

Mini-Project Report On

**Bus-T App (An Application for bus travellers to take
ticket with the help of mobile phone)**

*Submitted in partial fulfillment of the requirements for the
award of the degree of*

Bachelor of Technology

in

Computer Science & Engineering

By

**Martin Francis Paul (U2003130)
Muhammed Shinaz K P (U2003137)
Hemdan M K (U2003092)
Hariraman M (U2003091)**

**Under the guidance of
Dr. Uma Narayanan**



**Department of Computer Science & Engineering
Rajagiri School of Engineering and Technology (Autonomous)
Rajagiri Valley, Kakkanad, Kochi, 682039**

July 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
RAJAGIRI SCHOOL OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
RAJAGIRI VALLEY, KAKKANAD, KOCHI, 682039



CERTIFICATE

*This is to certify that the mini-project report entitled "**Bus-T App (An Application for bus travellers to take ticket with the help of mobile phone)**" is a bonafide work done by **Mr. Martin Francis Paul (U2003130), Mr. Muhammed Shinaz K P (U2003137), Mr. Hemdan M K (U2003092), Mr. Hariraman M (U2003091)**, submitted to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2020-2024.*

Dr. Preetha K. G.
Head of Department
Professor
Dept. of CSE
RSET

Mr. Uday Babu P.
Mini-Project Coordinator
Asst. Professor
Dept. of CSE
RSET

Dr. Uma Narayanan
Mini-Project Guide
Asst. Professor
Dept. of CSE
RSET

ACKNOWLEDGEMENTS

We wish to express our sincere gratitude towards **Dr. P. S. Sreejith**, Principal of RSET, and **Dr. Preetha K. G.**, Professor, Head of the Department of Computer Science and Engineering for providing us with the opportunity to undertake our mini-project, "Bus-T App".

We are highly indebted to our mini-project coordinators, **Mr. Uday Babu P**, Assistant Professor, Department of Computer Science and Engineering, and **Ms. Tripti C**, Assistant Professor, Department of Computer Science and Engineering

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our mini-project guide **Dr. Uma Narayanan**, for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Martin Francis Paul

Muhammed Shinaz K P

Hemdan M K

Hariraman M

ABSTRACT

Even though payment modes have changed from cash to digital in many firms, live bus ticket booking in our state is primarily done through cash. Additionally, the tickets are paper-based, which are eventually discarded. Our project aims to automate live bus ticket booking through QR scanning. The system is an app developed with Flutter SDK and Firebase. The app consists of two main modules: the passenger side and the conductor side. On the passenger side, passengers can add money to their wallets and scan the QR code sticker inside the bus to make payments from their wallets. A ticket will be generated for the specific journey. On the conductor side, the conductor manages bus routes and activates a particular route at the start of the journey. When the conductor asks passengers to show their tickets, they can display the generated ticket, which the conductor will verify using the always-on display feature containing his active bus route's tickets list. Passengers can only book tickets if the conductor has activated a route on his list of bus routes. Through our project, we aim to provide live ticket booking, ensuring that a ticket is booked only when passengers are inside the bus. Furthermore, the tickets will be paperless, and the payment method will be contactless.

Contents

Acknowledgements	ii
Abstract	iii
List of Figures	vii
1 Introduction	1
1.1 Background	1
1.2 Existing System	2
1.3 Problem Statement	2
1.4 Objectives	2
1.5 Scope	3
2 Literature Review	4
2.1 Manual Ticketing with Paper Tickets	4
2.2 Contactless Smart Cards	4
2.3 Punch Cards	4
2.4 NFC and RFID Technology	5
2.5 Handheld Machines	6
3 System Analysis	8
3.1 Expected System Requirements	8
3.2 Feasibility Analysis	8
3.2.1 Technical Feasibility	8
3.2.2 Operational Feasibility	8
3.2.3 Economic Feasibility	8
3.3 Hardware Requirements	9
3.4 Software Requirements	9
3.4.1 Visual Studio Code for flutter app development	9

3.4.2	Firestore	10
4	Methodology	11
4.1	Proposed Method	11
4.1.1	QR Scanner	11
4.1.2	Wallet Implementation	12
4.1.3	QR Generator	13
4.1.4	Live Ticket Display	14
5	System Design	16
5.1	Architecture Diagram	16
5.2	Usecase Diagram	18
6	System Implementation	20
6.1	Passenger login/signup	20
6.2	Conductor login/signup	20
6.3	QR Scanner	21
6.4	Ticket Generator	21
6.5	Payment Wallet	21
6.6	QR Generator	22
6.7	Route Management	22
6.8	Live Ticket Display	23
7	Testing	24
7.1	Passenger Side Testing	24
7.1.1	QR Code Scanning	24
7.1.2	Ticket Booking	24
7.1.3	Payment Integration	24
7.1.4	Ticket Generation	25
7.1.5	Firestore Integration	25
7.2	Conductor Side Testing	26
7.2.1	Live Ticket Display	26
7.2.2	Ticket Verification	26

7.2.3	Route Activation/Deactivation	26
7.2.4	Firestore Integration	27
7.3	Integration Testing	28
7.4	System Testing	28
7.5	Cross-Device and Cross-Platform Testing	28
7.6	Security Testing	28
8	Results	29
9	Risks and Challenges	34
10	Conclusion	35
	References	36
	Appendix A: Base Paper	36
	Appendix B: Sample Code	48
	Appendix C: CO-PO and CO-PSO Mapping	72

List of Figures

5.1	Architecture diagram	16
5.2	Usecase Diagram	18
7.1	Passenger details	25
7.2	Ticket details	26
7.3	Conductor details	27
7.4	Route details	27
7.5	Live Ticket	28
8.1	Splash screen	29
8.2	Welcome Screen	29
8.3	login and signup	29
8.4	Passenger Home	29
8.5	Wallet	30
8.6	QR Scanner	30
8.7	Ticket	30
8.8	Price Calculation	30
8.9	Ticket History	31
8.10	Conductor Home screen	31
8.11	QR Generator	31
8.12	Route Management	32
8.13	Add Route	32
8.14	Route Details	32
8.15	Route activating Screen	32
8.16	Deactivate Route screen	33
8.17	Live Tickets	33
8.18	Payment History	33

Chapter 1

Introduction

Bus transportation is a popular mode of travel for millions of people around the world. However, the traditional method of purchasing bus tickets can be inconvenient and time-consuming. Passengers often have to wait in line to buy tickets from a conductor, and the process can be slow and inefficient.

1.1 Background

The bus ticket automation system that you are proposing is a promising new technology that has the potential to improve the efficiency and security of bus transportation. The system uses two interfaces: a passenger side and a conductor side. The passenger side allows users to purchase tickets by selecting their from and to locations, and then paying for the tickets using a mobile payment app. The conductor side generates QR codes for each ticket, which are then scanned by passengers to validate their tickets. The tickets are also visible in the conductor's live ticket area, so they can easily keep track of who has paid for a ticket and who has not.

The system is implemented using Flutter and Firebase. Flutter is a cross-platform mobile development framework that allows developers to create native-looking apps for iOS and Android. Firebase is a backend-as-a-service platform that provides a variety of features, including real-time database, cloud storage, and authentication.

The bus ticket automation system has the potential to revolutionize the way that bus tickets are purchased and validated. The system is convenient, secure, and efficient, and it has the potential to save bus operators time and money.

1.2 Existing System

For the most part, in the transportation framework, the procedure of the ticket is for the most part taken care of by the conductor. Fundamentally, the entire procedure is finished by paper-based, the printed papers are issued as tickets. In introducing days the handholding machines are utilized to print the tickets this framework is having different weaknesses it additionally kept up by the RFID framework, in which tag and per user are utilized output the cards.

1.3 Problem Statement

The challenge is to create a mobile application to make bus ticket process in metropolitan and public transport as an easy way by QR code system .

More amount of paper is needed to print the Ticket. If a passenger wish to travel in a bus. He has to carry money with him. Conductor also need to carry change. Handheld ticketing machine is comparatively slow and need trained person to operate it.

1.4 Objectives

- **To increase convenience for passengers-** The system will allow passengers to purchase tickets quickly and easily using their mobile phones. This will save them time and hassle, and it will also allow them to purchase tickets from anywhere, at any time.
- **To reduce time spent by conductors -** The system will eliminate the need for conductors to collect cash or issue paper tickets. This will free up their time so that they can focus on other tasks, such as ensuring passenger safety and providing customer service.
- **To improve accuracy of fare collection -** The system will automatically calculate the fare based on the passenger's travel distance. This will help to ensure that passengers are paying the correct fare, and it will also help to prevent fare evasion

- **To increase security** - The system will track the number of tickets that have been sold. This will help to prevent ticket fraud, and it will also provide bus operators with valuable data about passenger travel patterns.
- **Improved customer satisfaction**- Passengers will appreciate the convenience and efficiency of the system, which will lead to increased customer satisfaction
- **Reduced costs**- The system will help to reduce costs for bus operators, such as the cost of paper tickets and the cost of cash handling.
- **Environmental benefits**-The system will help to reduce the environmental impact of bus transportation, as it will eliminate the need to print paper tickets.

1.5 Scope

The project will develop two mobile apps: a passenger app and a conductor app. The passenger app will allow users to purchase tickets by selecting their from and to locations, and then paying for the tickets using a mobile payment app. The conductor app will generate QR codes for each ticket, which are then scanned by passengers to validate their tickets. The tickets are also visible in the conductor's live ticket area, so they can easily keep track of who has paid for a ticket and who has not.

The project will also involve the development of a backend system that will store and manage the ticket data. The backend system will use a cloud-based database, such as Firebase, to store the ticket data. The system will also use a mobile payment gateway, such as Stripe, to process payments.

The project will be implemented using Flutter and Firebase. Flutter is a cross-platform mobile development framework that allows developers to create native-looking apps for Android. Firebase is a backend-as-a-service platform that provides a variety of features, including real-time database, cloud storage, and authentication.

In short, the project will develop a bus ticket automation system that will allow passengers to purchase tickets quickly and easily using their mobile phones. The system will also improve the accuracy of fare collection and increase security.

Chapter 2

Literature Review

2.1 Manual Ticketing with Paper Tickets

In many local bus systems, manual ticketing with paper tickets is still prevalent. Passengers purchase tickets directly from the bus driver or conductor upon boarding. The conductor may issue printed tickets or use a ticket book to tear off and hand over paper tickets to passengers.

This method is prone to human error and may lead to ticketing inconsistencies. It can be time-consuming during peak hours, causing delays and inconvenience for passengers. There is a risk of losing revenue due to unrecorded transactions or ticket fraud.

2.2 Contactless Smart Cards

Some cities and transit authorities use contactless smart card systems, where passengers can load funds onto a smart card or tap their credit/debit card for fare payment. The card is tapped on a reader upon boarding the bus for validation.

This method's setup cost can be high, causing some to hesitate. Infrastructure and readers need investment, it's something to contemplate. Managing the card distribution, a challenge to embrace, Lost, stolen, or damaged cards, a logistical maze. Privacy and security concerns, a topic that needs address, While encryption protects data, some passengers may still stress. Card clashes and double charging, an inconvenience to avoid, Accidents with multiple cards, something to be annoyed.

2.3 Punch Cards

Some local bus services use punch cards as a simple form of ticket automation. Passengers purchase a punch card with a certain number of rides, and the bus driver or conductor

punches the card for each trip taken. When all rides on the card are used up, passengers can purchase a new one.

Punch cards can be easily lost or damaged, leading to loss of rides and inconvenience for passengers. Managing and tracking remaining rides on the cards can be challenging, and replacing lost cards can be costly for the bus operator. It lacks real-time data and insights into passenger behavior and travel patterns.

2.4 NFC and RFID Technology

NFC (Near Field Communication) and RFID (Radio-Frequency Identification) are both contactless communication technologies commonly used in bus ticketing systems to enable quick and seamless fare collection. NFC and RFID operate based on radio waves and do not require physical contact between the card and the reader, making them convenient and efficient for use in public transportation.

In NFC-based ticketing, passengers use NFC-enabled smart cards or mobile devices (such as smartphones or smartwatches) to board the bus. When the NFC-enabled card or device is held close to the NFC reader on the bus, the fare is automatically deducted from the card's balance, allowing the passenger to enter without the need for manual ticket checks or cash payments.

Similarly, RFID-based ticketing works on the same principle of contactless communication. Passengers use RFID cards, which contain a chip and an antenna, to interact with the RFID readers on the bus. The bus reader detects the RFID card's unique identifier, deducts the fare from the card's balance, and grants access to the passenger.

While NFC and RFID technologies offer significant advantages for bus ticketing, they do have some drawbacks that need to be considered:

Initial Investment and Infrastructure: Implementing NFC or RFID technology requires an initial investment in installing card readers and backend systems on buses and at stations. This setup cost can be a significant factor, especially for smaller bus operators or public transportation systems with limited budgets.

Card Distribution and Management: Distributing NFC or RFID cards to all passengers and managing the card inventory can be a logistical challenge. Ensuring that passengers have easy access to these cards and dealing with lost or damaged cards may require

additional resources and effort.

Compatibility with Devices: For NFC-based ticketing through mobile devices, there might be concerns about the compatibility of NFC readers with various smartphone models and operating systems. Older or less technologically advanced smartphones may not support NFC, potentially excluding some passengers from using mobile ticketing.

Privacy and Security: While NFC and RFID technology use encryption and security measures to protect data, there are still concerns about the potential for unauthorized access or data breaches. Passengers might worry about their personal information being at risk when using contactless cards or mobile devices for ticketing.

Technical Issues and Malfunctions: Like any technology, NFC and RFID systems can experience technical glitches, such as reader malfunctions or card-reading errors. Technical issues can lead to boarding delays, frustrated passengers, and potential revenue loss.

Dependency on Power Source: NFC and RFID card readers require a power source to operate. If the power source fails or is unavailable, the ticketing system may become unusable, causing disruptions in fare collection.

2.5 Handheld Machines

Handheld machines are portable electronic devices used in bus ticketing systems to facilitate fare collection, ticket validation, and passenger management. These machines are typically carried by bus conductors or ticket inspectors, allowing them to interact directly with passengers and ensure a smooth ticketing process.

