delays.
5. **Resource Inefficiency:** Without clear data on routes, bus capacity, or delays, the resources of the institution were not being used effectively. Some buses remained underutilized while others were overcrowded.

These problems motivated us to create a centralized, digital platform that can solve these inefficiencies. The idea was simple yet powerful:

- Give administrators a tool to manage and update schedules instantly.
- Allow drivers to receive assignments and send updates through the same system.
- Provide students and staff with real-time access to accurate bus information.

With BusKoi, we sought to not only reduce miscommunication but also save valuable time for students and staff, improve operational efficiency for administrators, and create a more transparent and reliable transportation system overall.

1.3 Objectives

The BusKoi project has been designed with clear and specific objectives. These objectives were identified after analyzing the current challenges in university bus management and considering the needs of different stakeholders (administrators, drivers, and students).

- **Centralized Web System:** To create a single web-based platform where all bus schedules can be created, updated, and managed by the administration.
- **Role-Based Dashboards:** To provide separate, customized dashboards for different users — administrators, drivers, and students.
- **Real-Time Notifications:** To ensure that any changes or updates in schedules are communicated instantly to all stakeholders.
- **Live Tracking:** To enable real-time visibility of buses and schedules, ensuring students and staff can plan their time effectively.
- **Scalability:** To design the system in a way that it can easily integrate with future features such as GPS-based live bus tracking and mobile app support.

These objectives together reflect the core vision of BusKoi: a simple, efficient, and scalable transportation management system.

1.4 Methodology

To achieve the stated objectives, BusKoi was developed using modern web development technologies and frameworks that support scalability and real-time operations.

- **Backend Framework (Flask):** Flask, a lightweight Python web framework, was chosen due to its flexibility, simplicity, and ability to handle modular design. It integrates well with databases and supports extensions for real-time communication.
- **Database (MySQL + SQLAlchemy):** MySQL was selected as the database because of its reliability and performance in managing structured data such as users, drivers, routes, and schedules. SQLAlchemy ORM was used to simplify database interactions and ensure clean, maintainable code.
- **Real-Time Communication (Socket.IO):** To enable live updates and

notifications, Flask-Socket.IO was implemented. This ensures that whenever a schedule is updated, drivers and students receive the information instantly.

- **Forms and Security (Flask-WTF & WTForms):** These libraries were used to handle form submissions securely, with built-in validation and CSRF protection.
- **Frontend (Tailwind CSS, HTML, CSS, JavaScript):** The frontend was developed with Tailwind CSS for modern, responsive design. HTML and JavaScript ensured interactivity and usability.
- **Development Model (Agile):** The Agile methodology was adopted for development, dividing the project into multiple phases: setup, backend models, route handling, frontend integration, and testing. This iterative approach allowed continuous feedback and improvements.

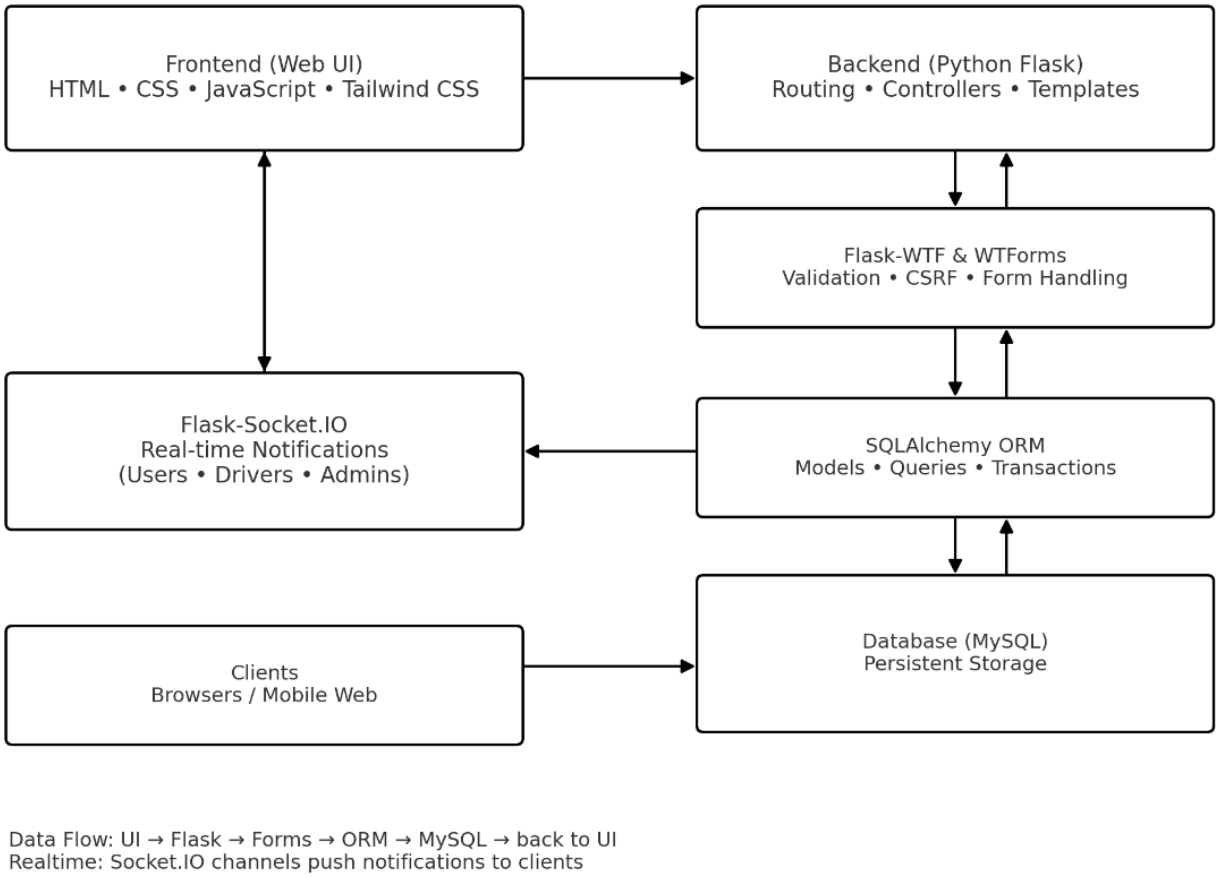


Figure 1.1: *methodology diagram*

1.5 Project Outcome

BusKoi is now a working, end-to-end bus schedule management system that does what it set out to do: make campus transport clear, fast, and easy for everyone.

- **Administrators**
  - Create, view, update, and delete buses, routes, drivers, and schedules from one place.

    - Assign drivers to trips in seconds and push instant notifications when plans change.
    - See the whole system at a glance, which cuts down on calls and manual follow-ups.
- **Drivers**
    - Register and sign in to a clean dashboard that shows today's route and timings.
    - Confirm assignments, share a live location/link, and report delays so everyone stays informed.
    - Spend less time coordinating by phone and more time staying on schedule.
- **Students/Users**
    - Check routes and departure times in real time from any device.
    - Get alerts for delays, cancellations, or replacement buses the moment they happen.
    - Request schedule adjustments when needed, improving day-to-day planning.

In short: BusKoi reduces confusion, speeds up communication, and helps students, drivers, and admins use their time better.

1.6 Organization of the Report

The report has been organized into six chapters:

- **Chapter 1:** Provides the introduction, motivation, objectives, methodology, outcomes, and organization of the project report.
- **Chapter 2:** Discusses the background of the problem, reviews related literature, compares similar applications, and identifies gaps that BusKoi addresses.
- **Chapter 3:** Explains the research methodology, system design, requirement specifications, data flow diagrams, user interface design, and task allocation.
- **Chapter 4:** Presents the implementation details, environment setup, testing methods, results, and analysis.
- **Chapter 5:** Highlights compliance with engineering standards, discusses the impact of the system on society and the environment, and addresses project management and financial considerations.
- **Chapter 6:** Concludes the report by summarizing the work, highlighting limitations, and suggesting future improvements such as GPS integration and mobile applications.

Chapter 2

Background

2.1 Introduction

Transportation is one of the essential support systems in modern educational institutions. Large universities and colleges often operate fleets of buses to ensure that students, faculty, and administrative staff can travel between campuses, residential areas, and other designated points. For thousands of individuals, buses are not merely a convenience; they are a daily necessity. Without reliable and efficient transport, many students would face difficulties attending classes on time, while faculty and staff would struggle to maintain punctuality and productivity.

Despite the importance of institutional transportation, managing such systems is far from simple. A transportation department must simultaneously handle dozens of buses, drivers, and routes. Each bus may have multiple schedules in a single day, covering different routes at different times. A sudden delay, a driver's absence, or unexpected traffic congestion can easily disrupt the entire system, leaving students stranded and administrators overwhelmed. These challenges highlight the critical need for a systematic and reliable transport management solution.

Globally, transportation management has been an active area of research. Many scholars have investigated scheduling conflicts, optimal route planning, fleet utilization, and real-time updates. Several solutions have been proposed and implemented, ranging from intelligent transport systems (ITS) used in urban cities to ride-sharing applications designed for individual commuters. However, these solutions are often either **too generalized** or **too complex** to be applied effectively in academic institutions. Systems designed for city-level transportation may include advanced modules like predictive analytics, large-scale GPS integration, and traffic management features, which are costly to implement in a university setting. On the other hand, basic digital tools such as static portals or spreadsheets are insufficient, as they cannot provide real-time updates or address the role-specific needs of different stakeholders.

The academic environment introduces unique challenges. Unlike urban transport, where the primary concern is passenger flow across a city, institutional transport systems are designed around fixed schedules aligned with academic timetables. Delays of even 10 to 15 minutes can disrupt classes, examinations, and administrative tasks. Moreover, communication between administrators, drivers, and students is often fragmented. Administrators may prepare paper-based schedules, drivers may receive verbal instructions, and students may rely on informal communication channels. This disjointed structure leads to inefficiency, confusion, and frustration.

The growing size of universities and the increasing number of enrolled students have amplified these problems. In many universities, buses operate at full capacity, and even a small scheduling error can create overcrowding in some routes while leaving others underutilized. Additionally, without proper data collection and analysis, institutions cannot make informed decisions about resource allocation, such as whether to add more buses to a route, change departure times, or optimize stop locations.

These issues call for a **lightweight, role-based, and real-time transport management**

system that is specifically designed for academic contexts. Such a system must:

- Be simple enough for administrators, drivers, and students to use without requiring specialized training.
- Provide real-time communication so that schedule changes and updates are instantly visible to all stakeholders.
- Include role-based dashboards, ensuring that administrators can manage resources, drivers can receive assignments, and students can access schedules — all within the same platform.
- Be cost-effective, leveraging open-source tools and scalable architecture so that institutions with limited budgets can adopt it.

This is where **BusKoi** positions itself. Unlike generalized transport solutions or static portals, BusKoi is tailored for the needs of educational institutions. It combines digital schedule management, real-time notifications, and a simple, role-based design to ensure smoother coordination. By filling the gap between manual systems and overly complex commercial solutions, BusKoi offers a practical alternative that universities can implement without excessive cost or complexity.

2.2 Literature Review

Many researchers have looked at how to run buses better how to plan schedules, track vehicles in real time, and reduce delays.
Here, I've picked the studies that actually matter for **BusKoi** and explain what they did, how they did it, and what we can learn from them.
First you'll see a quick table of methods and key findings, then a short, plain-English discussion of why each study is relevant to our system.

Table 2.1: Summary of Literature Reviewed.