In bus ticketing with handheld machines, the conductor or ticket inspector can scan or manually input ticket information into the device. The handheld machine can read various types of tickets, such as paper tickets with barcodes or QR codes. The device validates the ticket, checks its authenticity, and deducts the appropriate fare from the passenger's account or confirms its validity for travel.

Handheld machines offer several advantages in bus ticketing. They enhance the efficiency of fare collection, reduce the risk of revenue leakage, and enable real-time ticket validation. With these devices, bus operators can monitor passenger boarding and alighting patterns, and improve route planning and resource allocation.

However, handheld machines also have some drawbacks that need to be considered.

Firstly, they rely on battery power for operation, and if not adequately charged or maintained, they can run out of power during ticketing operations, causing disruptions and inconvenience to passengers.

Secondly, the use of handheld machines requires proper training for conductors or ticket inspectors. Operators must ensure that personnel are proficient in using the devices, scanning tickets correctly, and resolving any technical issues that may arise during ticket validation.

Additionally, the implementation of handheld machines may involve initial costs, including the purchase of devices and software, and ongoing expenses for maintenance and upgrades. These costs can be a significant consideration for smaller bus operators or those with limited financial resources.

Moreover, during peak travel times or in crowded buses, ticket validation with handheld machines may cause boarding delays, leading to passenger dissatisfaction. Conductors must efficiently handle ticket inspections to maintain the smooth flow of passengers.

Comparison

Each method has its own set of advantages and drawbacks. Manual ticketing with paper tickets is simple but can be inefficient and prone to errors. Contactless smart cards and NFC/RFID technologies offer faster and more efficient fare collection but require higher initial investments. Punch cards are straightforward but lack real-time data tracking. Handheld machines provide real-time validation but come with setup and maintenance costs and the need for proper staff training. The choice of method depends on factors such as budget, technological infrastructure, passenger preferences, and the level of automation desired by the bus operator or transportation authority.

Chapter 3

System Analysis

3.1 Expected System Requirements

The system of user which is a smart phone is expected to have the following features:

- Android platform with a version above 4.
- Requirement of Internet connection.
- A storage space of approximate 100 MB for the app.
- A minimum Ram size of 4GB is required in the device.

3.2 Feasibility Analysis

3.2.1 Technical Feasibility

The project is technically feasible since majority of the population are in possession of smartphones. The app only requires minimum requirements to run on a smartphone .

3.2.2 Operational Feasibility

The operations are built in a simple and easy to use manner for all type of people. Installation of the app is the only prerequisite operation to be done.

3.2.3 Economic Feasibility

The app can reduce the expense incurred by the bus owners since they need not need a handheld machines no ticket paper or training is required. The development of the app is also zero budget as it was built using free resources.

3.3 Hardware Requirements

The following are the system requirements to develop the Bus-Ti App.

- Processor: Intel Core i5
- Hard Disk: Minimum 100GB
- RAM: Minimum 8GB

3.4 Software Requirements

The following are the softwares used in the development of the app.

Operating System: Windows

3.4.1 Visual Studio Code for flutter app development

Visual Studio Code (VS Code) is a highly popular and efficient code editor that proves to be a perfect match for Flutter app development. Its user-friendly interface enables developers to focus solely on writing code without any distractions. With the dedicated Flutter extension, VS Code offers essential Flutter-specific features like code completion, formatting, and automatic imports, streamlining the development process. A highlight of the tool is its exceptional support for the Dart programming language used in Flutter development, providing syntax highlighting, error checking, and seamless code navigation. The renowned hot reload feature in VS Code allows developers to witness real-time updates to their app's user interface, dramatically speeding up the debugging and testing process. Coupled with debugging tools, widget tree visualization, and emulator integration, VS Code makes it effortless to run and troubleshoot Flutter apps. Additionally, the tool supports Git integration for version control and provides linting and code formatting for maintaining clean and consistent code. Its thriving extensions ecosystem further enriches the development experience by offering a wide range of community-built extensions, themes, and tools. All in all, Visual Studio Code proves to be an outstanding choice for Flutter app development, empowering developers to build robust and impressive apps efficiently.

3.4.2 Firebase

Firebase is a robust mobile and web development platform developed by Google, offering a suite of services and tools to simplify the app development process. At its core, Firebase provides real-time database capabilities, allowing data to synchronize instantly across clients. With Firebase Authentication, developers can easily add user sign-in and registration functionality with support for various authentication methods. Cloud Firestore, a scalable NoSQL database, offers more advanced querying and data structuring options for larger and complex applications. Firebase Cloud Storage facilitates secure cloud storage for user-generated files. Moreover, Firebase Hosting enables fast and reliable web app hosting, while Cloud Functions automates server-side logic. The platform also features analytics, crash reporting, performance monitoring, and cloud messaging services, providing valuable insights and enhancing the overall app experience. Firebase's seamless integration with Google services, coupled with its ease of use and scalability, makes it a popular choice for app developers looking to build powerful and real-time applications with minimum backend management.

Chapter 4

Methodology

4.1 Proposed Method

- Digitalizing this whole system of bus ticket by replacing it with a native application that will be cross-platform supported.
- The user will have their account in the app which will be logged in to the application in their mobiles.
- The system has two logins: Passenger side and Conductor side
- In passenger side passenger can add money to wallet, scan QR code, do payment and generate ticket.
- On conductor side conductor manage bus routes and view live tickets.

4.1.1 QR Scanner

The methodology of implementing a QR scanner in the bus ticket automation app using the Flutter scanner plugin. Here's an explanation of the code's methodology:

1. **Function 'scanQR()':** This function is responsible for initiating the QR code scanning process. When called, it checks the camera permission status using '**Permission.camera.status**'.
2. **Camera Permission Check:** If the camera permission is already granted, the app directly initiates the scanning process using '**scanner.scan()**'. The scanned QR code result is stored in the '**cameraScanResult**' variable.
3. **Processing the Scanned Result:** After successfully scanning the QR code, the function updates the app's state with the scanned result using '**setState**'. It then

navigates to the **'Busroutescreen'**, passing the **'documentId'** and **'userId'** parameters. The **'documentId'** represents the scanned QR code result, which likely contains information about the user or the ticket. The **'userId'** is presumably a user-specific identifier used in the app.

4. **Camera Permission Request:** If the camera permission is not granted, the function requests permission using **'Permission.camera.request()'**. If the user grants the permission, the QR scanning process is initiated similarly to the previously mentioned steps.
5. **Error Handling:** The code includes a **'try-catch'** block to handle potential exceptions that might occur during the QR scanning process. Any platform-specific exceptions are caught and printed to the console using **'print(e)'**.

In summary, the **'scanQR()'** function integrates QR code scanning into the bus ticket automation app. It checks for camera permission, requests permission if necessary, and then initiates the QR code scanning process. The scanned QR code result is used to navigate to the appropriate screen (**'Busroutescreen'**) and may provide essential information for further processing, such as identifying the ticket or the user associated with the QR code.

4.1.2 Wallet Implementation

The implementation of the functionality to add money to a user's wallet in the bus ticket automation app using the Razorpay payment gateway and Firestore for database operations. Here's an explanation of the code's methodology:

1. **User Interface:** The UI is created using the **'Scaffold'**, **'AppBar'**, **'Column'**, **'Text'**, **'TextField'**, and **'ElevatedButton'** widgets. The user enters the amount they want to add to their wallet using the **'TextField'**, and upon pressing the "Add to wallet" button, the payment process begins.
2. **Payment Integration:** The **'makePayment'** function handles the payment process. When the button is pressed, it creates a payment request with the specified amount and other details, such as the user's name, description, contact, and email.

The payment options are then passed to the Razorpay instance, and the payment gateway is opened to process the transaction.

3. **Payment Response Handling:** The **'handlePaymentSucess'**, **'handlePaymentFailure'**, and **'handleExternalWallet'** functions handle different payment response scenarios. If the payment is successful, the user's wallet amount is updated in Firestore using the update method, and a success message is displayed to the user. If the payment fails or an external wallet is used, appropriate error or wallet information messages are shown to the user using **'Fluttertoast'**.
4. **Firestore Integration:** The Firestore database integration is used to store and manage user wallet amounts. The user's wallet amount is retrieved from Firestore using a **'StreamBuilder'** that listens to changes in the Firestore document. The user's wallet amount is displayed in real-time to the user on the UI.
5. **Initialization and Disposal:** The **'initState'** method initializes the Razorpay instance and sets up event listeners for payment success, failure, and external wallet usage. The **'dispose'** method clears the Razorpay instance when the widget is disposed of to prevent any memory leaks.

Overall, this demonstrates how to implement a wallet system in the bus ticket automation app, allowing users to add money to their wallet using the Razorpay payment gateway and keeping the wallet amount updated in the Firestore database for real-time tracking and display.

4.1.3 QR Generator

The methodology of generating a QR code for the bus ticket automation app and downloading it as a PDF. Here's an explanation of the code's methodology:

1. **Variables and Initialization:** The code initializes several variables - **'qrCodeImageUrl'**, **'documentId'**, **'active'**, and **'collectionName'**. It also loads the QR code image when the home page is initialized using the **'loadQRCodeImage()'** function and retrieves shared preferences using **'loadSharedPrefs()'**.
2. **Loading QR Code Image:** The **'loadQRCodeImage()'** function retrieves the QR code image URL from Firebase Storage. It creates a reference to the image file

in Firebase Storage using `'firebasestorage.FirebaseStorage.instance'`, and then gets the download URL for the image using `'getDownloadURL()'`. The image URL is then set to the `'qrCodeImageUrl'` state.

3. **Shared Preferences:** The `'loadSharedPrefs()'` function loads the `'documentId'`, `'collectionName'`, and `'active'` values from shared preferences. These values likely represent the bus number, time, and the active status of the bus.
4. **Downloading QR Code as PDF:** The `'downloadQRCodeAsPDF()'` function downloads the QR code image from the network as a byte array using `'http.get()'`. It then creates a PDF document using the `'pw.Document()'` class from the `'pdf'` package and adds a page to the document with the QR code image.
5. **Temporary Directory and File:** The function retrieves the temporary directory path using `'getTemporaryDirectory()'` and saves the PDF document as a temporary file with a name based on the bus ID. The PDF file is written as bytes to the temporary file.
6. **Opening PDF File:** Finally, the function opens the PDF file using the default PDF viewer on the device, allowing the user to view and interact with the QR code.

The methodology showcased in the code enables the generation of a QR code for the bus ticket automation app, which can be downloaded as a PDF for further usage or sharing. The code utilizes Firebase Storage to store the QR code image and leverages shared preferences to access relevant data for generating and handling the QR code.

4.1.4 Live Ticket Display

The `'TicketScreen'` class is responsible for displaying the live tickets for a specific user in the bus ticket automation app. Here's an explanation of the code's methodology:

1. **Constructor:** The `'TicketScreen'` class has a constructor that receives two required parameters - `'documentId'` and `'collectionName'`. These parameters likely represent the user's unique identifier and the name of the collection in Firestore where the user's tickets are stored.

2. **StreamBuilder:** The **'body'** of the **'Scaffold'** contains a **'StreamBuilder'**, which listens to changes in the stream of tickets associated with the user. The stream is obtained from the **'TicketService'** class, which retrieves the live ticket data from Firestore.
3. **Snapshot Handling:** The **'builder'** function within the **'StreamBuilder'** handles different snapshot states - **'ConnectionState.waiting'**, **'hasError'**, no data, and having data. When the snapshot connection state is **'waiting'**, a loading indicator (**'CircularProgressIndicator'**) is displayed until the ticket data is fetched.
4. **No Tickets Found:** If the snapshot does not have data or the data list is empty, the app displays a message indicating that no tickets have been found for the user.
5. **Displaying Live Tickets:** If the snapshot contains ticket data, the app displays the live tickets using a **'ListView.builder'**. The **'ListView.builder'** iterates through the ticket data list, displaying each ticket as a **'Container'** with a color based on the **'ticket.color'** value fetched from Firestore. The **'ticket.id'** is displayed as the title for each ticket in a **'ListTile'**.
6. **Order of Display:** The tickets are displayed in reverse order, meaning the latest ticket is displayed first, achieved by using **'snapshot.data![snapshot.data!.length - index - 1]'**. This ensures that the most recent ticket appears at the top of the list.

Overall, the **'TicketScreen'** class efficiently fetches and displays the live tickets associated with the user. It dynamically updates the ticket list whenever changes occur in the Firestore collection, providing real-time updates to the user regarding their active tickets.

Chapter 5

System Design

5.1 Architecture Diagram

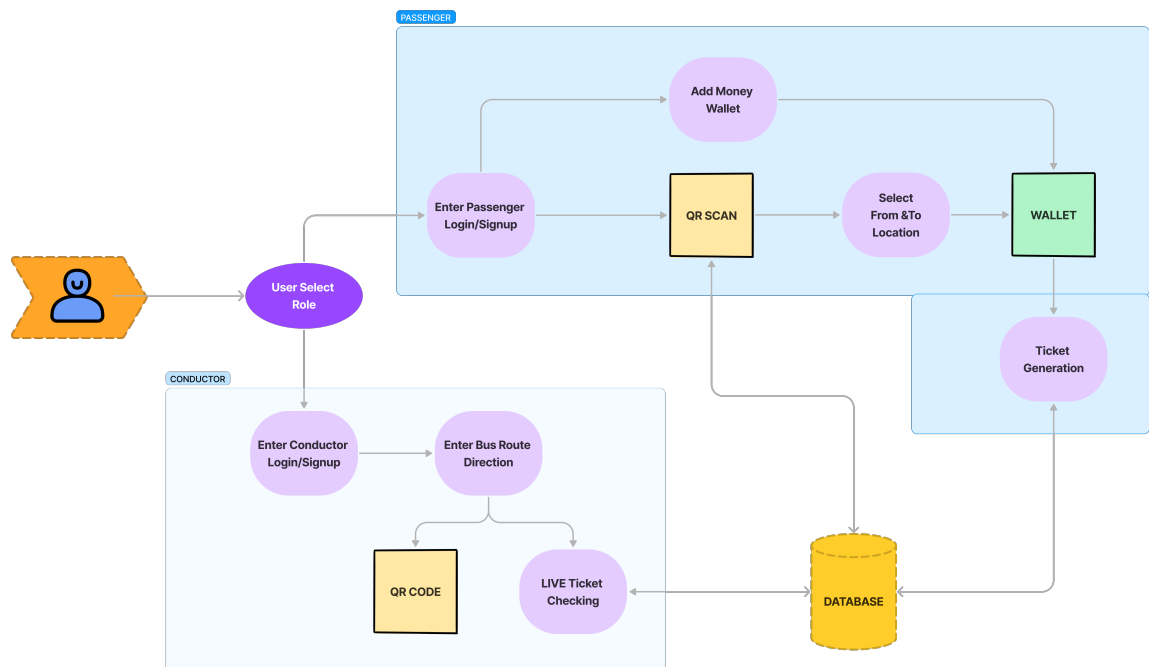


Figure 5.1: Architecture diagram

Architecture Diagram

The User will have the option to choose either of two choices that is Passenger or Conductor

Passenger Side

The passenger has to either log in or sign up to access the interface. The interface has two section QR scanner and add money to wallet. The passenger can add money to the wallet. The passenger can scan the QR code in the bus with the scanner and then enter the from and to locations he intend to travel and pay for it. The payment money will be deducted from the wallet. A ticket will be generated upon the payment.

Conductor Side

The Conductor has to either log in or register to access the interface. When registered a QR code generated and will be displayed in the interface. The Conductor interface has two sections route management and live ticket display. The conductor can set the route and activate it in route management and in live ticket display he can view the tickets generated by the passenger for the activated route.

Database Connectivity

The QR generated during the conductor registration is stored in database it is this QR that the passenger scan which will retrieve the route activated by the conductor since the route details are stored in the database. The wallet is also connected to database and when payment is done a ticket is generated with the required details collected from the database. These tickets are again stored in database and the live tickets page will retrieve them for verification

5.2 Usecase Diagram

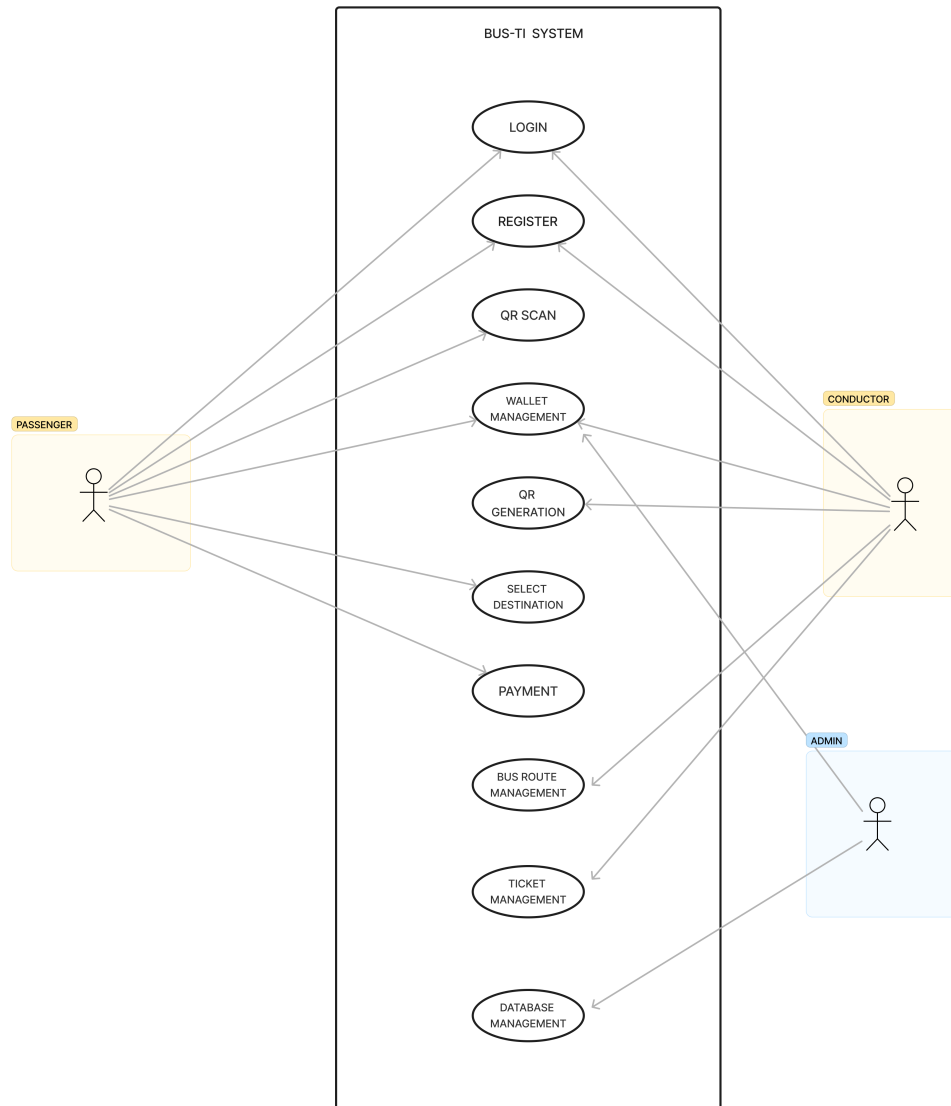


Figure 5.2: Usecase Diagram

Usecase Diagram

The given usecase has 3 actors, the passenger, conductor and Admin.

Passenger

The Passenger has the actions login, register, QR Scan, Wallet Management, Select destination and Payment. The passenger has to either log in or sign up to access the interface. The interface has two sections: QR scanner and add money to wallet. The passenger can add money to the wallet. The passenger can scan the QR code in the bus with the scanner and then enter the from and to locations he intends to travel and pay for it. The payment money will be deducted from the wallet. A ticket will be generated upon the payment.

Conductor

The Conductor has the actions login, register, Wallet Management, QR Generator, Route Management and Ticket Management. The Conductor has to either log in or register to access the interface. When registered a QR code is generated and will be displayed in the interface. The Conductor interface has two sections: route management and live ticket display. The conductor can set the route and activate it in route management and in live ticket display he can view the tickets generated by the passenger for the activated route.

Admin

The Admin has the actions Wallet Management and Database Management. The admin can access the database and they do the actions he requires such as edit data, retrieve data or delete data. And he can also access the Wallet implementation and in any case of error or malfunction he can access it and correct it.

Chapter 6

System Implementation

6.1 Passenger login/signup

The passenger would open the passenger app and click on the "Login" or "Sign up" button. The app would then authenticate the passenger with Firebase. Firebase is a cloud-based platform that provides authentication and other services for mobile apps. If the passenger is already logged in, they will be taken to the main screen of the app. If the passenger is not logged in, they will be taken to a login or sign up screen. On the login screen, the passenger would enter their email address and password. If the passenger's credentials are correct, they will be logged in to the app. If the passenger's credentials are incorrect, they will be prompted to try again. On the sign up screen, the passenger would enter their email address, password, and other information, such as their name and phone number. Once the passenger has entered their information, they would click on the "Sign up" button. Firebase would then verify the passenger's email address and create a new account for the passenger. If the account creation is successful, the passenger will be logged in to the app.

6.2 Conductor login/signup

The conductor would open the conductor app and click on the "Login" or "Sign up" button. The app would then authenticate the conductor with Firebase. Firebase is a cloud-based platform that provides authentication and other services for mobile apps. If the conductor is already logged in, they will be taken to the main screen of the app. If the conductor is not logged in, they will be taken to a login or sign up screen. On the login screen, the conductor would enter their bus ID and password. If the conductor's credentials are correct, they will be logged in to the app. If the conductor's credentials are incorrect, they will be prompted to try again. On the sign up screen, the conductor

would enter their bus ID, password, bus name, and phone number. Once the conductor has entered their information, they would click on the "Sign up" button. Firebase would then verify the conductor's bus ID and create a new account for the conductor. If the account creation is successful, the conductor will be logged in to the app.

6.3 QR Scanner

To purchase a ticket, the passenger would open the passenger app and click on the "QR Scanner" button. The app would then open the QR scanner and the passenger would scan the QR code on the bus. If the QR code is valid, the app would direct the passenger to a route selection page where they would select the route they want to travel and pay for the ticket. Once the passenger has paid for the tickets, the app would generate a ticket that contains the passenger's name, the bus ID, the route ID, the fare, and the time of purchase. The passenger would then be able to view their ticket in the app and show it to the conductor when they board the bus.

6.4 Ticket Generator

The ticket generator would be used by passengers to purchase tickets for bus travel. The passenger would select the route they want to travel, and the ticket generator would calculate the fare based on the distance of the route. The ticket generator would then deduct the fare from the passenger's wallet. Once the payment is processed, the ticket generator would generate a ticket. The ticket would contain the passenger's name, bus ID, route ID, fare, and time of purchase. The ticket would be stored in a database so that it can be easily retrieved.

6.5 Payment Wallet

A payment wallet is a digital wallet that stores payment information, such as credit card numbers, debit card numbers, and UPI IDs. Payment wallets allow users to make payments online or in-app without having to enter their payment information each time.

Payment wallet is implemented using a payment gateway, such as Razorpay. The payment gateway provides the infrastructure for storing and processing payments. The

payment wallet app provides the user interface for managing payments and making payments.

Payment wallets offer a number of benefits, including convenience, security, and flexibility. Payment wallets are convenient because users do not have to enter their payment information each time they make a payment. Payment wallets are secure because they use secure encryption to store payment information. Payment wallets are flexible because they can be used to make payments for a variety of goods and services.

6.6 QR Generator

At the time of SignUp a QR code is generated with the Bus Id and stored in database. When the conductor LogIn the QR is retrieved and displayed in the conductor interface. It can be downloaded so it can be stuck on the bus so the passenger can scan it.

6.7 Route Management

The route management page in the conductor side of the bus ticket automation app plays a pivotal role in efficiently handling bus routes and ensuring smooth operations. With an intuitive user interface, conductors can effortlessly add new routes by providing essential details like route name, starting and ending points, intermediate stops, distance from starting location, and price per km. Additionally, unwanted routes that are no longer in service can be promptly deleted, preventing passengers from selecting them for ticketing. When a new route is ready for service or an existing one is reactivated, conductors can easily activate it on the management page, making it visible and available for ticket booking. On the other hand, if a route needs to be temporarily suspended or undergo maintenance, conductors can deactivate it, temporarily removing it from passenger options. Furthermore, conductors have the flexibility to edit existing route details, including updates to route information, adjustments to intermediate stops and fare changes. By offering these functionalities, the route management page empowers conductors to efficiently control and keep routes up-to-date, ensuring passengers have accurate and accessible route options for their journeys. This streamlined route management process enhances the overall user experience and contributes to the seamless functioning of the bus ticket automation system.

6.8 Live Ticket Display

The live ticket display is a crucial feature on the conductor side of the bus ticket automation app, providing real-time access to all the tickets generated by passengers for an activated route. This page serves as a digital dashboard that allows the conductor to monitor and verify passenger tickets quickly and efficiently.

When a route is activated and passengers start booking tickets, the live ticket display populates with the details of these tickets. Each ticket is uniquely identified by a ticket ID or a color code for easy recognition. The display typically includes essential information such as the ticket id, boarding and destination points, fare paid, and the validity period.

The conductor can actively use the live ticket display during the journey to check and verify passengers' tickets as they board the bus. By cross-referencing the ticket ID or color code displayed on the passenger's mobile app with the information on the live ticket page, the conductor can ensure that each passenger has a valid ticket for the route they are traveling on.

The live ticket display is a valuable tool for conductors as it provides real-time insights into passenger ticketing status, facilitating smoother ticket validation processes and enhancing the overall efficiency of the bus ticketing system. With this feature, conductors can perform their duties more effectively, ensuring a seamless and hassle-free travel experience for passengers while maintaining accurate fare collection and compliance with the ticketing rules.

Chapter 7

Testing

7.1 Passenger Side Testing

7.1.1 QR Code Scanning

Tested the QR code scanning feature to ensure that passengers can scan the QR code generated on the conductor side successfully, and verified that the app can read and interpret the QR code data accurately.

In Figure 8.6 we can see the generated UI and the working scanner the result of the scanner is i will retrieve the corresponding details and routes of the bus Id which the QR is generated.

7.1.2 Ticket Booking

Tested the ticket booking process, including selecting the "from" and "to" locations, and verified that the app correctly displays available routes and fares.

In Figure 8.8 we can see the interface to select the from and to locations and the distance and corresponding price calculation

7.1.3 Payment Integration

Conducted tests to ensure smooth payment processing and that wallet payment method is properly integrated into the app.

In Figure 8.4 we can see the wallet and in Figure 8.5 we can see the interface to add money to the wallet which will be redirected to razorpay.It is using this wallet the payment for ticket is done

7.1.4 Ticket Generation

Verified that once payment is successfully made, the app generates a valid ticket for the selected route, and all relevant ticket details are displayed correctly.

In Figure 8.7 we have the generated ticket for a particular bus for a specified route. The ticket has all the relevant ticket details in it. The ticket also has a color embedded in it for easy recognition for the conductor.

7.1.5 Firebase Integration

Tested the integration with Firebase to ensure that passenger data, ticket information, and payment details are securely stored and retrieved from the database. Firebase integration on the passenger side of the bus ticket automation app is essential for various functionalities, including user authentication, ticket booking, and real-time data synchronization.

In Figure 7.1 we we can see the passenger details of the passenger which the entered for sign up and in Figure 7.2 we have the ticket details of the generated ticket stored under a specific ticket Id. The ticket Id in this collection is compared with the ticket Id in the live ticket collection (Figure 7.5) for the ticket verification by the conductor.