Author(s)	Year	Title	Methodology	Key Findings
Hasan & Hossen	2019	Development of an Android Based Real Time Bus Tracking System	Prototype Development	Introduced a mobile tracking app for university buses; proved feasibility of digital real-time systems.
Jain et al.	2019	Application Based Bus Tracking System	Prototype & Cloud Integration	Showed how mobile + cloud apps can provide ETA predictions for campus buses.
de la Cruz et al.	2021	Smart transportation system for public universities	Case Study & Prototype	Developed an Android app with live geolocation and route updates for a Peruvian university.
Gull et al.	2021	Smart School Bus Tracking: IoT-based Design	IoT System Architecture	Outlined IoT-driven GPS tracking for buses with safety notifications, applicable to universities.

Shabli et al.	2023	Campus Bus Tracking System using LoRa Technology	IoT Prototype & Testing	Demonstrated low-power LoRaWAN network for campus-wide real-time bus tracking.
Sobhana et al.	2023	Smart Campus Bus Tracking Alert System Using Real-Time GPS	Experimental Implementation	Built GPS-based system to send bus location alerts to students in real time.
Vincent et al.	2023	Live College Bus Tracking and Route Mapping Using IoT	IoT Implementation	Integrated GPS + cloud to track buses and map routes live for colleges.
Alamatsaz et al.	2024	Enhancing Electric Shuttle Bus Efficiency	Timetabling & Optimization	Optimized electric shuttle schedules to cut waiting times and reduce operational costs.
Jabin et al.	2022	Transportation Service and Student's Satisfaction: A Study on Dhaka University	Survey Analysis	Identified overcrowding, delays, and lack of inclusiveness as key student transport issues.
Rungskunroch et al.	2025	Simulation-Based Optimisation of University Shuttle Bus Systems	Simulation & Route Optimization	Used simulation to improve efficiency, reduce trips, and cut energy use in university shuttles.

Expanded Discussion:

- Hasan & Hossen (2019): Developed one of the first Android-based bus tracking prototypes for Bangladeshi universities. Their work demonstrated the practicality of digital real-time systems, laying the foundation for BusKoi's tracking features.
- Jain et al. (2019): Created a mobile and cloud-based bus tracking application with estimated time of arrival (ETA) predictions. This highlighted the potential of combining mobile technology and cloud services to improve reliability of institutional transport.
- de la Cruz et al. (2021): Conducted a case study at a Peruvian public university and built an Android app to show bus geolocation and route updates. Their findings prove the value of digital portals for improving student access to transport information.
- Gull et al. (2021): Proposed an IoT-based school bus tracking system architecture using GPS and cloud services. Although targeted at schools, the design principles apply directly to universities seeking safer and smarter tracking systems.
- Shabli et al. (2023): Implemented a LoRa-based IoT solution for campus bus tracking. Their results show that low-cost, low-power networks can provide reliable tracking across large university campuses.

- Sobhana et al. (2023): Built a real-time GPS-based alert system for campus buses, which instantly notifies students about bus locations. This strongly supports BusKoi's focus on notifications and live updates.
- Vincent et al. (2023): Designed an IoT system combining GPS with cloud servers for live bus tracking and route mapping. Their work confirmed that IoT is an effective approach to institutional bus monitoring.
- Alamatsaz et al. (2024): Focused on scheduling optimization for electric shuttle buses in Montreal. Their case study showed that optimization algorithms can reduce wait times and energy costs, aligning with BusKoi's goal of efficient scheduling.
- Jabin et al. (2022): Conducted surveys at Dhaka University to evaluate student satisfaction with existing bus services. They found issues like overcrowding, long waits, and lack of inclusivity, underscoring the demand for systems like BusKoi.
- Rungskunroch et al. (2025): Applied simulation modeling to redesign university shuttle routes, reducing trips and energy usage by over 30%. This validates the efficiency and sustainability benefits that BusKoi aims to achieve.

From these studies, it is clear that while many transport management systems exist, there is still no lightweight, cost-effective, real-time, and role-based system tailored for academic institutions. BusKoi aims to bridge this gap by integrating the strengths of previous research into a practical and scalable solution.

2.2.1 Similar Applications

In order to design BusKoi, it is important to first examine similar systems that already exist. By reviewing how other institutions and applications handle transport management, we can identify their strengths and limitations, and then highlight how BusKoi takes a more balanced approach. Three categories of related systems are discussed below:

2.2.2 Dhaka University Shuttle Management System

Dhaka University, like many other public institutions in Bangladesh, runs shuttle buses to help students travel to and from campus. However, the management of this system is entirely manual and non-digital. Schedules are often printed on paper and posted on notice boards, or communicated verbally among students and staff. While this approach is simple and requires no investment in technology, it suffers from significant drawbacks. If a bus is delayed or a route changes unexpectedly, there is no immediate way to notify students. This results in confusion, long waiting times, and frequent miscommunication. Furthermore, drivers do not receive digital instructions, and administrators cannot easily update or track operations. In short, while the Dhaka University Shuttle Management System demonstrates how manual approaches can function at a basic level, it highlights the urgent need for digital solutions that support real-time communication.

2.2.3 Ride-Sharing Applications (Uber, Pathao)

Popular ride-sharing apps such as Uber and Pathao are examples of highly digitalized transport systems that provide advanced features, especially real-time tracking. They allow individual users to book rides instantly, monitor the vehicle's live location, and make payments online. Although these apps are very

effective for personal travel, they are not designed to serve large groups of people such as university students or institutional staff. They also lack academic-specific features like role-based dashboards for administrators, drivers, and students. More importantly, ride-sharing services come at a high cost. If students were to use Uber or Pathao daily for commuting, the financial burden would be far greater than relying on institutional buses. These apps demonstrate the benefits of real-time technology but also show why they cannot be directly applied as a solution for universities.

2.2.4 University Bus Portals Abroad

In many international universities, bus services are supported by online portals. Students can log into a university website to view static bus schedules, routes, and sometimes even seat availability. While this is more advanced than manual systems, these portals usually lack flexibility. Schedules are updated only once or twice a semester, and any sudden changes are rarely reflected in real time. Notifications are not sent directly to students, and drivers are not actively integrated into the system. Essentially, these portals provide static information, not dynamic communication. As a result, while they may give students a general idea of the timetable, they fail to address the everyday challenges of delays, cancellations, or unexpected changes.

Analysis and Comparison

Each of these existing systems brings something valuable to the discussion. The Dhaka University shuttle system is simple but outdated, relying on manual communication. Ride-sharing apps are advanced in technology but expensive and not designed for institutional use. International university portals provide convenience but lack real-time updates and driver involvement.

BusKoi has been designed to combine the positive aspects of all these systems while eliminating their weaknesses. Like ride-sharing apps, it incorporates **real-time notifications** and digital tracking. Like university portals, it provides easy **digital access** to schedules. Unlike manual systems, it removes the need for paper notices and verbal communication. At the same time, it avoids the high costs of ride-sharing services and introduces academic-focused features such as **role-based dashboards, driver assignment management, and administrative control**.

In conclusion, the review of these applications shows that there is no existing system perfectly suited for academic environments. Each of the current approaches is either incomplete, costly, or too rigid. BusKoi's uniqueness lies in its ability to **adapt to the institutional context**, providing a reliable, cost-effective, and user-friendly platform that addresses the shortcomings of existing solutions.

2.3 Gap Analysis

Our review of existing transport tools shows a simple truth: none of them truly fit how a university operates. Some apps are built for individual riders and work fine for personal trips, while others only show static, once-in-a-while updates or still depend on manual steps. To make the differences easy to see, we compared features across four groups—traditional shuttle systems, ride-sharing apps, university bus portals, and the proposed BusKoi platform.

Table 2.2: Gap analysis

Features	Existing Shuttle Systems	Ride-Sharing Apps	University Portals	Proposed System (BusKoi)
Live schedule updates	No	Yes	No	Yes
Role-specific dashboards	No	No	Partial	Yes
Real-time notifications	No	Yes	No	Yes
Driver assignment management	No	Partial	No	Yes
Bus & route CRUD management	Partial	No	No	Yes
User-friendly web UI	No	Yes	Partial	Yes
Scalability (future GPS)	No	Partial	No	Yes
Cost-effectiveness	No	No	Partial	Yes

The comparison makes the gaps obvious. Traditional shuttle systems rely on paper notices, so there's no real-time information when plans change. Ride-sharing apps like Uber and Pathao are great at live tracking, but they're not designed for campuses—there's no role-based dashboard, no driver assignment by admins, and the cost isn't practical for daily student use. Many university portals do offer online access, but they're mostly static and don't support real-time updates or two-way communication between drivers and students.

In contrast, BusKoi combines all essential features in a single system. It provides real-time notifications like ride-sharing apps, role-based dashboards tailored for administrators, drivers, and students, and a user-friendly interface. Additionally, it supports CRUD (Create, Read, Update, Delete) functionality for buses and routes, which is vital for daily operations. BusKoi is also cost-effective, as it is built using open-source technologies and is scalable for future GPS integration. This ensures that institutions can implement the system without bearing the high costs of commercial solutions while still enjoying modern, real-time functionality.

2.4 Summary

The review of existing literature and the comparative analysis of current applications clearly show that transportation management remains a challenge in academic institutions. Although universities and colleges around the world recognize the importance of reliable transport services, the tools currently in use are either outdated, expensive, or not designed to serve the unique needs of an educational environment.

Manual shuttle systems are still very common in universities in Bangladesh and other developing countries. These systems rely on notice boards, verbal communication, or static printed schedules. While such approaches may appear simple, they are highly inefficient in practice. Any sudden delay, bus breakdown, or driver absence cannot be communicated quickly to students. As a result, students waste time waiting for buses, and administrators struggle to manage last-minute changes.

Ride-sharing applications such as Uber and Pathao represent the opposite end of the spectrum. They offer advanced features such as real-time tracking, dynamic routing, and

mobile accessibility. However, they are not suited for institutional use. First, these services are designed for individuals, not large groups. Second, their cost makes them impractical for daily commuting, especially for students. Finally, they lack academic-focused features such as role-based dashboards for administrators, drivers, and students. This means they cannot handle institutional requirements such as assigning specific drivers to specific buses or managing multiple routes under one centralized platform.

University portals abroad attempt to provide a middle ground, but they are also limited. Many international universities have websites where students can check bus timetables. While this provides some level of convenience, the schedules are typically static and updated only a few times during the semester. These systems do not support real-time updates or direct communication with drivers. As a result, they cannot respond effectively to sudden changes in transportation schedules.

When these three categories of existing solutions are compared, it becomes clear that **none of them fully meet the requirements of academic institutions**. Manual systems lack speed and reliability, ride-sharing apps are too costly and generalized, and static portals are too rigid and outdated.

BusKoi has been designed to fill these gaps by combining the **strengths of all three approaches while eliminating their weaknesses**. Like ride-sharing apps, it provides **real-time updates and notifications**. Like university portals, it makes information digitally available and easy to access. Unlike manual systems, it eliminates the need for outdated paper-based communication. At the same time, BusKoi introduces several **unique, academic-specific features**:

- Role-based dashboards that serve the distinct needs of administrators, drivers, and students.
- A centralized system for creating, updating, and managing bus schedules.
- Real-time notifications that instantly inform users of delays, route changes, or driver updates.
- CRUD (Create, Read, Update, Delete) functionality for buses, routes, and schedules.
- Scalability to add future features such as GPS-based live tracking and mobile applications.
- Cost-effectiveness, as it is developed entirely using open-source tools.

By integrating these features into one lightweight, user-friendly platform, BusKoi ensures smoother communication, better coordination, and improved reliability in university transportation systems. It allows administrators to manage resources more efficiently, gives drivers a structured way to receive and respond to assignments, and provides students with accurate, real-time bus information that helps them plan their day more effectively.

In conclusion, the literature review and application analysis highlight the shortcomings of existing systems and the necessity for a new approach. BusKoi stands out as a **practical, future-ready, and institution-friendly solution** that not only addresses current problems but also prepares universities for modern, technology-driven transportation management.