🏠 > users > 8590524668 More in Google Cloud		
bus-ti	users	8590524668
+ Start collection	+ Add document	+ Start collection
buses	8590524668 >	tickets
conductors	9567867353	
tickets	987654320	
usermap		
users >		+ Add field
		Email: "shinazmuhammed246@gmail.com"
		Name: "Muhammed Shinaz K P"
		Password: "123456"
		Phone Number: "8590524668"
		walletAmount: 285.5

Figure 7.1: Passenger details

Home > users > 8590524668 > tickets > 9599-16899574.. More in Google Cloud		
<div>8590524668</div> <div>+ Start collection</div> <div>tickets</div> <div>+ Add field</div> <div>Email: "shinazmuhammed246@gmail.com"</div> <div>Name: "Muhammed Shinaz K P"</div> <div>Password: "123456"</div> <div>Phone Number: "8590524668"</div> <div>walletAmount: 285.5</div>	<div>tickets</div> <div>+ Add document</div> <div>9599-1689957451915-D26</div> <div>9599-1689957474786-875</div> <div>9599-1689957489570-358</div> <div>9599-1689957503992-206</div>	<div>9599-1689957451915-D26</div> <div>+ Start collection</div> <div>+ Add field</div> <div>bookingDateTime: July 21, 2023 at 10:07:31 PM UTC+5:30</div> <div>busname: "Kay Jay"</div> <div>busno: "KL33N9599"</div> <div>color: 4283215696</div> <div>destination: "Triupunithara"</div> <div>start: "Chanaganassery"</div> <div>ticketcharge: 150</div>

Figure 7.2: Ticket details

7.2 Conductor Side Testing

7.2.1 Live Ticket Display

Tested the live ticket display feature to confirm that conductors can view all valid tickets generated by passengers for the activated route. Ensure real-time updates of ticket status.

In Figure 8.17 we have a Live Tickets Page where the passenger generated tickets are displayed.

7.2.2 Ticket Verification

Conducted tests to verify that conductors can check the validity of tickets by matching ticket IDs or color codes displayed on the passenger's app with the information shown in the live ticket area.

The conductor can check the details of the ticket generated by passenger (Figure 8.7) and the tickets displayed in live tickets (Figure 8.17) are matching or not.

7.2.3 Route Activation/Deactivation

Tested the ability of the conductor to activate and deactivate specific routes to control ticket availability for passengers.

In Figure 8.15 the routes of the specified bus is shown and from there the conductor can activate the required route. And he later deactivate it by clicking the deactivate button

in Deactivate route screen(Figure 8.16).

7.2.4 Firebase Integration

Similar to the passenger side, conducted tests to ensure seamless integration with Firebase to access live ticket data, route details, and other relevant information.

In Figure 7.3 we have the conductor details and the generated QR codes firebase storage link, In Figure 7.4 we have the route details set by the conductor and in Figure 7.5 we can see the live tickets for the activated route.

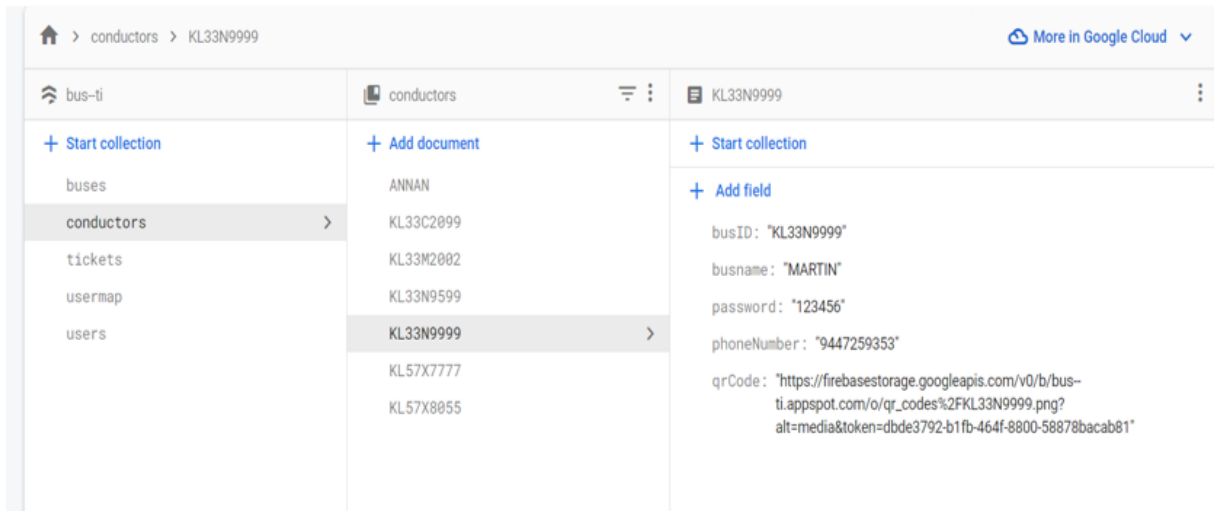


Figure 7.3: Conductor details

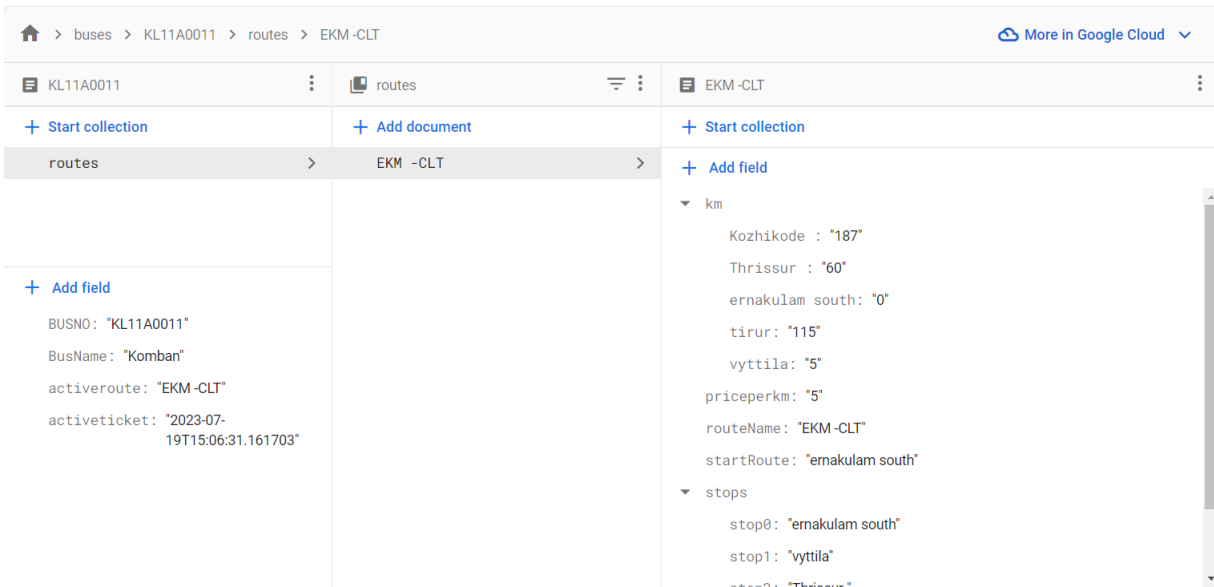


Figure 7.4: Route details

Home > tickets > KL11A0011 > 2023-07-19T14:... More in Google Cloud		
tickets	KL11A0011	2023-07-19T14:52:53.635554
+ Add document	+ Start collection	+ Add document
KL11A0011 >	2023-07-19T14:52:53.635554 >	0011-1689758687195-FC4 0011-1689758714007-0F1 0011-1689758731983-F5B ACTIVE ROUTE: EKM -CLT
KL33N9599	2023-07-19T15:06:31.161703 + Add field BUSNO: "KL11A0011" BusName: "Komban"	

Figure 7.5: Live Ticket

7.3 Integration Testing

Performed integration testing to ensure that the passenger side and conductor side work harmoniously together. Verified that data exchange between the two interfaces is accurate and real-time updates are reflected correctly.

7.4 System Testing

Conducted comprehensive system testing to evaluate the entire application's functionality, including interactions between the passenger and conductor sides. Ensure smooth communication, ticket generation, and data synchronization.

7.5 Cross-Device and Cross-Platform Testing

Tested the app on various devices, operating systems, and screen sizes to ensure a consistent and user-friendly experience across different platforms

7.6 Security Testing

Conducted security testing to identify potential vulnerabilities and ensure data encryption, secure payment processing, and protection of sensitive user information.

Chapter 8

Results



Figure 8.1: Splash screen

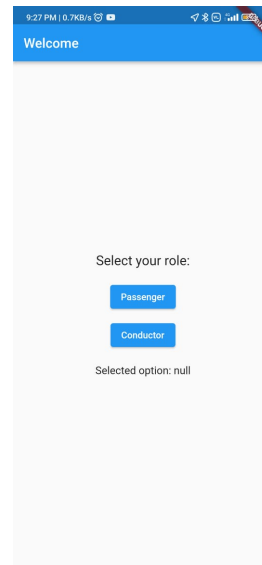


Figure 8.2: Welcome Screen

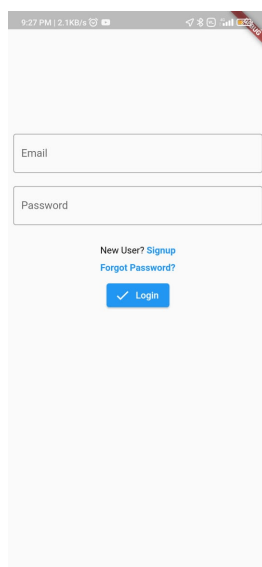


Figure 8.3: login and signup

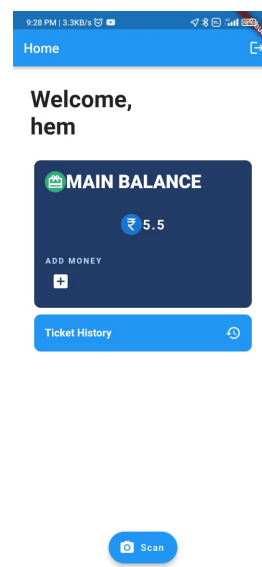


Figure 8.4: Passenger Home

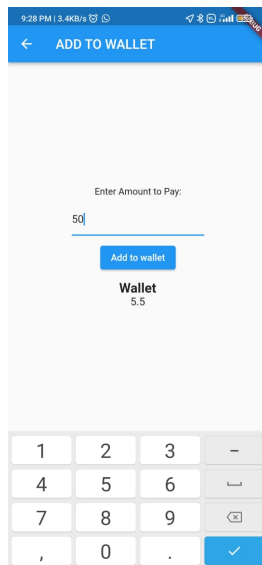


Figure 8.5: Wallet



Figure 8.6: QR Scanner

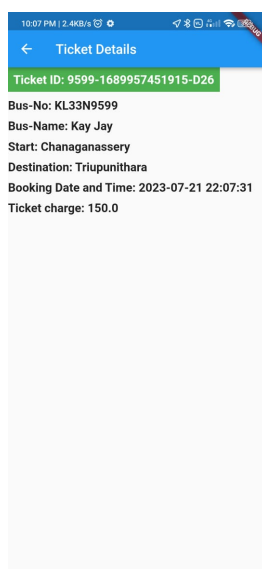


Figure 8.7: Ticket

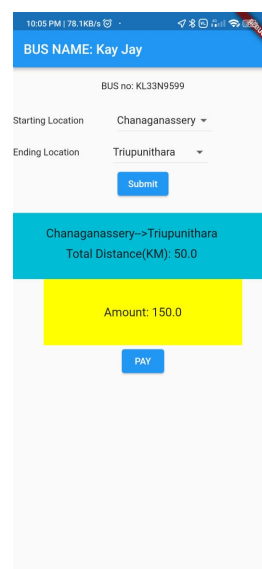


Figure 8.8: Price Calculation

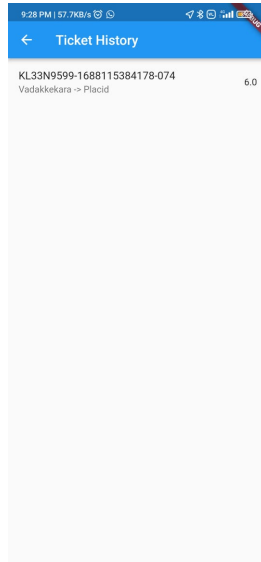


Figure 8.9: Ticket History

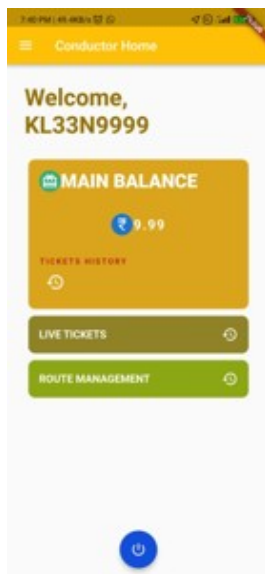


Figure 8.10: Conductor Home screen

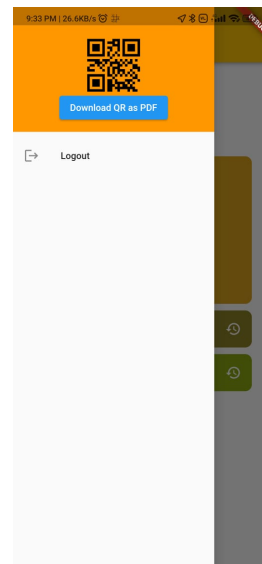


Figure 8.11: QR Generator

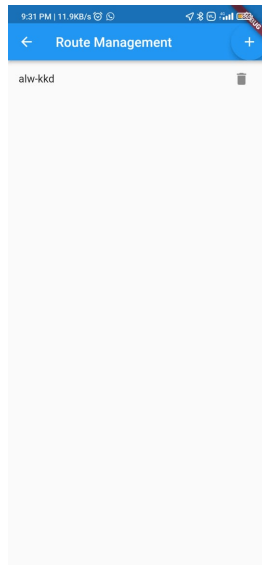


Figure 8.12: Route Management

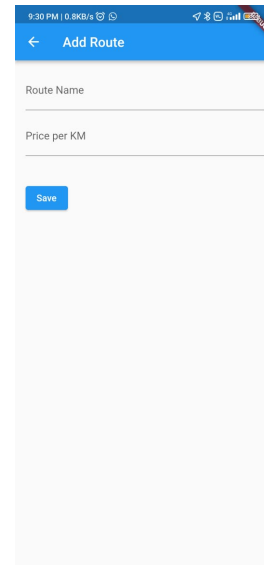


Figure 8.13: Add Route

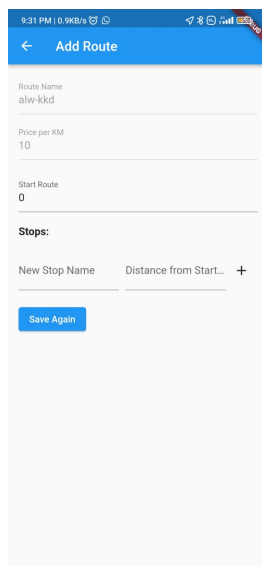


Figure 8.14: Route Details

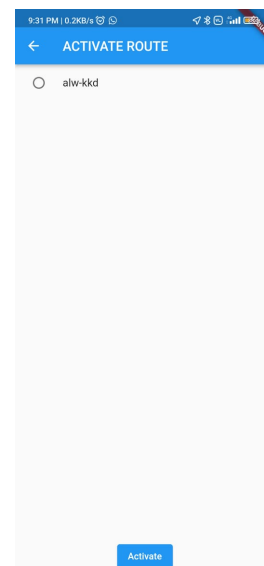


Figure 8.15: Route activating Screen

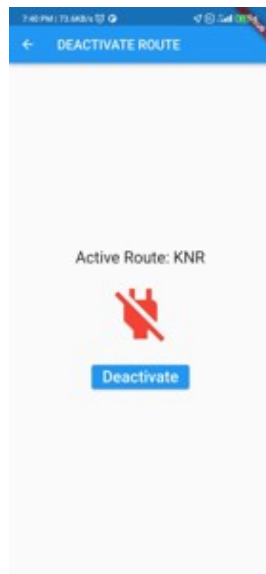


Figure 8.16: Deactivate Route screen

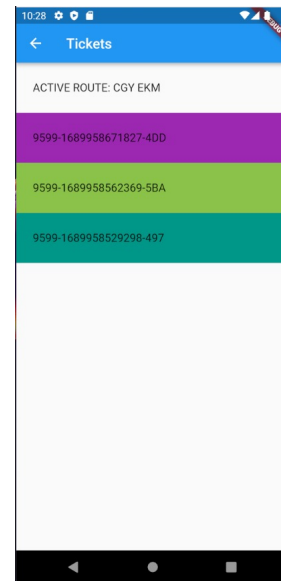


Figure 8.17: Live Tickets

PAYMENT HISTORY		
9599-1689761751265-F0E	petta -> vytila	15.0
9599-1689761721175-873	petta -> vytila	15.0
9599-1689761692588-0F5	petta -> vytila	15.0
9599-1689744999877-AB5	Chanaganassery -> Kurupunthura	90.0
9599-1689744984194-986	Chanaganassery -> Kottayam	60.0
9599-1689744966146-BDF	Chanaganassery -> Triupunithara	240.0
9599-1689744926764-1DE	Chanaganassery -> Enakulam	240.0

Figure 8.18: Payment History

Chapter 9

Risks and Challenges

1. Connectivity and Infrastructure: Reliance on stable internet connectivity and infrastructure to ensure app availability.
2. Operational Coordination: Coordinating with bus operators, , and accurate scheduling.
3. User Adoption: Passengers may resist the transition to an automated system, requiring clear instructions and user-friendly interfaces for smooth adoption.

Chapter 10

Conclusion

We have developed an android bus ticket automation app using Flutter and Firebase. The app features two interfaces, one for passengers and one for conductors. Passengers can scan the conductor-generated QR code inside the bus, select their desired locations, make payments, and generate tickets. The tickets are visible in real-time on the conductor's interface. This comprehensive solution improves efficiency, simplifies the ticketing process, and provides a seamless experience for both passengers and conductors. The project would be valuable to general society who are looking with the present issues. The Proposed framework would empower the general population to have the ticket for movement in metropolitan and for the most part open transport, in addition, the printed material will be decreased and losing of the card is additionally wiped out. It would guarantee the minimization of tedious and cash issues like change. Further, the subtle elements of the ticket of each traveller are put away in the database alongside conductor points of interest which are followed by the administrator.

Future scope

1. In this method we can give us a count of the number of passengers present in bus, which could be used further to estimate the capacity of the bus.
2. The program can be slightly modified to obtain safe travel of any transportation system such as Railways, school buses etc. More powerful algorithms can provide real time location information in internet, ensuring in time keeping of services.

References

- [1] Authors "Akshay Sonawane, Kushal Gogri, Ankeet Bhanushali, Milind Khairnar" have proposed "Real-Time Bus Tracking System" International Journal of Engineering Research and Technology (IJERT) Vol. 9 Issue 06, June-2020.
- [2] Albert Mayan J, Kuldeep Anand D.S and Sadhvi Neha 2020 International Conference on Information Communication and Embedded Systems (ICICES) (Chennai) Efficient and secure server migration on cloud storage with VSM and dropbox services 1-5
- [3] Hamilton P and Suresh S 2022 Intelligent Agent based RFID System for On Demand Bus Scheduling and Ticketing International Journal of Future Computer and Communication 2 399-406
- [4] Hasan Foisal Mahedi et al 2020 RFID-based Ticketing for Public Transport System : Perspective Megacity Dhaka 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT) 6 459-462
- [5] Xiao-Lei, M et al (2021). "Transit Smart Card Data Mining for Passenger Origin Information Extraction", I Journal of Zhenjiang University Science C, Vol. 13(10), pp.750-760
- [6] Authors "M. A. Hannan, A. M. Mustapha, A. Hussain and H. Basri" have implemented the system "Intelligent Bus Monitoring and Management System" Proceedings of the World Congress on Engineering and Computer Science 2020 Vol II WCECS Oct 2020.
- [7] Authors "Jivan Shelke, Aniket Mahangde, Sagar Karwa, Vishwajeet Mane" have implemented "Bus Pass Mobile Application Using QR Code" International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056 Volume: 05 Issue: 05 — May 2019

Appendix A: Base Paper

An Online Ticketing System using QR Code for Public Transport

Dr. Keshetti Sreekala^{1*}, R.Sai Kiran Reddy²

¹Assistant Professor, ²Student

Department of Computer Science & Engineering,
MGIT, Hyderabad, TS, India.

***Corresponding Author**

E-mail Id:-ksrikala_cse@mgit.ac.in

ABSTRACT

Today we have different tickets/passes for Bus, Metro and any other modes of public transport. It's been a problem to everyone every time to stand in queue for tickets/passes. Would it not be of great convenience to have single unified approach to ticketing from the place where you can book it comfortably? It will definitely save time and energy of human by just utilizing the technology properly. Now, we are developing an application which provides an interface between public transport users (PTU) and transport users (TU), where the PTU will register themselves with appropriate details about them and journey and get their validated ticket through QR code and on the other side by scanning the code, the admin can get details of journey and collect the fair respectively.

Keywords:-*Tickets, Public Transport Users (PTU), Transport Users (TU), QR code.*

INTRODUCTION

In Public transport system, the generation of ticket and issuing of ticket is a tedious work to the superior authorities. In places such as Bus Stand, Railway Station, a paper slip is issued to people by an official, which generally takes lots of time to receive by standing in queues. So, a secured electronic version of ticket in form of a QR Code can be used instead of traditional paper slips. A Quick Response Code (QR Code) is a two-dimensional barcode, which has fast readability and greater storage capacity. A QR Code can be read by an imaging device such as camera and then be processed.

This system simplifies the process of granting tickets to people in an easy way and eliminates paperwork. It also eliminates the workload and human effort by automatically displaying the ticket details for an employee and fair details for both user and employee in a transport in the webpage so that user can be easily

granted ticket and use the transport connecting different modes of usage of public transport on the basis of those factors. And once when the ticket is generated, a unique QR code is generated with user travelling details. The QR Code when scanned by the Authority at other point, displays the Source Point, Destination Point, No. Of Passengers and Fare. This QR Code can definitely be a secured way and easiest way for both users and authority to use transport system.

It is Client-Server application software. The main objective of this system is to enhance and upgrade the existing system by increasing its efficiency and effectiveness by reducing the manual work. The software improves the working methods by replacing the existing manual system with the Online-based system. This project is used to overcome the entire problem which they are facing currently, and making complete atomization of manual system to computerized system.

RELATED WORK

All researchers have aimed to develop and provide a generalized solution to monitor the various works that are carried out by public transport users for automation of various tasks. They provided up to date information of the system which improved efficiency of public transport record management.

In existing systems, in most of the public transport system a ticket generation and issuing is a step by step lengthy time taking process. Person has to approach authority like a Conductor or reach ticket counter for ticket. Then the conductor will issue ticket. If he loses a ticket then there is no other chance or backup for the ticket, if he is caught he must pay the fine. In this system, initially the user has to sign up and give the details. Then he must choose the source and destination point, he can also choose no. of passengers, then the fare regarding his travel is displayed and when he enters to generate the QR code a unique code is generated..A unique QR code is generated at every time of approval. This QR code is later scanned by authority or conductor and it is verified.

The following are some of the surveys done on the digital transport ticket systems:

In year 2013 Naveen Kumar, Sameer Ali [1] have proposed a smart transportation which is based on IOT in Social, Mobile, Analytics and Cloud which is user friendly and portable and its disadvantages are it

could not fetch GPS location and there is no recharge card.

In year 2014 Le Minh Kieu, Edward Chung [2] proposed passenger segmentation using smart card which provides Segment their customers and provide them with well-suited information and services but its disadvantage was providing data sets and spatial travel patterns.

In year 2016 Haotian Feng, Yongmei Huang [3] proposed Public Transport recharge using smart card which is hands free card wallet with high security using communication and control systems.

In year 2018 Paul-Antonin, Dublanche[4] proposed bus bunching identification using smart card which would increase the share rate of public transport using parallel and distribution systems. Its main disadvantage was it couldn't manage the bus arrival time prediction.

METHODOLOGY AND ARCHITECTURE

Architecture

Steps involved in building a popularity model recommendation system are as follows:

1. Login page
2. Selection page
3. Ticket Selection
4. QR code generator
5. QR code validation
6. User history

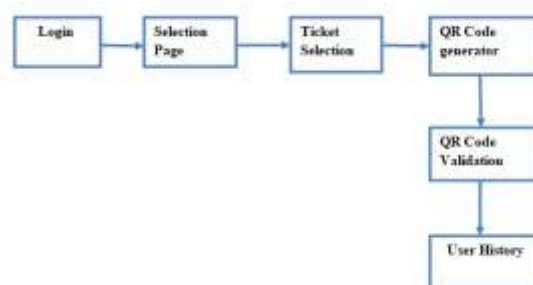


Fig.1:-System Architecture of Online Ticketing System

Android Studio IDE

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on Jet Brains Intel J IDEA software and designed specifically for Android development [4]. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application

development.

Software Architecture

In the below Figure 2, the administrator can create an app by entering into android studio and write code and complete the respective UI design then manage compiling by gradle build and then connect to Android Virtual Device(AVD) through emulator or can connect to device directly.

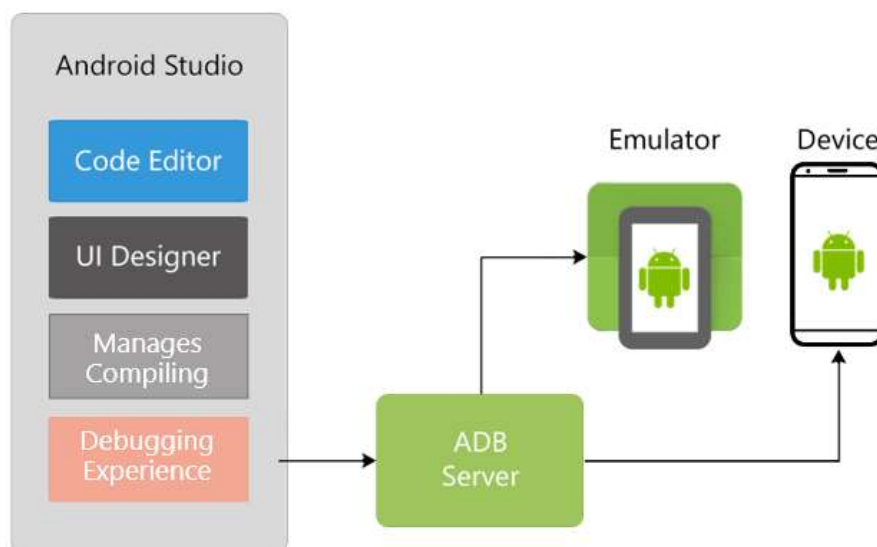


Fig.2:-Software Architecture of Android Studio system

Firebase Server

This is the main database for the application which helps us maintain every single action that is performed in the application.

These may include the unregistered user signing up to the application and all the details getting stored up in the database.

Then the database when it creates the user also tracks all the actions performed by the registered user, like the creation of new groups or chat and updating of the current registered user's details like name, password, tags, organization, etc.

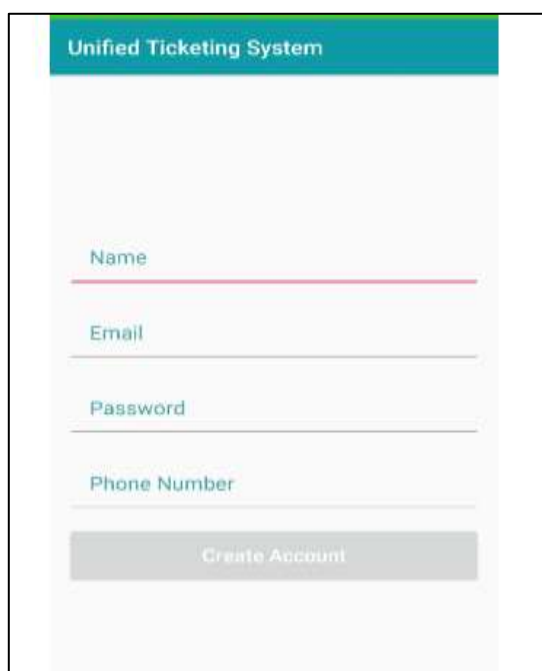
Every single time the user logs in or log's

out, the database marks the action and the status of each registered user in the user's portion of the database. Firebase Cloud Messaging (FCM) is a cross-platform solution for messages and notifications for Android, iOS, and web applications.