Chapter 3

Research Methodology

3.1 Methodology/Requirement Analysis & Design Specification

3.1.1 Overview

The methodology adopted for this project was not a one-time, linear process but rather a structured and systematic approach that combined requirement analysis, design planning, and incremental development. The entire development cycle was carefully crafted to ensure that the final system — BusKoi — would be reliable, user-friendly, and tailored to the needs of universities.

The process began with a requirement analysis phase, where the research team studied existing transport management systems in detail. This involved observing how universities currently manage their shuttle services, analyzing the limitations of manual scheduling, and reviewing academic papers and case studies that explored transport scheduling, real-time communication, and resource management. By conducting this thorough background study, we were able to identify the most pressing problems: frequent delays, lack of real-time communication, difficulties in assigning drivers, and the inefficiency of paper-based systems.

Once these issues were clearly identified, the next step was to define the requirements for an improved system. We divided these requirements into two categories: functional requirements, which described what the system should do (such as allowing admins to add or update routes and drivers, enabling students to view schedules, and sending notifications), and non-functional requirements, which described how the system should perform (such as security, scalability, and reliability). This distinction allowed us to create a clearer vision of the final product and ensured that all user groups — administrators, drivers, and students — were taken into account.

Following requirement analysis, the design planning stage focused on selecting the most appropriate technologies and frameworks. Multiple options were evaluated at this stage: Django, Node.js, and Laravel were considered for the backend but rejected due to their complexity or limitations in real-time features. Similarly, PostgreSQL and MongoDB were considered for the database but were set aside in favor of MySQL, which was better suited for structured academic data. Finally, Flask was chosen as the backend framework for its simplicity and flexibility, SQLAlchemy was selected as the ORM for easier database management, and Flask-Socket.IO was adopted to ensure real-time communication. These design decisions were critical to achieving a balance between simplicity, cost-effectiveness, and scalability.

The system architecture was then planned in a modular way, with clear separation of concerns: the frontend was built using HTML, CSS, JavaScript, and Tailwind CSS for a clean and responsive design; the backend handled business logic, security, and authentication; the database stored all structured information; and the real-time communication module ensured instant updates for users and drivers. This modular architecture was specifically chosen to make the system maintainable and expandable in the future, allowing easy integration of features such as GPS tracking or mobile app support.

Once the design was finalized, the development phase was carried out incrementally. Instead of building the entire system in one go, the work was broken into smaller modules — for example, setting up the environment and database first, then building models, followed by developing the admin dashboard, then the user and driver dashboards, and finally integrating real-time notifications. Each module was tested individually before moving on to the next stage. This incremental strategy reduced the risk of major failures, since issues could be identified and resolved early in the process.

The iterative approach played a crucial role in the success of the project. Feedback was gathered continuously after each development stage, and improvements were made before moving forward. This cycle of design, development, testing, and feedback helped refine the system step by step, ensuring that the final product was both functional and user-friendly.

In short, the methodology combined requirement analysis, systematic design decisions, modular architecture, incremental development, and iterative testing. This structured approach not only ensured that BusKoi met its technical objectives but also made sure it addressed the real-world challenges faced by universities in managing bus transportation.

3.1.2 Proposed Methodology/ System Design

The system design followed a modular approach, ensuring that different parts of the application — backend, frontend, database, and real-time features — could be developed and improved independently.

- Backend: Built using Flask (Python) for its simplicity and flexibility. It manages routing, business logic, and communication with the database.
- Database: MySQL was chosen for its ability to handle structured data like users, routes, and schedules. SQLAlchemy ORM simplified database interactions and improved maintainability.
- Real-time Features: Flask-Socket.IO was implemented to send instant notifications to drivers and users whenever a schedule was updated.
- Frontend: Developed using HTML, CSS, JavaScript, and Tailwind CSS to provide a modern and responsive user interface.
- Security and Forms: Flask-WTF and WTForms handled form submissions securely, ensuring validation and CSRF protection.

The overall design can be visualized as a layered structure:

- Presentation Layer (Frontend) – User-facing dashboards for admin, driver, and student.
- Application Layer (Backend) – Handles business logic, authentication, and schedule management.
- Data Layer (Database) – Stores all information such as bus details, drivers, routes, schedules, and notifications.
- Communication Layer (Realtime) – Connects the system with drivers and students instantly via notifications.

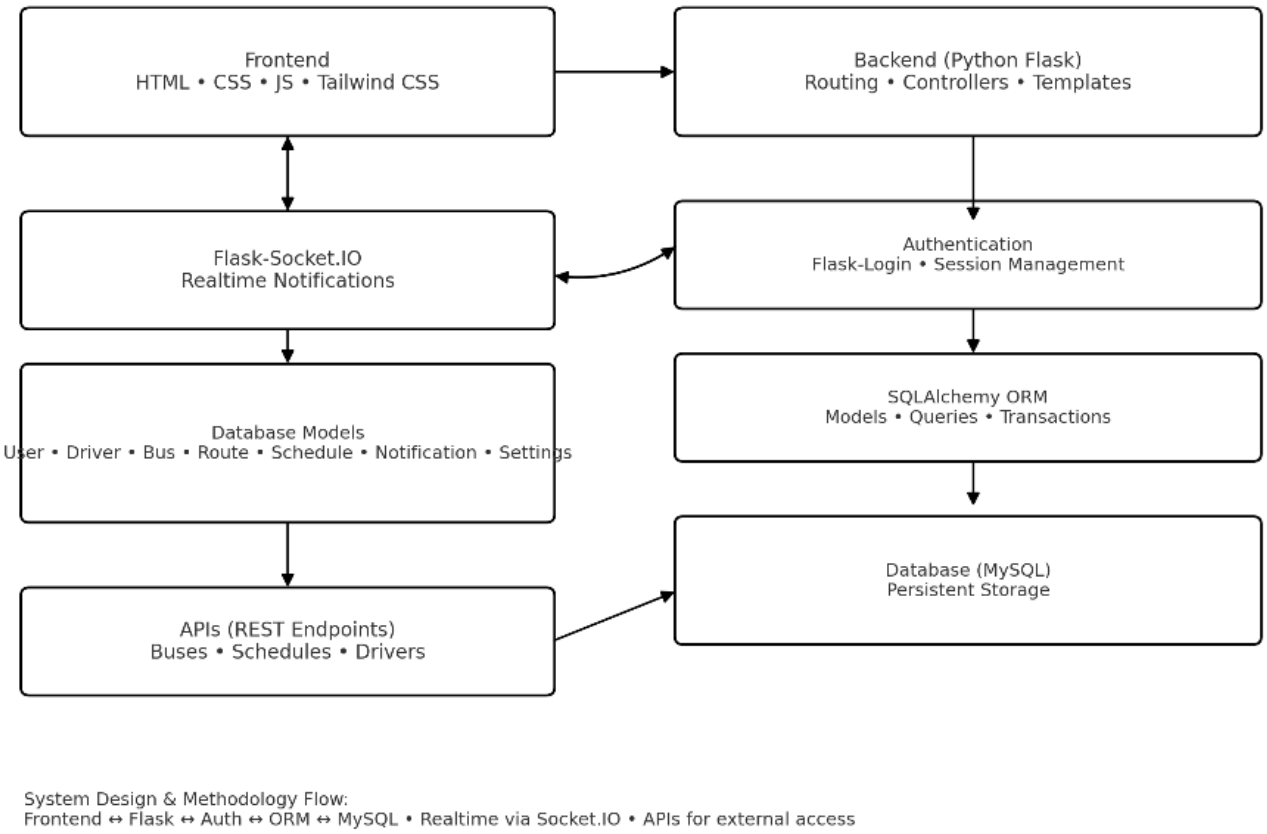


Figure 3.1: *System Design Diagram* (Frontend → Backend → Database, with Socket.IO for real-time communication)

3.1.3 Functional and Nonfunctional Requirements

When designing a software system, it is important to separate requirements into functional and non-functional categories. Functional requirements define *what the system should do* the tasks, services, and operations it must support for its users. Non-functional requirements, on the other hand, define *how the system should perform* focusing on quality, performance, and long-term sustainability. For BusKoi, both categories are equally important to ensure that the system is not only feature-rich but also secure, reliable, and scalable for future use.

Functional Requirements

The core functional requirements of BusKoi are centered around the three major stakeholders: administrators, drivers, and users (students/staff).

1) Administrator functions

- Manage everything in one place: Add, edit, or delete buses, routes, drivers, and schedules from the admin dashboard.
- Assign people to trips: Match specific drivers to buses and routes with simple role/assignment controls.
- Keep everyone informed: Send instant alerts about delays, route changes, or cancellations to drivers and students.
- See what's working: Generate quick reports on bus usage, driver assignments, and schedule performance to support decisions.

2) User (student/staff) functions

- Check live schedules: View up-to-date routes and times right from the dashboard.
- Ask for adjustments: Request short delays (e.g., when a class overruns) or share feedback with admins.
- Get real-time alerts: Receive notifications the moment a schedule changes, so you don't miss your ride.
- Track trips (current + future): For now, see driver-shared live location links; the system is ready to support full GPS tracking later.

3) Driver functions

- See today's plan: Log in to view assigned routes and timings—no more confusion or last-minute phone calls.
- Share live location: Update a live link so students can estimate arrivals accurately.
- Confirm trips: Mark schedules as "in progress" or "done," giving admins a real-time view of operations.
- Report issues fast: Send immediate alerts about breakdowns or incidents so admins

can reassign or adjust quickly.

These features keep admins, drivers, and students on the same page reducing confusion, improving punctuality, and making campus transport more reliable and transparent

Non-Functional Requirements

While the functional requirements describe what BusKoi can do, the non-functional requirements set the bar for how well it does it things like performance, scalability, and ease of use. Below is the security standard we follow, in plain terms.

1. **Security**
   - **Right access for the right people:** We use role-based access control so admins, drivers, and students only see and do what's relevant to them (least-privilege by default).
   - **Passwords are never stored as text:** They're hashed with a strong algorithm and a unique salt, so even if the database were leaked, raw passwords wouldn't be exposed.
   - **Locked-down sessions & forms:** Sessions use secure, short-lived, Http Only, Same Site cookies; all forms include **CSRF tokens**; and the server validates every request to block unauthorized access.
2. **Responsiveness**
   - The frontend must be responsive, meaning it should adapt seamlessly to different screen sizes, including desktops, laptops, tablets, and mobile phones.
   - Since students and drivers are likely to access the system from smartphones while on the move, ensuring mobile compatibility is a critical requirement.
   - Pages should load quickly, and navigation should remain simple and intuitive, regardless of device.
3. **Scalability**
   - The system must be scalable to handle growth in both the number of users and the complexity of features.
   - For example, while the initial version uses driver-updated links for live bus tracking, the architecture must support the integration of GPS modules in the future without requiring a complete redesign.
   - Scalability also ensures that the system can support thousands of concurrent users if adopted by larger institutions.
4. **Reliability**
   - The system must deliver accurate and consistent updates with minimal downtime.
   - Notifications should be sent instantly and should never be lost, even during peak usage.
   - Scheduled maintenance should be rare and carefully planned so that it does not disrupt daily transportation needs.
   - Database backup and recovery mechanisms should be in place to ensure that critical data (bus routes, schedules, user information) is never lost.

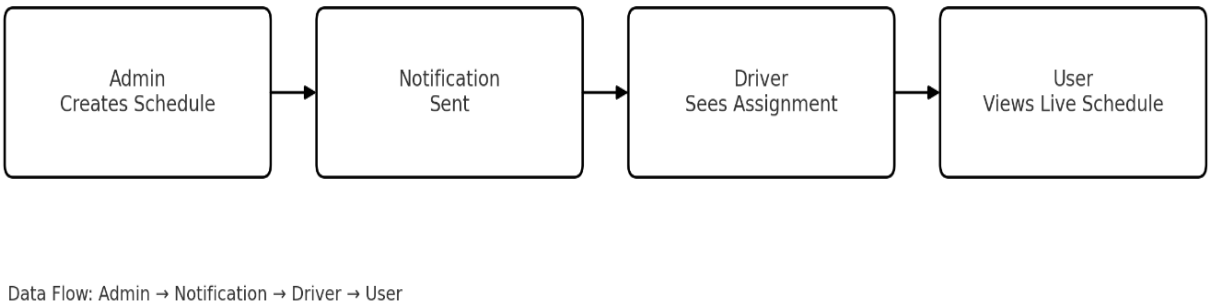


Figure 3.2: *Data flow diagram*

3.1.4 Data Flow Diagram

The data flow of BusKoi works as follows:

1. The admin creates or updates a bus schedule in the backend.
2. The database stores the updated information.
3. Socket.IO instantly pushes notifications to drivers and students.
4. Users log in to view updated schedules through the frontend.

This flow ensures real-time synchronization between the database and all stakeholders.