RESULT

Create Account Page

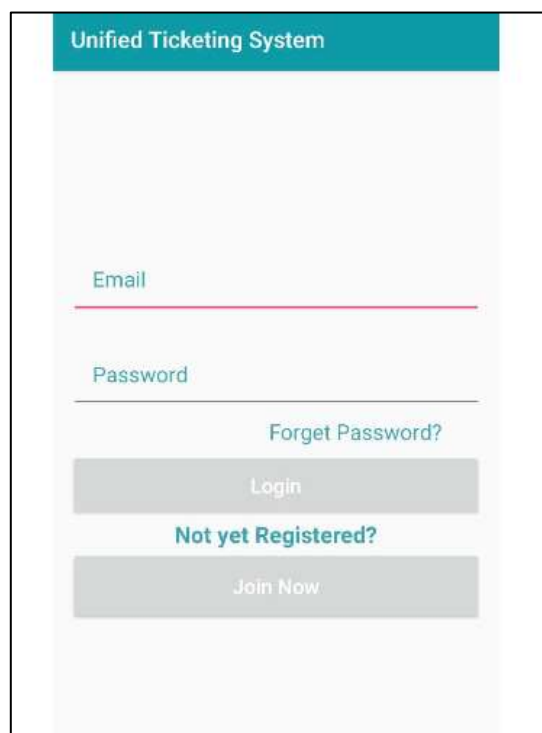
In Figure 3: when the user first enter the application he should give the details contains of his name, email, password and phone number and then click on create account button then the account is created. It accepts the user_id as input and password with several specifications is accepted and enters the pages. The account is created.



The image shows a mobile application interface for a 'Unified Ticketing System'. At the top, there is a teal header bar with the text 'Unified Ticketing System' in white. Below the header, the background is a light gray. The form consists of four input fields, each with a label above it: 'Name', 'Email', 'Password', and 'Phone Number'. Each input field has a thin pink underline. Below these fields is a wide, light gray button with the text 'Create Account' in a small, dark font.

Fig.3:-Create Account for Unified Ticketing System

Login Page



The image shows a mobile application interface for a 'Unified Ticketing System'. At the top, there is a teal header bar with the text 'Unified Ticketing System' in white. Below the header, the background is a light gray. The form consists of two input fields, each with a label above it: 'Email' and 'Password'. Each input field has a thin pink underline. Below the 'Password' field is a link labeled 'Forget Password?' in a small, teal font. Below this link is a wide, light gray button with the text 'Login' in a small, dark font. Below the 'Login' button is a link labeled 'Not yet Registered?' in a small, teal font. Below this link is another wide, light gray button with the text 'Join Now' in a small, dark font.

Fig.4:-Login page for Unified Ticketing System

In the above Figure 4, the user after creating the account in the above page, now the user is ready to get the ticket process.

The user has to give the correct email and

password and then login to application for further process.

If the given email and password does not match then he cannot enter into further application process.

Ticket Selection Page

Fig.5:-Selection page for unified ticketing system

In Figure 5, the user after login into the application views the above page, in this the user will select the source and destination point for travel. The user can also select the no. of passengers for generating the ticket fair.

Unified Ticketing System Page

In the below Figure 6, the user is given different modes of travel using public

transport,

(i): User can use bus from certain source to destination and can use other public transport like metro, corresponding fair is mentioned.

(ii): User can only use bus from source to destination.

Here, the respective fair for each mode of public transport is displayed and QR code is generated.

Fig.6:-Unified ticketing system page

Generation of QR code



Fig.7:-Generation of QR code page

In the above figure 7, the user after selecting the path is ready to generate the QR code, after checking the details when

the user click on generate button the respective QR code is generated.

Users History Page



Fig.8:-Users History page

In the above Figure 8, the admin after scanning the tickets can view the history of

no. of tickets generated in the history section.

DESIGN OF THE ONLINE TICKETING SYSTEM USING QR CODE GENERATION

Use Case Diagram

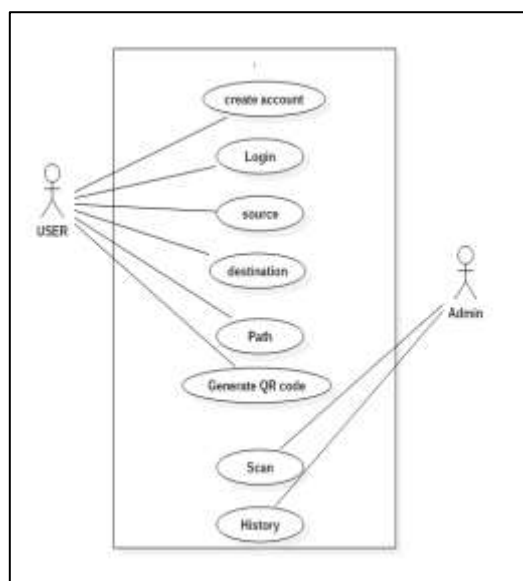


Fig.9:-Use Case Diagram

In the above Figure 9, the use case diagram consists of two actors.

They are User and Admin. The user is associated to all use cases and QR code is generated.

The various use cases are as follows:-

1. Create: The user can create an account.
2. Login: The user can login using the given details.
3. Source: The user can select the source point for travelling in public transport.
4. Destination: The user can select the destination point
5. Path: The user can select the path using public transport.
6. Generate QR Code: If all the path is set, QR code can be generated.
7. Scan QR Code: The Admin at the other point scans the QR Code and allows user.
8. History: The Admin can view the history of tickets he scanned.

Class Diagram

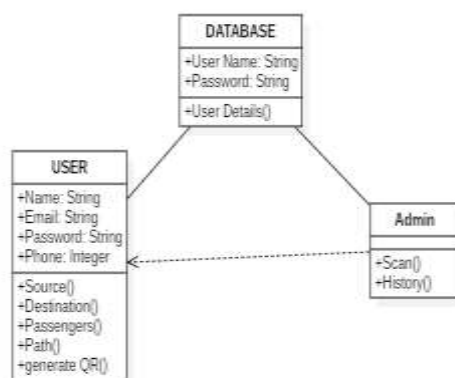


Fig.10:-Class Diagram

In the above Figure 10, the class diagram consists of three classes namely USER, DATABASE, ADMIN.

The attributes and operations of each class are :

1 Admin:-

Operations: Scan(), History()

2 User:-

Attributes: Name, Email, Password, Phone
Operations: Source(), Destination(), Passengers, Path(), QR code()

3 Data Base:-

Attributes: User Name, Password

Operations: User Details

Activity Diagram

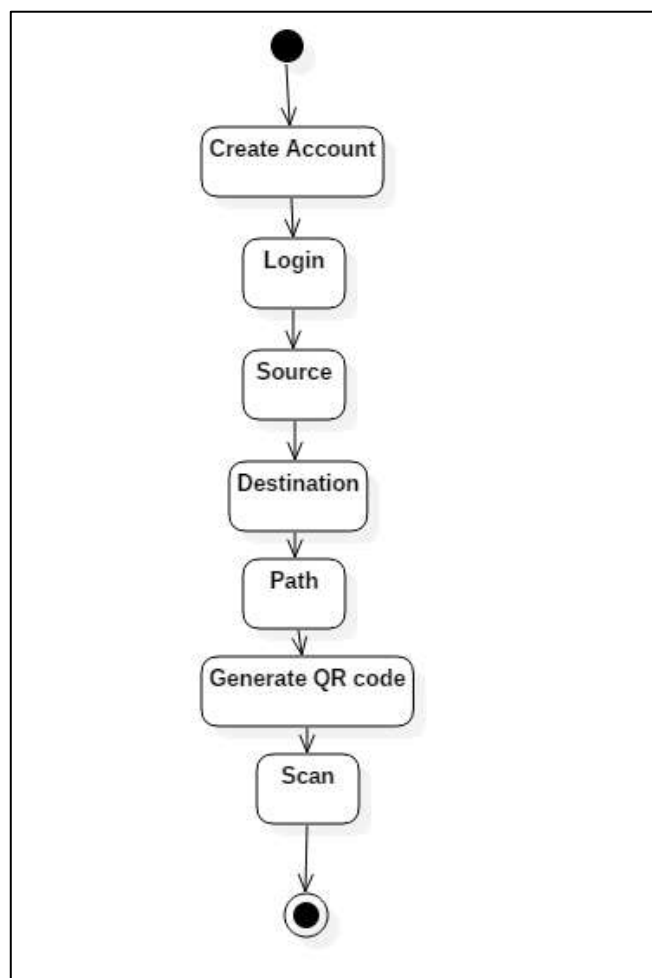


Fig.11:-Activity Diagram

In the above Figure 11, the activity diagram represents workflows in a graphical way. This diagram below describes the business overflow on the operational workflow of any component in a system.

The activity states in this diagram are:-

1. Create Account

2. Login

3. Source

4. Destination

5. Path

6. Generate QR Code

7. Scan the QR Code

Sequence Diagram

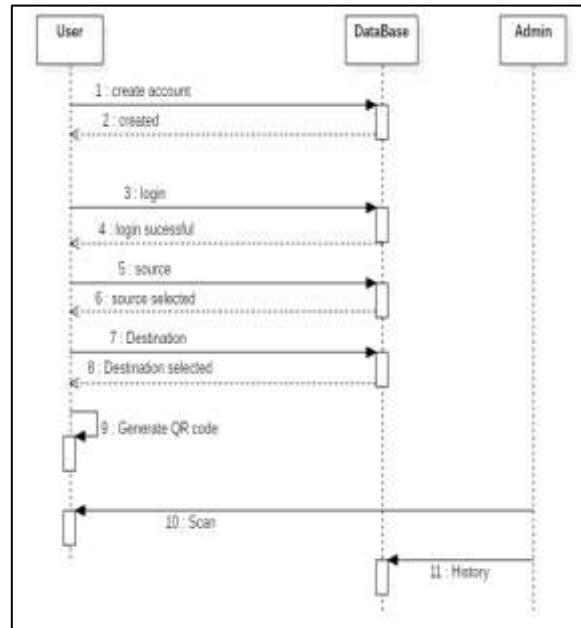


Fig.12:-Sequence Diagram

In the above Figure 12, the sequence diagram defines the flow of operations. The below sequence diagram is about student activities and operation.

The behavioral execution states are:

1. Create account
2. Created
3. Login
4. Login Successful
5. Source
6. Source selected
7. Destination
8. Destination selected
9. Generate QR code
10. Scan
11. History

CONCLUSION

The ultimate goal of this project is to increase the convenience level of people to have single unified approach to ticketing and payment. This project enables secured, fast and easy ticket generation and approval. This project reduces the overhead involved from passenger side as well as employee side and also eliminates paper work.

With the help of QR Code the generation

of ticket is very easy and then validating the ticket is also easy which makes simple for people. The project is very simple and easy to maintain and use. The installation of the project is simple and does not require any professionals for implementation or handling. It gives reliable and efficient way of implementing the unified ticketing system. Hence, we can conclude that this system is user-friendly and intuitive and would prove to be an efficient system to obtain a ticket in a very convenient way.

REFERENCES

1. <https://www.techspot.com/guides/1676-qr-code-explained>
2. <https://www.qr-code-generator.com/qr-code-marketing/qr-codes-basics>
3. <https://www.affordablewebdesign.com/benefits-of-using-qr-codes>
4. https://en.wikipedia.org/wiki/Android_Studio
5. Du, B., & Dublanche, P. A. (2018, December). Bus bunching identification using smart card data. In *2018 IEEE 24th International Conference on Parallel and*

- Distributed Systems (ICPADS)* (pp. 1087-1092). IEEE.
6. Feng, H., Huang, Y., & Xu, M. (2016, July). Public Transport Recharge System Based on Smart Phone. In *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)* (pp. 884-887). IEEE.
7. Bhaskar, A., & Chung, E. (2014). Passenger segmentation using smart card data. *IEEE Transactions on intelligent transportation systems*, *16*(3), 1537-1548.
8. Mrityunjaya, D. H., Kumar, N., Ali, S., & Kelagadi, H. M. (2017, February). Smart transportation. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)* (pp. 1-5). IEEE.

Appendix B: Sample Code

FULL PROJECT - https://github.com/martin-francs/bus_t1.git

SAMPLE CODE

main.dart

```
import 'package:bus_t/screens/loginsignup/splash.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
//import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_core/firebase_core.dart';
String userId='9567867353';
void main(List<String> args) async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const Bust());
}
```

```
class Bust
extends StatelessWidget {
  const Bust
  ({super.key});

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      theme: ThemeData(primaryColor: Colors.indigo),
      home: const SplashScreen(),
    );
  }
}
```

Passenger- home.dart

```
import 'package:bus_t/screens/passsenger/walletaddmoney.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:qrscan/qrscan.dart' as scanner;
import 'package:cloud_firestore/cloud_firestore.dart';
import '../loginsignup/welcome.dart';
import 'busroute.dart';
import 'ticketshistory.dart';
import 'package:shared_preferences/shared_preferences.dart';

class Homescreen extends StatefulWidget {
  const Homescreen({Key? key}) : super(key: key);
```

```

@override
_HomescreenState createState() => _HomescreenState();

Future<String?> _getDocumentIdFromSharedPreferences() async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
  return prefs.getString('mobileNumber');
}
}

class _HomescreenState extends State<Homescreen> {
  String fname = "";
  String result = "Hello World...!";
  late String documentId;

  @override
  void initState() {
    super.initState();
    _retrieveDocumentId();
  }

  Future<void> _retrieveDocumentId() async {
    String? mobileNumber = await widget._getDocumentIdFromSharedPreferences();
    if (mobileNumber != null) {
      setState(() {
        documentId = mobileNumber;
      });
      FirebaseFirestore.instance
        .collection('users')
        .doc(documentId)
        .get()
        .then((snapshot) {
          if (snapshot.exists) {
            setState(() {
              fname = snapshot.data()?['Name'] ?? "";
            });
          }
        });
    }
  }

  @override
  Widget build(BuildContext context) {

```

```

return Scaffold(
  appBar: AppBar(
    title: const Text('Home'),
    actions: [
      IconButton(
        icon: const Icon(Icons.logout),
        onPressed: () {
          signout(context);
        },
      ),
    ],
  ),
  backgroundColor: Colors.white,
  body: SafeArea(
    child: Column(
      children: [
        const SizedBox(
          height: 30,
        ),
        SizedBox(
          width: double.infinity,
          child: Text(
            ' Welcome,\n $fname',
            style: const TextStyle(fontSize: 35, fontWeight: FontWeight.bold),
            textAlign: TextAlign.left,
          ),
        ),
        const SizedBox(
          height: 30,
        ),
        StreamBuilder<DocumentSnapshot>(
          stream: FirebaseFirestore.instance
            .collection('users')
            .doc(documentId)
            .snapshots(),
          builder: (context, snapshot) {
            if (snapshot.hasError) {
              return const Text('Error fetching balance');
            }

            if (snapshot.connectionState == ConnectionState.waiting) {
              return const CircularProgressIndicator();
            }
          }
        )
      ],
    ),
  ),
);

```

```
double balance = snapshot.data?['walletAmount'] ?? 0.0;
```

```
return Container(  
  margin: const EdgeInsets.symmetric(horizontal: 32),  
  decoration: const BoxDecoration(  
    borderRadius: BorderRadius.all(Radius.circular(10)),  
    color: Color.fromRGBO(35, 60, 103, 1),  
  ),  
  padding: const EdgeInsets.all(16),  
  child: Column(  
    crossAxisAlignment: CrossAxisAlignment.start,  
    children: <Widget>[  
      const Row(  
        children: <Widget>[  
          CircleAvatar(  
            radius: 16,  
            backgroundColor: Color.fromRGBO(50, 172, 121, 1),  
            child: Icon(  
              Icons.wallet_giftcard_sharp,  
              color: Colors.white,  
              size: 24,  
            ),  
          ),  
          Text(  
            "MAIN BALANCE",  
            style: TextStyle(  
              fontStyle: FontStyle.normal,  
              fontSize: 28,  
              color: Colors.white,  
              fontWeight: FontWeight.w900),  
          )  
        ],  
      ),  
      const SizedBox(  
        height: 32,  
      ),  
      Row(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: [  
          const CircleAvatar(  
            radius: 16,  
            child: Icon(  
              Icons.currency_rupee,  
              color: Colors.white,
```

```

    ),
  ),
  Text(
    balance.toString(),
    style: const TextStyle(
      fontSize: 20,
      color: Colors.white,
      fontWeight: FontWeight.w700,
      letterSpacing: 2.0),
  ),
],
),
const SizedBox(
  height: 32,
),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text(
          "ADD MONEY",
          style: TextStyle(
            fontSize: 12,
            color: Colors.blue[100],
            fontWeight: FontWeight.w700,
            letterSpacing: 2.0),
        ),
        IconButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => addmoney(documentId: documentId),
              ),
            );
          },
          icon: const Icon(Icons.add_box),
          iconSize: 28,
          color: Colors.white,
        ),
      ],
    ),
  ],
),

```

```

        ],
      ),
    ],
  ),
);
},
),
const SizedBox(height: 10),
GestureDetector(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => TicketHistoryScreen(
          documentId: documentId,
        ),
      ),
    );
  },
  child: Container(
    margin: const EdgeInsets.symmetric(horizontal: 32),
    decoration: const BoxDecoration(
      borderRadius: BorderRadius.all(Radius.circular(10)),
      color: Colors.blue, // Replace with your desired color
    ),
    padding: const EdgeInsets.all(16),
    child: const Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: <Widget>[
        Text(
          "Ticket History",
          style: TextStyle(
            fontSize: 16,
            color: Colors.white,
            fontWeight: FontWeight.w700,
          ),
        ),
        Icon(
          Icons.history_outlined,
          color: Colors.white,
        ),
      ],
    ),
  ),
);

```

```

        ),
    ],
),
floatingActionButton: FloatingActionButton.extended(
    icon: const Icon(Icons.camera_alt),
    onPressed: () {
        _scanQR();
    },
    label: const Text("Scan"),
),
floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
);
}

```

```

Future<void> _scanQR() async {
    try {
        var cameraStatus = await Permission.camera.status;
        if (cameraStatus.isGranted) {
            String? cameraScanResult = await scanner.scan();
            setState(() {
                result = cameraScanResult!;
                print(result);
                Navigator.of(context).push(
                    MaterialPageRoute(
                        builder: (ctx) {
                            return Busroutescreen(documentId: result,userId: documentId);
                        },
                    ),
                );
            });
        } else {
            var isGranted = await Permission.camera.request();
            if (isGranted.isGranted) {
                String? cameraScanResult = await scanner.scan();
                setState(() {
                    result = cameraScanResult!;
                    Navigator.of(context).push(
                        MaterialPageRoute(
                            builder: (ctx) {
                                return Busroutescreen(documentId: result,userId: documentId,);
                            },
                        ),
                    );
                });
            }
        }
    } catch (e) {
        print(e);
    }
}

```

```

    });
  }
}
} on PlatformException catch (e) {
  print(e);
}
}

```

```

signin(BuildContext ctx) async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.clear();
  Navigator.of(ctx).pushAndRemoveUntil(
    MaterialPageRoute(builder: (ctx1) => const WelcomeScreen()),
    (route) => false,
  );
}
}

```

Passenger walletaddmoney.dart

```

import 'package:flutter/material.dart';
import 'package:razorpay_flutter/razorpay_flutter.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

```

```

class addmoney extends StatefulWidget {
  final String documentId;

  const addmoney({super.key, required this.documentId});
  @override
  _addmoneyState createState() => _addmoneyState();
}

```

```

class _addmoneyState extends State<addmoney> {
  final TextEditingController _amountController = TextEditingController();
  final String _displayedAmount = "";
  Razorpay? _razorpay;

  void _handlePaymentSuccess(PaymentSuccessResponse response) {
    String? paymentId = response.paymentId;
    double paidAmount = double.tryParse(_amountController.text) ?? 0.0;

    // Update Firestore document
    FirebaseFirestore.instance
      .collection('users')
      .doc(widget.documentId)
      .update({'walletAmount': FieldValue.increment(paidAmount)})
  }
}

```



```

        .then((value) {
        // Firestore update successful
        Fluttertoast.showToast(
            msg: "SUCCESS PAYMENT: $paymentId", timeInSecForlosWeb: 4);
    }).catchError((error) {
        // Firestore update failed
        Fluttertoast.showToast(
            msg: "Error updating wallet amount", timeInSecForlosWeb: 4);
    });
}

void _handlePaymentFailure(PaymentFailureResponse response) {
    Fluttertoast.showToast(
        msg: "ERROR: ${response.code}", timeInSecForlosWeb: 4);
}

void _handleExternalWallet(ExternalWalletResponse response) {
    Fluttertoast.showToast(
        msg: "External wallet is: ${response.walletName}", timeInSecForlosWeb: 4);
}

void makePayment() async {
    double amount = double.tryParse(_amountController.text) ?? 0.0;

    if (amount <= 0) {
        Fluttertoast.showToast(
            msg: "Please enter a valid amount", timeInSecForlosWeb: 4);
        return;
    }

    var options = {
        'key': 'rzp_test_EtRIJbLp1Snpr',
        'amount': (amount * 100).toInt(),
        'name': "Martin",
        'description': 'hello test',
        'prefill': {
            'Contact': "+919567867353",
            'email': "francismartin0078@gmail.com"
        }
    };

    try {
        _razorpay?.open(options);
    } catch (e) {
        debugPrint(e.toString());
    }
}

```

```
}
```

```
@override
void initState() {
  super.initState();
  _razorpay = Razorpay();
  _razorpay?.on(Razorpay.EVENT_PAYMENT_SUCCESS, _handlePaymentSuccess);
  _razorpay?.on(Razorpay.EVENT_PAYMENT_ERROR, _handlePaymentFailure);
  _razorpay?.on(Razorpay.EVENT_EXTERNAL_WALLET, _handleExternalWallet);
}
```

```
@override
void dispose() {
  _razorpay?.clear();
  super.dispose();
}
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('ADD TO WALLET'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const Text(
            'Enter Amount to Pay:',
          ),
          const SizedBox(height: 10),
          SizedBox(
            width: 200,
            child: TextField(
              controller: _amountController,
              keyboardType: TextInputType.number,
              decoration: const InputDecoration(
                hintText: '0.00',
              ),
            ),
          ),
          const SizedBox(height: 10),
          ElevatedButton(
            onPressed: makePayment,
            child: const Text('Add to wallet'),
          ),
        ],
      ),
    ),
  );
}
```

```

),
const SizedBox(height: 10),
StreamBuilder<DocumentSnapshot>(
  stream: FirebaseFirestore.instance
    .collection('users')
    .doc(widget.documentId)
    .snapshots(),
  builder: (context, snapshot) {
    if (snapshot.hasError) {
      return const Text('Error fetching wallet amount');
    }

    if (snapshot.connectionState == ConnectionState.waiting) {
      return const CircularProgressIndicator();
    }

    double walletAmount = snapshot.data?['walletAmount'] ?? 0.0;

    return Column(
      children: [
        const Text(
          'Wallet',
          style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
          ),
        ),
        Text(
          walletAmount.toString(),
          style: const TextStyle(
            fontSize: 16,
          ),
        ),
      ],
    );
  },
),
],
),
),
);
}
}

```

CONDUCTOR- c_home.dart

```
import 'dart:io';
```

```

import 'package:bus_t/screens/conductor/active_off.dart';
import 'package:bus_t/screens/conductor/active_on.dart';
import 'package:bus_t/screens/conductor/paymenthistory.dart';
import 'package:bus_t/screens/conductor/routes_management.dart';
import 'package:bus_t/screens/loginsignup/welcome.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:pdf/widgets.dart' as pw;
import 'package:open_file/open_file.dart';
import 'package:http/http.dart' as http;
import 'package:firebase_storage/firebase_storage.dart' as firebase_storage;
import 'package:path_provider/path_provider.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'livetickets.dart';

```

```

Future<List<String>> fetchRouteIds(String? busno) async {
  List<String> routeIds = [];
  if (busno != null) {
    QuerySnapshot snapshot = await FirebaseFirestore.instance
      .collection('buses')
      .doc(busno)
      .collection('routes')
      .get();
    for (var doc in snapshot.docs) {
      routeIds.add(doc.id);
    }
  }
  return routeIds;
}

class ConductorHomePage extends StatefulWidget {
  final String busID;

  const ConductorHomePage({super.key, required this.busID});
  Future<String?> _getDocumentIdFromSharedPreferences() async {
    final SharedPreferences prefs = await SharedPreferences.getInstance();
    return prefs.getString('busno');
  }
  @override
  _ConductorHomePageState createState() => _ConductorHomePageState();
}

class _ConductorHomePageState extends State<ConductorHomePage> {

```

```

String qrCodeImageUrl = "";
String? documentId;
bool active = false;
String? collectionName;
@override
void initState() {
  super.initState();
  loadQRCodeImage();
  _loadSharedPrefs(); // Load the QR code image when the home page is initialized
}
_loadSharedPrefs() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  setState(() {
    documentId = prefs.getString('busno');
    collectionName = prefs.getString('time');
    active = prefs.getBool('active')!;
  });
}

Future<void> loadQRCodeImage() async {
  try {
    // Retrieve the QR code image URL from Firebase Storage
    final storageReference = firebase_storage.FirebaseStorage.instance
      .ref()
      .child('qr_codes')
      .child('${widget.busID}.png');
    final downloadURL = await storageReference.getDownloadURL();

    // Update the state with the QR code image URL
    setState(() {
      qrCodeImageUrl = downloadURL;
    });
  } catch (error) {
    print('Error loading QR code image: $error');
  }
}

void downloadQRCodeAsPDF() async {
  try {
    // Retrieve the QR code image bytes from the network
    final response = await http.get(Uri.parse(qrCodeImageUrl));
    final qrCodeImageBytes = response.bodyBytes;

    // Create a PDF document
    final pdf = pw.Document();
  }
}

```

```

// Add a page to the PDF document
pdf.addPage(
  pw.Page(
    build: (pw.Context context) {
      return pw.Image(
        pw.MemoryImage(qrCodeImageBytes),
        fit: pw.BoxFit.contain,
      );
    },
  ),
);

// Get the temporary directory path
final tempDir = await getTemporaryDirectory();
final tempPath = tempDir.path;

// Save the PDF document to a temporary file
final filePath = File('$tempPath/${widget.busID}_qr_code.pdf');
await filePath.writeAsBytes(await pdf.save());

// Open the PDF file with the default PDF viewer
OpenFile.open(filePath.path);
} catch (error) {
  print('Error downloading QR code as PDF: $error');
}
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.amber,
      title: const Text('Conductor Home'),
    ),
    drawer: Drawer(
      child: ListView(
        padding: EdgeInsets.zero,
        children: [
          DrawerHeader(
            decoration: const BoxDecoration(
              color: Colors.orange,
            ),
            child: Align(
              alignment: Alignment.center,
              child: Column(

```

```

crossAxisAlignment: CrossAxisAlignment.center,
mainAxisAlignment: MainAxisAlignment.center,
children: [
  if (qrCodeImageUrl.isNotEmpty)
    Image.network(
      qrCodeImageUrl,
      width: 200, // Adjust the width to make the image larger
      height: 80, // Adjust the height to make the image larger
      fit: BoxFit.contain,
    ),
    ElevatedButton(
      onPressed: downloadQRCodeAsPDF,
      child: const Text('Download QR as PDF'),
    ),
  ],
),
),
),
),
ListTile(
  leading: const Icon(Icons.logout),
  title: const Text('Logout'),
  onTap: () {
    signout1(context);
  },
),
],
),
),
body: SafeArea(

  child: Column(
    // mainAxisAlignment: MainAxisAlignment.center,
    children: [
      const SizedBox(
        height: 30,
      ),
      SizedBox(
        width: double.infinity,
        child: Text(
          ' Welcome,\n $documentId',
          style: const TextStyle(fontSize: 35, fontWeight: FontWeight.bold,color:
Color.fromARGB(255, 148, 111, 0)),
          textAlign: TextAlign.left,
        ),
      ),
      const SizedBox(