3.1.5 UI Design

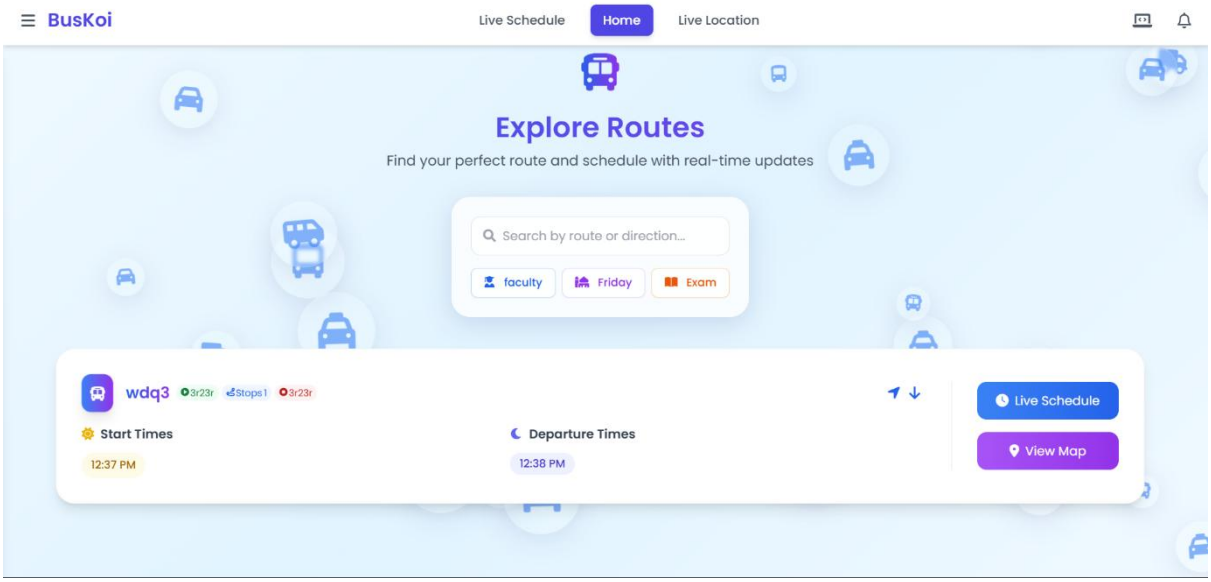


Figure 3.3: *home page*

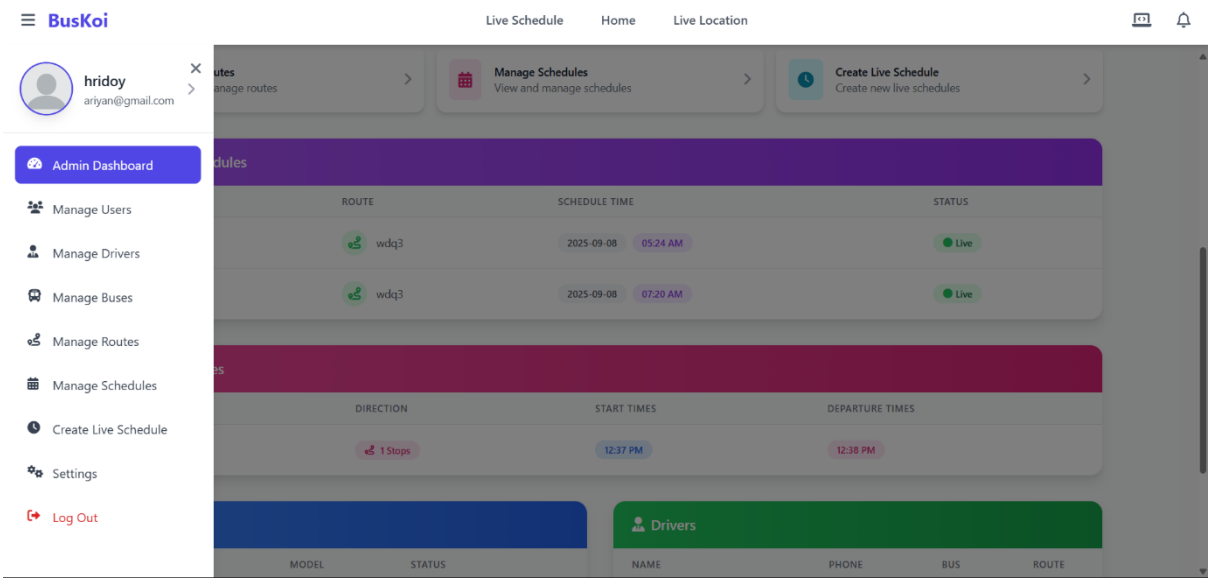


Figure 3.4: *admin dashboard*

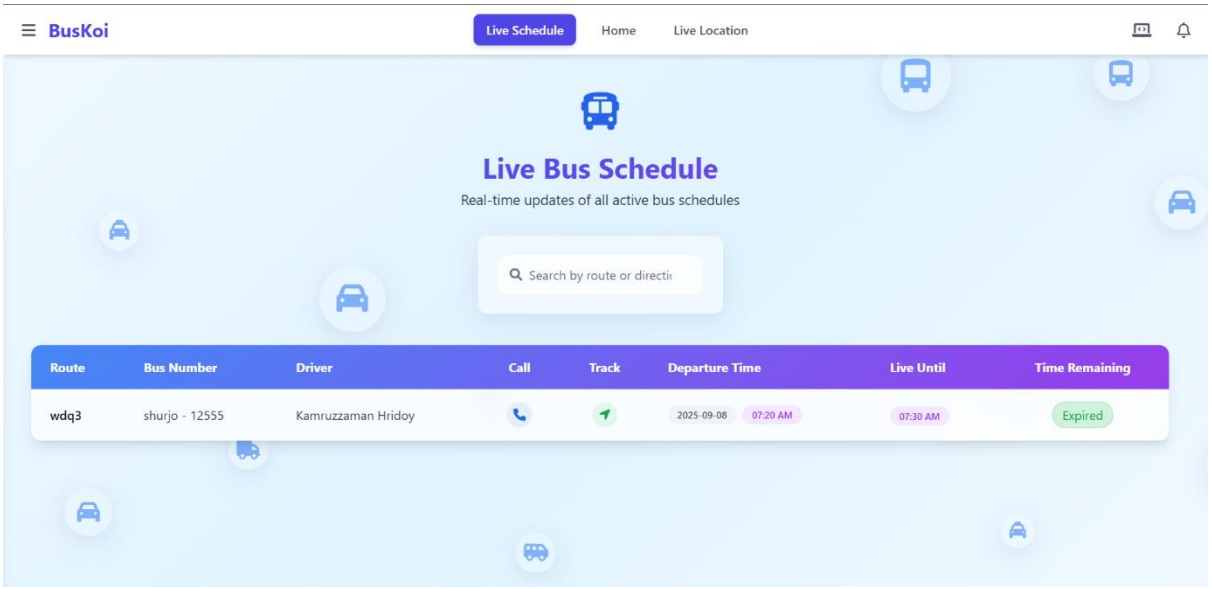


Figure 3.5: *live schedule page*

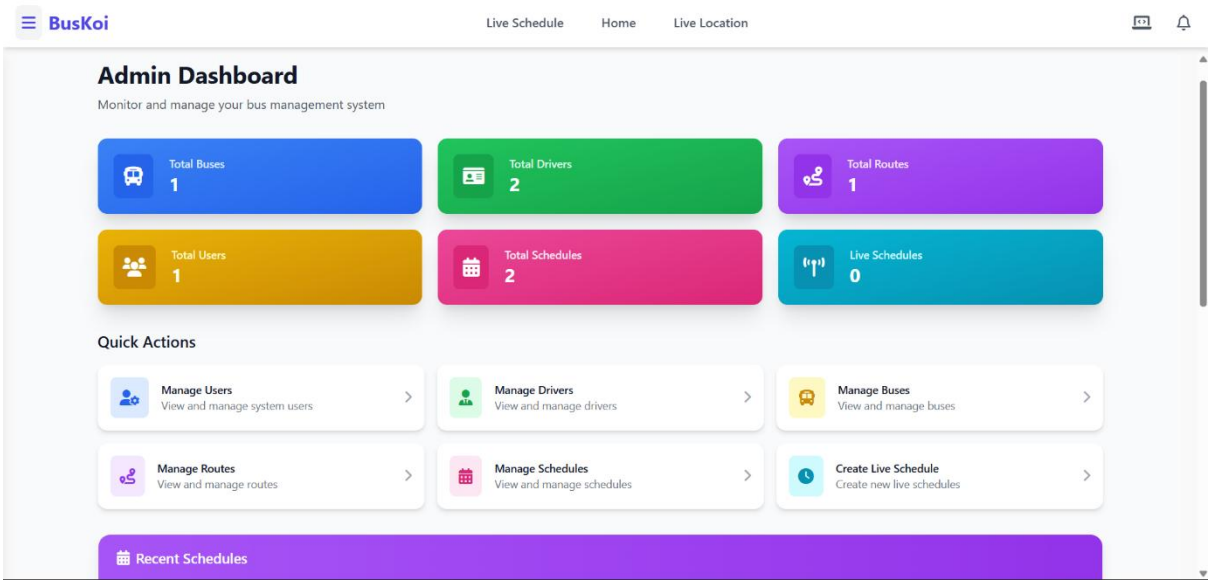


Figure 3.6: *admin Dashboard*

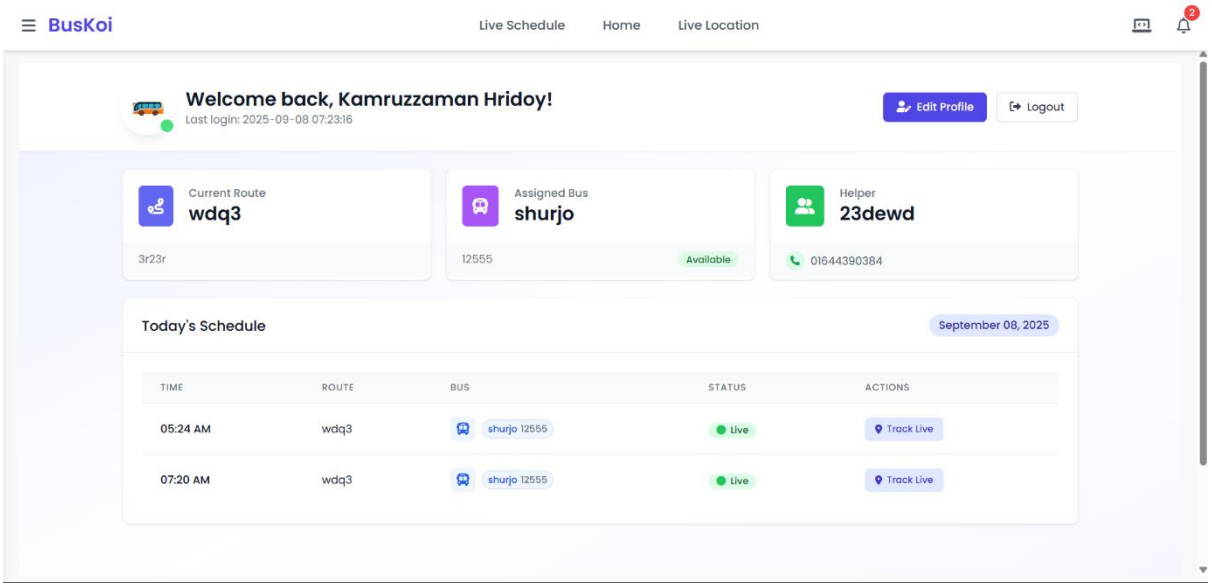


Figure 3.7: *Driver Dashboard*

3.2 Detailed Methodology and Design

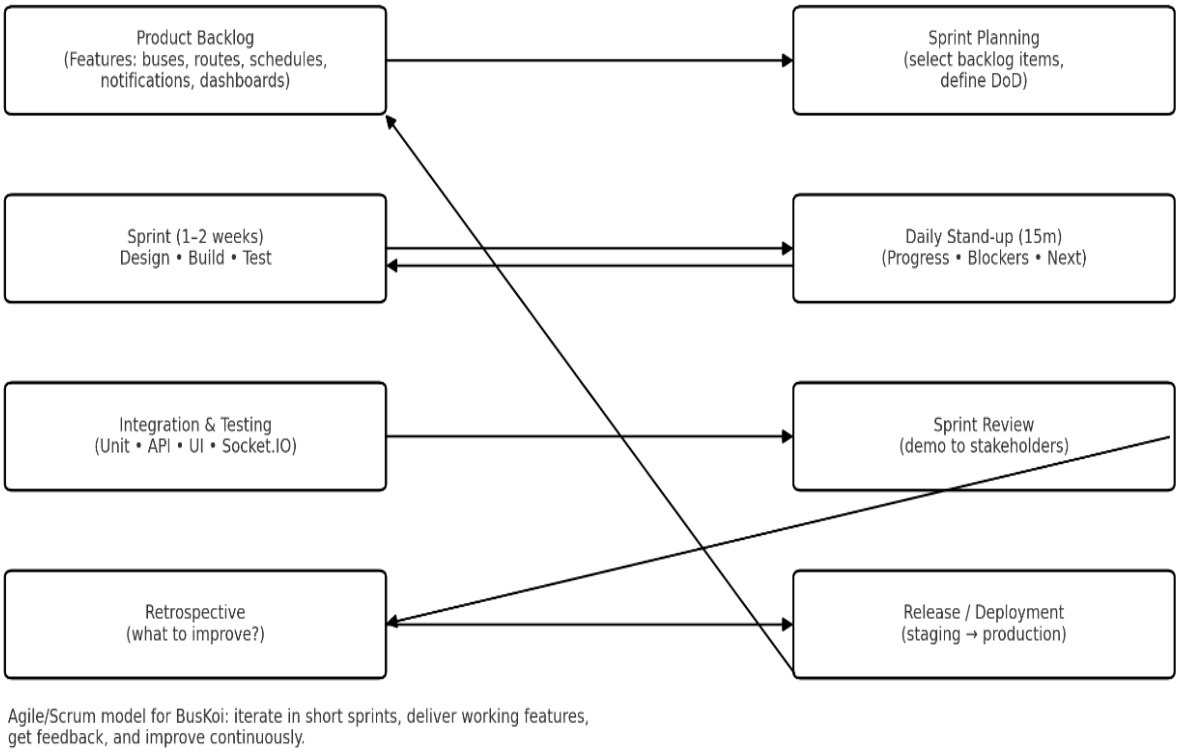


Figure 3.8: *Working process model agile*

While building BusKoi, several technology stacks were evaluated for different parts of the system, including the backend, real-time communication module, database, and frontend. Each option was carefully considered based on factors such as performance, ease of use, scalability, cost, and suitability for an academic bus management environment.

Backend Options

The backend forms the core of the system as it manages authentication, routing, business logic, and communication with the database. Different frameworks were considered:

- **Django (Python):** Django is a powerful and full-stack Python framework with built-in admin tools, ORM support, and strong security features. However, it was considered *too heavy* for this project. Django's monolithic structure, while excellent for large enterprise solutions, introduces unnecessary overhead for a relatively lightweight academic scheduling system. In addition, Django is not as flexible for integrating real-time features such as WebSocket communication compared to Flask.
- **Node.js with Express:** Node.js offers non-blocking I/O and is highly scalable. Express, being lightweight, could serve as a flexible backend option. However, setting up Express for real-time communication, ORM integration, and secure session management requires combining multiple external libraries, which increases complexity. The development team wanted a more unified ecosystem rather than manually stitching together multiple modules.

- **Laravel (PHP):** Laravel is a feature-rich PHP framework widely used in web development. While it offers built-in features like ORM and templating, it is not optimized for real-time communication out of the box. Implementing WebSockets or real-time notifications in Laravel requires additional layers of configuration and third-party packages. This lack of native real-time support made it less appealing.
- **Flask (Python):** Flask was ultimately chosen because it is **lightweight, flexible, and easy to customize**. It provides only the essential components, allowing developers to integrate additional features like Socket.IO for real-time updates. Flask also works seamlessly with SQLAlchemy ORM and is well-suited for modular development, making it ideal for an academic project.

Final Choice: Flask was selected for its **simplicity, extensibility, and ability to integrate smoothly with other components** of the system.

Real-time Communication Options

Real-time updates were a critical requirement, as students and drivers must receive schedule changes instantly. The following options were evaluated:

- **Firebase Realtime Database:** Firebase offers ready-made real-time synchronization services. However, it was rejected due to its recurring **subscription costs** and dependency on an external cloud service. Using Firebase would have made the system less cost-effective for universities with limited budgets.
- **Raw WebSockets:** WebSockets provide a direct way to handle real-time communication between the server and clients. However, implementing raw WebSockets would require extensive boilerplate code and custom management of connection states, error handling, and security. This would increase development complexity and maintenance burden.
- **Flask-Socket.IO:** This option provided the best balance between **simplicity and power**. Flask-Socket.IO integrates directly with Flask, making it possible to broadcast notifications to students and drivers without additional third-party services. It handles room management, event broadcasting, and error handling out of the box.

Final Choice: Flask-Socket.IO was selected because it is **simple, open-source, and natively compatible with Flask**, ensuring reliable and efficient real-time notifications.

Database Options

The database was a crucial consideration, as the system stores structured data such as buses, drivers, routes, schedules, and notifications. Several database technologies were analyzed:

- **PostgreSQL:** Known for its advanced features and support for complex queries, PostgreSQL was initially considered. However, for the needs of BusKoi — which primarily involve structured academic data and CRUD operations — PostgreSQL was unnecessarily **complex and resource-intensive**.
- **MongoDB (NoSQL):** MongoDB offers flexibility with schema-less design. However, BusKoi's data (routes, schedules, users) is **highly structured and relational**, making relational databases a better fit. Using MongoDB would have complicated the relationships between entities such as buses, drivers, and routes.
- **MySQL:** MySQL was chosen because it is a **reliable, relational, and widely supported database system**. It is open-source, well-documented, and integrates smoothly with SQLAlchemy ORM. It provides strong support for relational data, making it ideal for managing academic schedules.

Final Choice: MySQL with SQLAlchemy ORM was selected as the database solution due to its **reliability, simplicity, and suitability for relational academic data**.

Frontend Options

The frontend is the part of the system that interacts directly with users, so its design needed to be intuitive, responsive, and modern.

- **Bootstrap:** Bootstrap is a widely used CSS framework with prebuilt components. However, Bootstrap tends to produce heavy, uniform designs that look similar across applications. It also adds unnecessary bulk for projects that prioritize lightweight performance.
- **Tailwind CSS:** Tailwind CSS was chosen because of its **utility-first approach**, which allows highly customized and responsive design without writing excessive CSS code. Tailwind enables developers to create modern and lightweight interfaces quickly, while keeping the design flexible and maintainable.

Final Choice: Tailwind CSS was selected for its **flexibility, modern design capabilities, and responsive layouts**, ensuring the system looks professional and works well on all devices.

Final Technology Stack

After careful evaluation, the final chosen technologies were:

- **Backend:** Flask + SQLAlchemy + MySQL → Lightweight, easy to maintain, and efficient for relational data.
- **Real-time Communication:** Flask-Socket.IO → Provides robust real-time notifications without extra cost.
- **Frontend:** Tailwind CSS (with HTML, CSS, JS) → Ensures a clean, modern, and responsive interface.

The final design was selected because it **balances simplicity, cost-effectiveness, performance, and scalability**, making it ideal for a university bus management system.

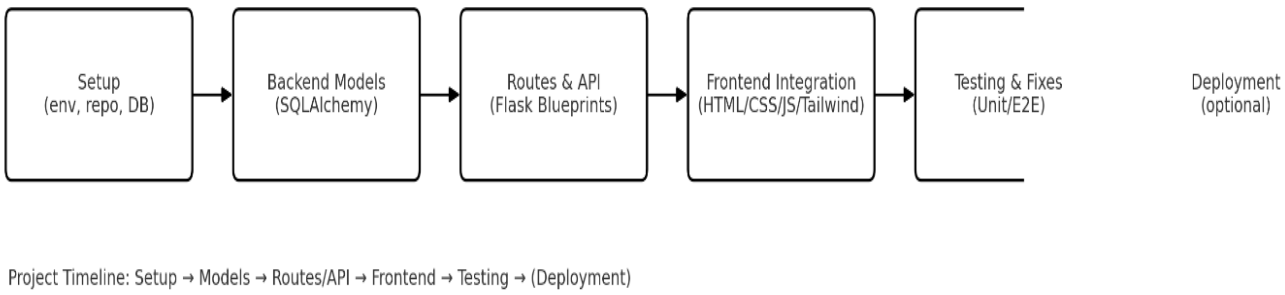


Figure 3.9: *Project timeline diagram*

3.3 Project Plan

The development of BusKoi was carefully divided into multiple phases in order to maintain an organized and systematic workflow. Instead of attempting to build the entire system in one step, the project followed a phased approach where each stage focused on a specific area of development. This method allowed the team to test, refine, and improve the system incrementally, ensuring that errors were identified early and that progress was steady throughout the project timeline.

The major phases of the project were as follows:

3.3.1 Setup Phase

Before writing any features, we focused on getting the basics right so the work would be smooth and repeatable. We set up a clean development environment, agreed on a few ground rules, and sketched the data we'd need so we wouldn't fight the tools later.

- **Environment & tools:** Installed Python, created a virtual environment, and added Flask, SQLAlchemy, and Flask-Migrate. MySQL was set up locally, and VS Code/Git were configured for day-to-day work.
- **Source control:** Initialized a Git repo, added a sensible. gitignore, and used a simple branching habit (main for stable, feature branches for changes). This kept history clean and made rollbacks easy.
- **Configuration & secrets:** Centralized settings in config.py and kept credentials in a .env file (not committed), so teammates could run the project with the same settings without exposing secrets.
- **Database blueprint**: Drafted the initial schema for core entities—buses, drivers, routes, schedules, notifications—and ran the first migration to create tables. This gave us a stable place to store data and evolve safely.
- **Smoke test:** Booted the Flask app, hit a health endpoint, and confirmed MySQL connections worked. That quick check caught setup issues early instead of mid-feature.

Why this matters: A solid setup phase saves hours later. With a consistent environment, clear config, and a ready database, we could onboard fast, avoid "works on my machine" bugs, and start building features with confidence.

3.3.2Backend Development Phase

After the setup, we built the "engine" of the app. Using SQLAlchemy, we modeled the core pieces—users (with roles), buses, routes, schedules, and notifications then exposed clean Flask routes for simple, reliable CRUD. Sign-in uses hashed passwords and role checks so admins manage data, drivers see assignments, and students view schedules. We added solid validation and consistent error handling, and each saved change emits an event, laying the groundwork for real-time updates with Socket.IO. This gave the frontend a stable, secure API to plug into.

3.3.3. Frontend Integration Phase

With the backend solid, we moved to the face of the app. Using HTML, CSS, JavaScript, and Tailwind, we built clean, responsive dashboards for admins, drivers, and students each showing the right info and actions for that role and wired them up to the APIs for a smooth, real-time experience. For example, the admin dashboard included management features for routes and drivers, while the driver dashboard allowed assignment viewing and live updates. The frontend integration ensured that all backend functionality was accessible in a simple and user-friendly way. This phase also focused on making the interface responsive, so it could be accessed from desktops, tablets, and smartphones.

3.3.4. Testing and Debugging Phase

Once both the backend and frontend were connected, the system underwent thorough testing. Different testing methods were used, including:

- Unit Testing: To verify that individual components (such as login, schedule creation, or notifications) worked as expected.
- Manual Testing: To check real-world usage scenarios by navigating through

the system as an admin, driver, or user.

- End-to-End Testing: To simulate complete workflows, such as creating a schedule as an admin and checking if drivers and students received real-time updates.

This phase also involved fixing bugs and optimizing the code for better performance. By dedicating time to testing, the team ensured that the system was reliable and ready for actual deployment.

3.3.5. Final Phase (Deployment, Demo, and Documentation)

In the last stretch, we polished what we'd built and got it ready to show and ship. We packaged the app with environment variables, ran final DB migrations, and recorded a clean demo covering schedule creation, driver assignment, and real-time notifications. We also wrote straightforward docs how to install, configure, and use each role's dashboard plus quick troubleshooting notes. Based on feedback, we fixed small UX snags, tightened error messages, and did a light performance check so it's smooth for a first deploy.

Why a Phased Approach?

Breaking the work into phases kept risk low and momentum high. Each step delivered something testable, so issues surfaced early instead of all at once. It also made hand-offs clear (setup → backend → frontend → testing → wrap-up), let us iterate with real feedback, and avoided rework by validating decisions as we went.

Conclusion

Following this plan, BusKoi moved steadily from setup to a working backend, a responsive frontend, thorough testing, and a deployment-ready demo. The result is a complete, reliable bus management system that meets the project goals and is ready for real use and future upgrades.

3.4 Task Allocation

To keep the work organized and predictable, we split the project into clear quarterly milestones—from late 2023 through early 2025. Working in time-boxed phases helped us keep momentum, spot issues early, and hit each milestone without piling up last-minute pressure. The table (Table 3.3) shows the quarter-by-quarter plan at a glance.

Table 3.3: task progress and time allocation

Tasks	Q4 2023	Q1 2024	Q2 2024	Q3 2024	Q4 2024	Q1 2025
System setup (environment, database, repo)	■					
Backend models & routes development		■	■			
Frontend integration (HTML, CSS, JS, Tailwind)			■	■		
Testing & bug fixing				■	■	
Prepare demo & documentation					■	■

System Setup (Q4 2023):

We laid the groundwork: installed Python/Flask, wired up MySQL, added SQLAlchemy/Flask-Migrate, and set up Git with a clean branching habit. We drafted the first database schema for buses, drivers, routes, schedules, and notifications, then ran an initial migration so everyone could start from the same stable base.

Backend Development (Q1–Q2 2024):

Next, we built the core: models, relationships, and secure endpoints. Admin CRUD for buses/routes/drivers/schedules, login with hashed passwords, and role checks (admin/driver/student) went in first. By the end of this phase, the API was stable and ready for real-time hooks.

Frontend Integration (Q2–Q3 2024):

With the API solid, we connected a responsive UI using HTML, CSS, JavaScript, and Tailwind. We delivered three focused dashboards admin, driver, and student—each showing the right actions and data for that role.

Testing and Debugging (Q3–Q4 2024):

We ran unit tests on models and endpoints, did manual walk-throughs across all roles, and executed end-to-end flows (create schedule → assign driver → user sees updates). Bugs were fixed as we found them, which kept quality high and surprises low.

Final Demo and Documentation (Q4 2024–Q1 2025):

Finally, we packaged a clean demo highlighting schedule management, driver assignment, and real-time notifications. We wrote straight-to-the-point docs install, configure, use plus quick troubleshooting. Feedback from dry runs led to small UX and performance tweaks before the final hand-off.

Result

Following this timeline gave us a clear path from setup to delivery, with no major overlaps or stalls.

3.5 Summary

Our approach combined careful requirement analysis, sensible tech choices, and a modular design. We picked **Flask** for a lightweight backend, **MySQL + SQLAlchemy** for reliable, structured data, **Flask-Socket.IO** for real-time updates, and **Tailwind CSS** for a fast, responsive UI. Working in phases let us validate decisions early, keep risks small, and deliver steady progress instead of a late crunch.

The outcome is a **real-time, role-based bus management system** that fixes the problems we saw at the start: miscommunication, stale information, and manual coordination. Admins can manage resources confidently, drivers get clear assignments and can share live updates, and students see accurate schedules with instant notifications. The system is practical today and ready to grow with GPS tracking, mobile apps, and richer analytics when the institution needs them.

Chapter 4

Implementation and Results

4.1 Environment Setup

Before building features, we invested time in a clean, dependable setup so the project wouldn't fight us later. The goal was simple: make development smooth, keep environments consistent, and be "deployment-ready" from day one.

Operating-systems.
We developed and tested on both **Windows** and **Linux**. That dual setup helped us catch OS-specific hiccups early and made it easier to deploy on whatever the institution prefers—an on-prem Linux server or a Windows VM.

Python-toolchain-&-dependencies.
We used **Python 3** with a dedicated virtual environment to isolate packages. Project libraries were pinned in requirements.txt, so anyone on the team could recreate the same environment with a single install command. Configuration lived in a .env file and a central config.py, keeping secrets (DB URL, session keys) out of the codebase and making local vs. production switches painless.

Backend-stack.
The server side runs on **Flask** because it's lightweight and easy to extend. **SQLAlchemy** gives us clear, maintainable models and relationships for core entities—buses, routes, drivers, schedules, users, and notifications. We used **Flask-Migrate** (Alembic under the hood) to version the schema, so changes roll forward (or back) safely without manual SQL. This kept the database in lockstep with the code throughout development.

Database.
We chose **MySQL** for its stability with relational data and familiar tooling. Tables use InnoDB for transactions and referential integrity. During setup, we created the initial schema and seeded baseline data so the app could boot with sensible defaults. Local development used a lightweight instance via **XAMPP**, which also gave us quick access to phpMyAdmin for sanity checks.

Frontend-foundation.
The interface is standard **HTML, CSS, and JavaScript**, styled with **Tailwind CSS**. Tailwind's utility classes kept pages responsive without a lot of custom CSS. We wired the UI to the API early, so we could test real flows (create schedule → assign driver → user sees update) while we built new screens.

Dev tools & workflow.

We used **Git** for version control with a straightforward flow: main for stable code and short-lived feature branches for active work. **VS Code** was the primary editor—useful extensions (Python, linting, SQL) and the built-in debugger sped

up everyday tasks. We added a basic .gitignore, a sample .env.example, and a quick "health check" route so teammates could verify their setup in seconds.

Why-this-matters.
A solid environment prevented "works on my machine" bugs, shortened onboarding, and let us focus on features instead of firefighting. With dependencies pinned, configs externalized, migrations in place, and tooling aligned, we had a stable base for both development and deployment.

4.2 Testing and Evaluation/Performance/ Comparative Analysis

Testing and evaluation were critical to ensure that BusKoi not only functioned correctly but also performed reliably under different conditions. Several layers of testing were conducted:

1. **Unit Testing:**
   - Each module (such as user authentication, schedule creation, and notification sending) was tested individually.
   - For example, unit tests confirmed that when an admin created a new schedule, it was properly stored in the database and could be retrieved by users and drivers.
2. **Manual Testing:**
   - After unit testing, manual testing was conducted by navigating through the system as different roles (admin, driver, and user).
   - Admin dashboards were tested for adding and editing buses, while drivers' dashboards were tested for updating live links.
   - Users' dashboards were tested for receiving notifications and viewing updated schedules.
3. **End-to-End Testing:**
   - Real-world scenarios were simulated, such as an admin updating a schedule, a driver receiving that update, and a user viewing the new schedule in real time.
   - These tests confirmed that data flowed correctly across the system, from backend to frontend with real-time notifications.

Performance-Evaluation:

The system was tested with multiple users simultaneously to check how it handled concurrent requests. Flask-Socket.IO ensured that notifications were delivered instantly, even when multiple drivers and users were connected. The system showed stable performance, with response times remaining within acceptable limits.

Comparative

Analysis:

When compared to manual scheduling and static portals, BusKoi proved significantly more efficient:

- Real-time updates reduced miscommunication.
- Automated notifications saved time for both students and administrators.
- Role-based dashboards made the system more organized than generalized portals.

4.3 Results and Discussion

After development and testing, BusKoi delivered the following results:

- **Admin Dashboard:**
    - The admin was able to perform full CRUD operations for buses, routes, drivers, and schedules.
    - Notifications could be sent to users and drivers instantly whenever a schedule was updated.
    - The system also maintained a clean log of driver assignments, making it easier for administrators to monitor operations.
- **Driver Dashboard:**
    - Drivers successfully registered and logged in.
    - They were able to view their assignments in real time and update live location links for users.
    - Drivers also had the ability to confirm completed trips, improving accountability.
- **User Dashboard:**
    - Students and staff could view bus schedules and routes clearly on their dashboard.
    - Real-time notifications alerted them immediately if a bus was delayed or rerouted.
    - Users were also able to make delay requests, providing flexibility in communication between students and administrators.

Discussion:

The results confirmed that the system met its original objectives. Unlike manual systems, BusKoi provided instant communication, better organization, and improved reliability. Compared to costly solutions like ride-sharing apps, it achieved similar real-time functionality but in a more affordable and institution-specific way.

The system was also shown to be scalable: while it currently uses driver-updated links for live location, its architecture supports future GPS integration and mobile app development.

4.4 Summary

This chapter demonstrated that BusKoi was successfully implemented and tested using the chosen technology stack of Flask, SQLAlchemy, MySQL, Tailwind CSS, and Flask-Socket.IO. The environment setup ensured smooth development, while rigorous testing confirmed that the system worked as expected across different roles.

The results showed that administrators could manage schedules effectively, drivers could update assignments and live locations, and users could view real-time schedules and notifications. Performance testing proved the system to be reliable under concurrent usage, and comparative analysis showed clear improvements over manual systems and static portals.

In conclusion, BusKoi achieved its goal of creating a **real-time, role-based, and scalable bus schedule management system**, providing a practical solution for universities and similar institutions.

Chapter 5

Engineering Standards and Design Challenges

5.1 Compliance with the Standards

When developing a software system for institutional use, it is important to follow recognized standards to ensure quality, maintainability, and long-term usability. BusKoi was developed in compliance with software, hardware, and communication standards. Alternative approaches were also considered, and the final selections were made based on pros, cons, and overall suitability.

5.1.1 Software Standards

- **Backend-Software-(Flask,-SQLAlchemy):**
  Flask and SQLAlchemy were chosen because they are **open-source, lightweight, and well-documented**. Flask provides flexibility without forcing a rigid project structure, while SQLAlchemy ORM ensures clean and maintainable database operations.
  - *Alternative:* Django was considered but rejected for being too heavy and opinionated, which could limit customization.
  - *Rationale:* Flask + SQLAlchemy offered the right balance of **simplicity, modularity, and reliability**, ideal for academic use cases.
- **Communication-(Socket.IO):**
  Socket.IO was used for **real-time communication**, enabling instant updates to users and drivers.
  - *Alternative:* Firebase Realtime Database was reviewed but discarded due to recurring subscription costs and reliance on external infrastructure.
  - *Rationale:* Socket.IO integrates directly with Flask and is free, making it both cost-effective and efficient.
- **Security-(Password-Hashing,-CSRF-Protection):**
  Passwords are hashed securely before storage, and forms are protected with CSRF tokens. These are considered standard practices for web security.
  - *Alternative:* Plaintext storage of credentials (sometimes seen in weak systems) was not acceptable due to risks of data leaks.
  - *Rationale:* Security was prioritized to **protect user data** and ensure ethical compliance.

5.1.2 Hardware Standards

BusKoi does not require any special or dedicated hardware. It runs smoothly on:

- Standard **institutional servers** or **cloud instances**.
- User devices such as **computers, tablets, and smartphones**.
- *Alternative:* Dedicated GPS servers or proprietary devices were considered but were unnecessary at this stage.

- *Rationale:* Keeping hardware requirements low makes the system **cost-effective and easy to adopt**, especially for universities with limited resources.

5.1.3 Communication Standards

A critical part of BusKoi's design is the way communication takes place between different components of the system and its users. Since the system involves multiple stakeholders administrators, drivers, and students communication must be reliable, secure, and instantaneous. To achieve this, BusKoi follows widely accepted **web communication standards** that ensure both compatibility and scalability.

1. **HTTP/HTTPS Protocols**
    - BusKoi uses the **Hypertext Transfer Protocol (HTTP)** for general client-server communication. This is the global standard for exchanging data on the web and ensures that the system is accessible on any browser or device without requiring specialized software.
    - To enhance security, all communication is encrypted using **HTTPS**. This prevents unauthorized access, data leaks, or tampering during transmission. For example, when a student logs in, their credentials are protected by HTTPS so that sensitive information cannot be intercepted.
    - By relying on HTTP/HTTPS, BusKoi guarantees compatibility with existing institutional IT infrastructure and ensures that students and drivers can access the system on their mobile devices or computers without additional setup.
2. **WebSockets via Socket.IO**
    - For features that require **real-time updates**, such as schedule changes or driver notifications, BusKoi uses **WebSockets** through the Socket.IO library. Unlike HTTP, which works on a request-response model, WebSockets allow continuous, two-way communication between the server and clients.
    - This means that once a connection is established, updates can be pushed instantly from the server to the client without requiring repeated requests. For example, if an administrator changes a bus schedule, all affected users and drivers immediately receive the update on their dashboard.
    - Socket.IO was selected because it builds on WebSockets but also provides fallback options, event handling, and error recovery, making it more reliable for real-world use cases.
3. **Alternative Considered – SMS-Based Communication**
    - Before finalizing WebSockets, SMS-based notifications were considered as an alternative. SMS messages could notify students and drivers of schedule updates directly on their phones.
    - However, this method presented several disadvantages:
        - **Cost:** Each SMS carries a charge, making it unsustainable for large institutions where thousands of messages may be needed daily.
        - **Delay:** SMS delivery is not always instantaneous; network congestion or operator delays could cause late notifications.
        - **Scalability:** Managing SMS gateways and integrating them into the system would add unnecessary complexity.

4. **Rationale for Selection**

Use what the web already does best. Sticking to standard HTTP/HTTPS for requests and **WebSockets** for live events gives us speed, reliability, and low cost without exotic tech or vendor lock-in.

Secure by default. HTTPS encrypts every login, cookie, and API call, protecting sessions and data integrity. **WebSockets** then keep a secure, always-on channel so updates reach users instantly.

Built to grow. The same stack easily expands to **GPS streams**, **mobile apps**, and **real-time analytics** without a redesign. It also plays nicely with load balancers, proxies, and campus firewalls, making scaling and operations straightforward.

5.2 Impact on Society, Environment and Sustainability

5.2.1 BusKoi is more than a technical build it changes everyday routines, makes campus operations smoother, and supports greener transport. By replacing guesswork and paper notices with live, shared information, it helps people plan better, reduces waste, and aligns with long-term sustainability goals.

5.2.2 Impact on Life

For students. Real-time schedules replace guesswork with clarity. Instead of standing at a stop wondering if the bus has already left, students can check live timings from their phone and decide whether to head out now or grab five more minutes to print notes. That certainty helps with everything from getting to morning labs on time to coordinating group work across campuses. On exam days or in bad weather when anxiety is already high instant updates reduce stress and cut down on risky last-minute sprints. The app also supports different routines: commuters, dorm residents, evening students, and those with accessibility needs can all plan safer, shorter routes with fewer unnecessary waits.

For faculty and staff. When transport is predictable, classes start on time and stay on rhythm. Fewer late arrivals mean fewer mid-lecture disruptions, better attendance tracking, and smoother transitions between sessions. Office hours and meetings become easier to schedule because everyone is working from the same live timetable. The system also reduces admin overhead: instead of fielding calls about delays, staff can point to a single source of truth that updates automatically.

For drivers. A clear, role-specific dashboard replaces scattered phone calls and ad-hoc instructions. Drivers see today's assignments, route order, and timing in one place, then acknowledge trips and share a live location link with a tap. If there's congestion or a minor breakdown, an alert to the admin triggers fast reassignments or schedule tweaks, cutting confusion on the ground. That calmer workflow lowers stress, improves safety, and makes shift handovers cleaner because the digital trail shows what happened and when.

Bottom line. With everyone looking at the same real-time picture, punctuality improves, queues shrink, and frustration drops. Students make better time

decisions, instructors keep lessons on track, and drivers operate with fewer surprises. Day to day, that adds up to a more reliable campus routine—and a transport system people can trust.

5.2.3 Impact on Society & Environment

BusKoi's value goes beyond convenience for individual riders it improves how the whole campus community moves and lowers the environmental footprint of daily transport.

Societal impact. Everyone sees the same information at the same time, which cuts rumor and confusion. If a bus is delayed, students, drivers, and admins get the alert instantly, so there's no finger-pointing or guesswork. That shared "source of truth" builds trust, reduces complaints, and makes cooperation easier classes, meetings, and events can be planned around accurate, live updates instead of static notices.

Operational fairness and efficiency. Smarter scheduling spreads demand more evenly: no more chronically packed buses on one route while others run half empty. That balance improves comfort and reliability for riders and reduces the day-to-day firefighting for administrators (fewer calls, fewer manual reschedules). Clear data also makes it easier to adjust service levels during peak exams, bad weather, or special events.

Environmental benefits. Optimized routes and better timing mean fewer empty runs and less idling at stops. That directly lowers fuel use and cuts emissions (e.g., CO_2 and particulates), which saves money and supports campus sustainability goals. As GPS tracking and richer analytics are added, the system can fine-tune headways and detours even further, squeezing out more waste and improving air quality around busy corridors.

Bottom line. By reducing delays, dead miles, and miscommunication, BusKoi delivers a cleaner, calmer, and more predictable transport experience helping people, the institution, and the environment at the same time.

5.2.4 Ethical Aspects

Ethics sits at the core of BusKoi. The system is built to be fair, transparent, and secure so people can trust it every day.

Data-privacy.
Sensitive data is protected end-to-end. Passwords are never stored as plain text they're hashed (with salt) before saving. Traffic runs over secure channels, and we keep only the minimum information needed to deliver the service. This reduces the risk of leaks and misuse.

Role-based-access.
Everyone sees only what they need. Students can't touch admin tools; drivers don't see student details. This "least-privilege" approach keeps information where it belongs and limits the blast radius if something goes wrong.

Fair-and-inclusive-access.
Features are designed to benefit all stakeholders admins, drivers, and students without favoritism. Updates reach everyone at the same time, and the interface is kept simple and consistent so it works well across devices and for different user abilities.

Transparency-and-accountability.
Key actions are recorded: drivers confirm trips, admins log changes, and students receive time-stamped notifications. These digital traces create a clear, auditable history of what happened and when, which helps resolve disputes and improve service.

In short: BusKoi isn't just technically sound it's socially responsible, protecting user data, limiting access appropriately, treating users fairly, and making operations visible and accountable.

5.2.5 Sustainability Plan

From the start, BusKoi was built to last. We designed the system to scale as campuses grow, new routes are added, and more users come on board without needing to rip out and rebuild the core. The architecture is modular and API-driven, so new features can slot in cleanly and existing parts can be upgraded with minimal disruption.

What's-in-place-now
Today, BusKoi delivers role-based dashboards and real-time notifications the essentials that solve immediate scheduling and communication problems on campus
.
What comes next (by design):

- **GPS-integration**
  Plug-and-play GPS modules can feed live location data directly into BusKoi. That removes the need for drivers to share manual links and unlocks accurate, map-based tracking (routes, ETAs, and stop-by-stop progress). Because the system already supports real-time events, GPS streams can be added without changing the rest of the app.
- **Mobile-apps-(Android/iOS)**
  Native apps will make access even smoother for students and drivers. Push notifications for delays and assignments, quick actions (confirm trip, share location), and optional offline caching will keep users informed even on spotty networks.
- **Analytics-and-reporting**
  Adding an analytics layer will turn raw events into insight: peak load times, on-time performance, route utilization, and dwell/idle patterns. Simple dashboards and exportable reports will help administrators tune headways, balance capacity, and plan resources with data, not guesswork.
- **Cloud-scalability**
  Hosting on scalable cloud platforms allows the system to grow with demand more users, more routes, more notifications while keeping performance steady and costs predictable. Horizontal scaling, managed databases, and CDN-backed assets make it easy to handle exam weeks, events, or new campuses.

5.3 Project Management and Financial Analysis

The BusKoi project was developed using mostly open-source tools, which kept the development cost minimal. The only significant cost is expected from hosting and ongoing maintenance. Below is a cost analysis table:

Table 5.1: cost calculation table

Item	Description	Cost
Backend Framework	Flask (Open-source)	Free
ORM & DB Library	SQLAlchemy, Flask-Migrate (Open-source)	Free
Database	MySQL Community Edition	Free
Frontend	Tailwind CSS (Open-source)	Free
Version Control	Git & GitHub (Free tier)	Free
Testing Tool	Postman (Free version)	Free
IDE/Editor	VS Code	Free
Hosting	Cloud hosting (e.g., AWS/Heroku/DigitalOcean)	Approx. $10–20/month
Domain & SSL	Custom domain + SSL certificate	Approx. $15–30/year
Maintenance	Database backup, server monitoring	Approx. $5–10/month

In summary, the development cost was almost zero due to the use of open-source tools. The primary expenses are for hosting, domain, and maintenance, making the project highly cost-effective.

5.4 Complex Engineering Problem

5.4.1 Complex Problem Solving

Table 5.2: Mapping with Complex Engineering Problem.

EP1 Dept of Knowledge	EP2 Range Of Conflicting Requirements	EP3 Depth of Analysis	EP4 Familiarity of Issues	EP5 Extent of Applicable Codes	EP6 Extent Of Stakeholder Involvement	EP7 Interdependence
✓	✓	✓	✓	✓	✓	✓

EP1 Depth of Knowledge:

BusKoi blends multiple layers of engineering knowledge. At the foundation, we designed a normalized relational schema and transaction rules so that a schedule, driver, and bus cannot contradict one another. On top of that, we used specialist tooling Flask, SQLAlchemy, Flask-SocketIO, and secure session handling to turn those rules into a working service. Design knowledge was needed to split the system into clear modules (Users, Drivers, Buses, Routes, Schedules, Notifications, Settings) with clean interfaces. Finally, we exercised engineering practice through migrations, seed scripts, and automated checks so that changes roll out safely without breaking production data.

EP2 Range of Conflicting Requirements:
Real-time updates are valuable but can overload the server if every page constantly polls. We balanced speed with resource limits by using Socket.IO for push-based updates, caching expensive queries (like "today's live schedules"), and paginating heavy lists in admin screens. Security versus convenience was another tension: we enforced strict role checks and CSRF protection but kept common tasks one-click (e.g., assign driver → auto-notify). We also weighed simplicity against completeness only the features essential for a university context made the first release so the UI stayed clean and fast.

EP3 Depth of Analysis:
There is no off-the-shelf product that fits university shuttles. We mapped real campus rhythms class blocks, exam days, event peaks, and route turnarounds then stress-tested edge cases: two drivers on one bus, a bus covering two routes in one morning, and surprise delays from weather or traffic. These scenarios shaped validation rules (e.g., overlapping schedules on the same bus are blocked) and workflow design (e.g., rescheduling triggers immediate notifications and updates the live board within seconds).

EP4 Familiarity of Issues:
Most institutions still coordinate buses through paper notices or group calls, which leads to last-minute confusion. We cataloged the familiar failure modes notice not seen, driver not informed, student waiting at the wrong gate and translated them into product guardrails: role-based dashboards, unread notifications, and time-boxed "live until" windows so users can trust what they see.

EP5 Applicable Codes/Standards:
There is no single "university transport portal" standard. We therefore anchored to widely adopted web standards and policies: HTTPS everywhere, salted password hashing, CSRF protection, secure cookies, and audit-friendly logging of admin actions. For data portability we kept RESTful endpoints and predictable JSON so future mobile apps or GPS modules can integrate without rewrites.

EP6 Stakeholder Involvement:
Admin, driver, and student needs often diverge. We ran the system around those roles: admins get high-leverage CRUD tools and bulk actions; drivers get a minimal, readable schedule with one-tap confirmation and a field for a live location link; students see a quiet, read-only view with search and instant updates. Notifications are tailored to each role so the right person sees the right alert at the right time.

EP7 Interdependence:
A single action ripples across modules. When an admin creates a schedule, the system validates the bus capacity and route timing, locks the bus for that window, assigns a driver, writes an audit entry, and emits a Socket.IO event so the driver dashboard and the public schedule refresh immediately. That tight coupling meant we invested in database constraints and transaction boundaries to keep the whole chain consistent.

Mapping with Knowledge Profile

Table 5.3: Mapping with knowledge Profile.

K3 Engineering Fundamentals	K4 Specialist Knowledge	K5 Engineering Design	K6 Engineering Practice	K8 Research Literature
✓	✓	✓	✓	✓

K3 Engineering Fundamentals:
We applied data modeling, indexing, foreign-key constraints, and transaction design to protect integrity—for example, a schedule cannot exist without a valid bus and route, and deletes are guarded to prevent orphaned records. Authentication and authorization models follow principles of least privilege so each role only accesses what it needs.

K4 Specialist Knowledge:
Flask gave a small, composable core; SQLAlchemy handled ORM patterns and migrations; Socket.IO delivered push notifications without wasteful polling. Together they enabled real-time UX while staying simple to operate on common university servers.

K5 Engineering Design:
We separated concerns into modules and REST endpoints so features can evolve independently: /buses, /routes, /drivers, /schedules, /notifications. The UI mirrors this structure—each dashboard presents only the actions relevant to its role, reducing error and training time.

K6 Engineering Practice:
We seeded a minimal dataset for local testing, wrote unit tests for models and forms, and used migration scripts for safe schema changes. Manual route walks and end-to-end checks validated live flows: create schedule → notify → driver confirms → students see it update within seconds.

K8 Research & Prior Art:
We reviewed university shuttle portals and real-time systems to understand what works (clear lists, live indicators) and what usually fails (feature bloat, heavy GPS at v1). That informed a "lightweight first" release that still leaves clean hooks for GPS and mobile apps later.

5.4.2 Engineering Activities

Mapping with Complex Engineering Activities

Table 5.4: Mapping with Complex Engineering Activities.

EA1 Range of resources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society & environment	EA5 Familiarity
✓	✓	✓	✓	✓

EA1 Range of Resources:
Human: admins, drivers, students. Technical: a modest web server, MySQL, Socket.IO workers, and storage for logs/backups. This mix keeps costs low while supporting busy hours like morning classes.

EA2 Level of Interaction:
High collaboration across roles. Admin actions immediately inform drivers; driver confirmations and delay flags instantly inform students. This feedback loop reduces calls and prevents contradictory information.

EA3 Innovation:
Most solutions are either heavy (enterprise) or personal (ride-sharing). BusKoi is a lean, role-based, real-time tool shaped specifically for universities, built entirely on free/open-source software so institutions can adopt it without licensing hurdles.

EA4 Consequences for Society & Environment:
Better coordination shortens waits, reduces idle trips, and lowers fuel use. Students reach class on time; drivers face less confusion; admins spend fewer hours firefighting. The ripple effects are calmer mornings and a smaller carbon footprint.

EA5 Familiarity:
We replaced the paper-notice workflow with a traceable digital process. The features feel familiar—lists, schedules, announcements—but now they are consistent, searchable, and live, which lifts overall reliability without forcing people to relearn everything.

5.5 Summary

The BusKoi project addresses a complex engineering problem by combining multiple modules (bus, route, schedule, notifications) into a unified system. It requires deep knowledge of software engineering and balancing conflicting needs like real-time updates and performance. The knowledge mapping shows strong use of fundamentals, specialist tools, design, and practice. The engineering activities highlight resource use, stakeholder interaction, innovation, and clear benefits for both institutions and users, making BusKoi a practical and impactful solution.

Chapter 6

Conclusion

6.1 Summary

Universities have long depended on paper notices, phone calls, and static web pages to run their buses methods that can't keep up with last-minute changes. The result is familiar: delays, mixed messages, and a lot of wasted time for students, staff, and drivers.

BusKoi replaces that patchwork with one place to manage everything, live. Admins create and update schedules, assign drivers, and push instant alerts from a central dashboard. Drivers get a clear daily plan and can share live updates without juggling calls. Students and staff see accurate, real-time schedules on any device, so they can plan their day with confidence.

Under the hood, we chose a stack that stays fast and affordable: **Flask** for a lightweight backend, **SQLAlchemy** with **MySQL** for clean, reliable data, **Flask-Socket.IO** for real-time updates, and **Tailwind CSS** for a responsive UI. In testing, the system handled real workflows smoothly communication improved, punctuality went up, and small issues were caught early instead of becoming daily headaches.

In short, BusKoi is a practical, campus-ready solution. It makes daily commuting simpler for students and staff and gives administrators a dependable way to manage resources now, and as the system grows.

6.2 Limitation

BusKoi meets its main goals, but a few gaps remain in this version:

1. **No-built-in-GPS-yet**
   Right now, drivers share a live location link manually. It works, but it's not as precise or reliable as automatic, device-based GPS tracking on a map.
2. **Web-only-experience**
   The system runs well in the browser and adapts to phones and laptops, but there's no native mobile app. That means no push notifications, limited offline support, and fewer OS-level conveniences.
3. **Basic-analytics**
   BusKoi focuses on schedules and alerts. Deeper insights—like route load, on-time performance, or long-term trends—aren't built in yet.

These aren't deal-breakers; they simply mark the next places to grow. The core platform is solid and already improves day-to-day transport—these additions would make it even stronger.

6.3 Future Work

To take BusKoi from "works well today" to "smart and truly seamless," we'll focus on a few high-impact upgrades:

1. **GPS-powered-live-tracking**
   Built-in GPS on buses would stream locations to a map in real time. Students and staff could see accurate ETAs without driver-updated links, and admins could spot delays or bottlenecks faster.
2. **Native-mobile-apps-(Android/iOS)**
   Apps would make everyday use effortless: one-tap access to schedules, push notifications for delays or reroutes, and optional offline caching for spotty networks. This also opens the door to OS-level features (widgets, background updates).
3. **Advanced-analytics-&-reporting**
   A reporting layer can surface route load, peak demand, on-time performance, and recurring issues. With clear dashboards and exports, admins can tune headways, balance capacity, and plan resources with data not guesswork.
4. **Cloud-first-scalability**
   Deploying on scalable cloud infrastructure will keep performance steady as users and routes grow. Managed databases, autoscaling, and CDN-backed assets will handle exam weeks, events, and multi-campus expansion smoothly.
5. **Richer-features-for-students-&-drivers**
   Planned additions include student ride reservations, driver shift planning, and calendar integration (class timetables, exam slots, events). These tools reduce back-and-forth and keep everyone aligned.

Bottom line: Addressing these areas will turn BusKoi into a comprehensive, adaptable transport platform ideal for universities now and easily extendable to schools, corporate campuses, and other organizations.

References

[1] Hasan, M.N. and Hossen, M.S., 2019, May. Development of an android based real time bus tracking system. In 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT) (pp. 1-5). IEEE.

[2] Jain, S., Trivedi, A. and Sharma, S., 2019, February. Application based bus tracking system. In 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon) (pp. 152-154). IEEE.

[3] de la Cruz, C., Pacheco, A., Robles, I., Duran, A. and Flores, E., 2021. Smart transportation system for public universities. International Journal of Information Technology, 13(4), pp.1643-1647

[4] Gull, H., Aljohar, D., Alutaibi, R., Alqahtani, D., Alarfaj, M. and Alqahtani, R., 2021, June. Smart school bus tracking: requirements and design of an iot based school bus tracking system. In 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 388-394). IEEE.

[5] Mohd Shabli, A.H., Md. Rejab, M., Zulkifli Abai, N.H. and Chit, S.C., 2023, February. Campus bus tracking system using Lora technology: IoT system testing. In Proceedings of the 2023 12th International Conference on Software and Computer Applications (pp. 234-239).

[6] Sobhana, M., Chowdary, T.R., Venkatesh, M.S. and Devendra, K.S., 2023, March. Smart Campus Bus Tracking Alert System Using Real-Time GPS. In 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS) (Vol. 1, pp. 1777-1781). IEEE.

[7] Vincent, B., Sabu, J., Mathew, C., Nair, S.S. and George, S.B., 2023, March. Live college bus tracking and route mapping using internet of things. In 2023 2nd International Conference on Computational Systems and Communication (ICCSC) (pp. 1-7). IEEE.

[8] Alamatsaz, K., Quesnel, F. and Eicker, U., 2024. Enhancing Electric Shuttle Bus Efficiency: A Case Study on Timetabling and Scheduling Optimization. Energies, 17(13), p.3149.

[9] Jabin, N., Al Noman, A., Parvin, S., Kader, A., Aktar, T. and Uddin, I., 2022. Transportation Service and Student's Satisfaction: A Study on Dhaka University. Indian Journal of Social Science and Literature, 1(4), pp.6-13.

[10] Rungskunroch, P., Maneerat, P. and Srikram, P., Simulation-Based Optimisation Of University Shuttle Bus Systems: A Case Study on Route Efficiency and Energy Utilisation. Available at SSRN 5298309.

.

213-15-4565

ORIGINALITY REPORT

7%	6%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	2%
2	iecscience.org Internet Source	1%
3	arxiv.org Internet Source	<1%
4	journal.ugm.ac.id Internet Source	<1%
5	jseepublisher.com Internet Source	<1%
6	ijrti.org Internet Source	<1%
7	Submitted to Rose-Hulman Institute of Technology Student Paper	<1%
8	Submitted to University of Florida Student Paper	<1%
9	Submitted to University of Nottingham Student Paper	<1%
10	www.journals.latticescipub.com Internet Source	<1%
11	Submitted to North West Regional College Student Paper	<1%
12	ijsrcseit.com Internet Source	<1%