```

```
height: 30,  
)
```

```
StreamBuilder<DocumentSnapshot>(
  stream: FirebaseFirestore.instance
    .collection('conductors')
    .doc(documentId)
    .snapshots(),
  builder: (context, snapshot) {
    if (snapshot.hasError) {
      return const Text('Error fetching balance');
    }

    if (snapshot.connectionState == ConnectionState.waiting) {
      return const CircularProgressIndicator();
    }

    double balance = snapshot.data?['walletAmount'] ?? 0.0;

    return Container(
      margin: const EdgeInsets.symmetric(horizontal: 32),
      decoration: const BoxDecoration(
        borderRadius: BorderRadius.all(Radius.circular(10)),
        color: Color.fromRGBO(218, 164, 28, 1),
      ),
      padding: const EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          const Row(
            children: <Widget>[
              CircleAvatar(
                radius: 16,
                backgroundColor: Color.fromRGBO(50, 172, 121, 1),
                child: Icon(
                  Icons.wallet_giftcard_sharp,
                  color: Colors.white,
                  size: 24,
                ),
              ),
              Text(
                "MAIN BALANCE",
                style: TextStyle(
                  fontStyle: FontStyle.normal,
                  fontSize: 28,
                  color: Colors.white,
```



```

        fontWeight: FontWeight.w900),
    ),
  ],
),
const SizedBox(
  height: 32,
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const CircleAvatar(
      radius: 16,
      child: Icon(
        Icons.currency_rupee,
        color: Colors.white,
      ),
    ),
    Text(
      balance.toString(),
      style: const TextStyle(
        fontSize: 20,
        color: Colors.white,
        fontWeight: FontWeight.w700,
        letterSpacing: 2.0),
    ),
  ],
),
const SizedBox(
  height: 32,
),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text(
          "TICKETS HISTORY",
          style: TextStyle(
            fontSize: 12,
            color: Color.fromARGB(255, 182, 34, 11),
            fontWeight: FontWeight.w700,
            letterSpacing: 2.0),
        ),
        IconButton(
          onPressed: () {

```

```

        Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => paymentHistoryScreen(documentId:widget.busID),
      ),
    );

    },
    icon: const Icon(Icons.history),
    iconSize: 28,
    color: Colors.white,
  ),
],
),
],
),
],
),
);
},
),
const SizedBox(height: 10),
GestureDetector(
  onTap: () {
    _loadSharedPrefs();
    // print(documentId);
    // print(collectionName);
    if (active!=false) {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => TicketScreen(
            documentId: documentId!,
            collectionName: collectionName!,
          ),
        ),
      );
    } else {
      // Handle case where shared preferences are not set
      showDialog(
        context: context,
        builder: (context) => AlertDialog(
          title: Text('Error'),
          content: Text('SET A ROUTE ACTIVE BY CLICKING POWERON ICON .'),
          actions: [
            ElevatedButton(
              onPressed: () {

```

```

        Navigator.pop(context);
      },
      child: Text('OK'),
    ),
  ],
),
);
}
},
child: Container(
  margin: const EdgeInsets.symmetric(horizontal: 32),
  decoration: const BoxDecoration(
    borderRadius: BorderRadius.all(Radius.circular(10)),
    color: Color.fromARGB(255, 144, 130, 39), // Replace with your desired color
  ),
  padding: const EdgeInsets.all(16),
  child: const Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: <Widget>[
      Text(
        "LIVE TICKETS",
        style: TextStyle(
          fontSize: 16,
          color: Colors.white,
          fontWeight: FontWeight.w700,
        ),
      ),
      Icon(
        Icons.history_outlined,
        color: Colors.white,
      ),
    ],
  ),
),
const SizedBox(height: 10),
GestureDetector(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => RouteManagement(busID: widget.busID,),
      ),
    );
  },
  child: Container(

```

```
margin: const EdgeInsets.symmetric(horizontal: 32),
decoration: const BoxDecoration(
  borderRadius: BorderRadius.all(Radius.circular(10)),
  color: Color.fromARGB(255, 139, 167, 19), // Replace with your desired color
),
padding: const EdgeInsets.all(16),
child: const Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    Text(
      "ROUTE MANAGEMENT",
      style: TextStyle(
        fontSize: 16,
        color: Colors.white,
        fontWeight: FontWeight.w700,
      ),
    ),
    Icon(
      Icons.history_outlined,
      color: Colors.white,
    ),
  ],
),
),
),
],
),
),
floatingActionButton: FloatingActionButton(
  onPressed: () async {
    String? activeRoute =
      await SharedPreferences.getInstance().then((prefs) {
        return prefs.getString('activeRoute');
      });

    if (activeRoute != "") {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) =>
            const activeoffscreen(),
        ),
      );
    } else {
      String? busno = await widget._getDocumentIdFromSharedPreferences();
      List<String> routelds = await fetchRoutelds(busno);
```

```

Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) =>
      activeonscreen(busno: busno, routelds: routelds),
  ),
);
},
},
backgroundColor: const Color.fromARGB(255, 18, 78, 217),
child: const Icon(Icons.power_settings_new_sharp),
),
floatingActionButtonLocation: FloatingActionButtonLocation.centerFloat,
);
}
signout1(BuildContext ctx) async {
  final SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.clear();
  Navigator.of(ctx).pushAndRemoveUntil(
    MaterialPageRoute(builder: (ctx1) => const WelcomeScreen()),
    (route) => false,
  );
}
}

```

CONDUCTOR - **livetickets.dart**

```

import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

```

```

class Ticket {
  final String id;
  final int color; // Add a color field

```

```

  Ticket({
    required this.id,
    required this.color,
  });

```

```

factory Ticket.fromFirestore(DocumentSnapshot doc) {
  Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
  return Ticket(
    id: doc.id,
    color: data['color'] ?? Color.fromARGB(0, 255, 255, 255).value, // Fetch the color value from
Firestore
  );
}
}

```

```

class TicketService {
  final String documentId;
  final String collectionName;

  TicketService({required this.documentId, required this.collectionName});

  Stream<List<Ticket>> getTicketsStream() {
    return FirebaseFirestore.instance
      .collection('tickets')
      .doc(documentId)
      .collection(collectionName)
      .snapshots()
      .map((snapshot) => snapshot.docs.map((doc) => Ticket.fromFirestore(doc)).toList());
  }
}

```

```

class TicketScreen extends StatelessWidget {
  final String documentId;
  final String collectionName;

  TicketScreen({required this.documentId, required this.collectionName});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Tickets'),
      ),
      body: StreamBuilder<List<Ticket>>(
        stream: TicketService(documentId: documentId, collectionName:
collectionName).getTicketsStream(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error fetching tickets'));
          } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
            return Center(child: Text('No tickets found'));
          } else {
            return ListView.builder(
              itemCount: snapshot.data!.length,
              itemBuilder: (context, index) {
                var ticket = snapshot.data![snapshot.data!.length - index - 1];
                return Container(
                  color: Color(ticket.color), // Use the color value fetched from Firestore

```

```
padding: const EdgeInsets.all(8.0),
child: ListTile(
  title: Text(ticket.id),
),
);
},
);
}
},
),
);
}
}
```

Appendix C: CO-PO And CO-PSO Mapping

COURSE OUTCOMES:

After completion of the course the student will be able to

SL. NO	DESCRIPTION	Blooms' Taxonomy Level
CO1	Identify technically and economically feasible problems (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO2	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO3	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions of minimal complexity by using modern tools & advanced programming techniques (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO4	Prepare technical report and deliver presentation (Cognitive Knowledge Level: Apply)	Level 3: Apply
CO5	Apply engineering and management principles to achieve the goal of the project (Cognitive Knowledge Level: Apply)	Level 3: Apply

CO-PO AND CO-PSO MAPPING

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	3	3	3	3		2	2	3	2	2	2	3	2	2	2
CO2	3	3	3	3	3	2		3	2	3	2	3	2	2	2
CO3	3	3	3	3	3	2	2	3	2	2	2	3			2
CO4	2	3	2	2	2			3	3	3	2	3	2	2	2
CO5	3	3	3	2	2	2	2	3	2		2	3	2	2	2

3/2/1: high/medium/low

JUSTIFICATIONS FOR CO-PO MAPPING

MAPPING	LOW/ MEDIUM/ HIGH	JUSTIFICATION
100003/CS6 22T.1-PO1	HIGH	Identify technically and economically feasible problems by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS6 22T.1-PO2	HIGH	Identify technically and economically feasible problems by analysing complex engineering problems reaching substantiated conclusions using first principles of mathematics.
100003/CS6 22T.1-PO3	HIGH	Design solutions for complex engineering problems by identifying technically and economically feasible problems.
100003/CS6 22T.1-PO4	HIGH	Identify technically and economically feasible problems by analysis and interpretation of data.
100003/CS6 22T.1-PO6	MEDIUM	Responsibilities relevant to the professional engineering practice by identifying the problem.
100003/CS6 22T.1-PO7	MEDIUM	Identify technically and economically feasible problems by understanding the impact of the professional engineering solutions.
100003/CS6 22T.1-PO8	HIGH	Apply ethical principles and commit to professional ethics to identify technically and economically feasible problems.
100003/CS6 22T.1-PO9	MEDIUM	Identify technically and economically feasible problems by working as a team.
100003/CS6 22T.1-PO10	MEDIUM	Communicate effectively with the engineering community by identifying technically and economically feasible problems.
100003/CS6 22T.1-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering and management principles by selecting the technically and economically feasible problems.
100003/CS6 22T.1-PO12	HIGH	Identify technically and economically feasible problems for long term learning.
100003/CS6 22T.1-PSO1	MEDIUM	Ability to identify, analyze and design solutions to identify technically and economically feasible problems.
100003/CS6 22T.1-PSO2	MEDIUM	By designing algorithms and applying standard practices in software project development and Identifying technically and economically feasible problems.
100003/CS6 22T.1-PSO3	MEDIUM	Fundamentals of computer science in competitive research can be applied to Identify technically and economically feasible problems.
100003/CS6 22T.2-PO1	HIGH	Identify and survey the relevant by applying the knowledge of mathematics, science, engineering fundamentals.

100003/CS6 22T.2-PO2	HIGH	Identify, formulate, review research literature, and analyze complex engineering problems get familiarized with software development processes.
100003/CS6 22T.2-PO3	HIGH	Design solutions for complex engineering problems and design based on the relevant literature.
100003/CS6 22T.2-PO4	HIGH	Use research-based knowledge including design of experiments based on relevant literature.
100003/CS6 22T.2-PO5	HIGH	Identify and survey the relevant literature for getting exposed to related solutions and get familiarized with software development processes by using modern tools.
100003/CS6 22T.2-PO6	MEDIUM	Create, select, and apply appropriate techniques, resources, by identifying and surveying the relevant literature.
100003/CS6 22T.2-PO8	HIGH	Apply ethical principles and commit to professional ethics based on the relevant literature.
100003/CS6 22T.2-PO9	MEDIUM	Identify and survey the relevant literature as a team.
100003/CS6 22T.2-PO10	HIGH	Identify and survey the relevant literature for a good communication to the engineering fraternity.
100003/CS6 22T.2-PO11	MEDIUM	Identify and survey the relevant literature to demonstrate knowledge and understanding of engineering and management principles.
100003/CS6 22T.2-PO12	HIGH	Identify and survey the relevant literature for independent and lifelong learning.
100003/CS6 22T.2-PSO1	MEDIUM	Design solutions for complex engineering problems by Identifying and survey the relevant literature.
100003/CS6 22T.2-PSO2	MEDIUM	Identify and survey the relevant literature for acquiring programming efficiency by designing algorithms and applying standard practices.
100003/CS6 22T.2-PSO3	MEDIUM	Identify and survey the relevant literature to apply the fundamentals of computer science in competitive research.
100003/CS6 22T.3-PO1	HIGH	Perform requirement analysis, identify design methodologies by using modern tools & advanced programming techniques and by applying the knowledge of mathematics, science, engineering fundamentals.
100003/CS6 22T.3-PO2	HIGH	Identify, formulate, review research literature for requirement analysis, identify design methodologies and develop adaptable & reusable solutions.

100003/CS6 22T.3-PO3	HIGH	Design solutions for complex engineering problems and perform requirement analysis, identify design methodologies.
100003/CS6 22T.3-PO4	HIGH	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS6 22T.3-PO5	HIGH	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools.
100003/CS6 22T.3-PO6	MEDIUM	Perform requirement analysis, identify design methodologies and assess societal, health, safety, legal, and cultural issues.
100003/CS6 22T.3-PO7	MEDIUM	Understand the impact of the professional engineering solutions in societal and environmental contexts and Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS6 22T.3-PO8	HIGH	Perform requirement analysis, identify design methodologies and develop adaptable & reusable solutions by applying ethical principles and commit to professional ethics.
100003/CS6 22T.3-PO9	MEDIUM	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS6 22T.3-PO10	MEDIUM	Communicate effectively with the engineering community and with society at large to perform requirement analysis, identify design methodologies.
100003/CS6 22T.3-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering requirement analysis by identifying design methodologies.
100003/CS6 22T.3-PO12	HIGH	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by analysis, identify design methodologies and develop adaptable & reusable solutions.
100003/CS6 22T.3-PSO3	MEDIUM	The ability to apply the fundamentals of computer science in competitive research and prior to that perform requirement analysis, identify design methodologies.
100003/CS6 22T.4-PO1	MEDIUM	Prepare technical report and deliver presentation by applying the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS6 22T.4-PO2	HIGH	Identify, formulate, review research literature, and analyze complex engineering problems by preparing technical report and deliver presentation.

100003/CS6 22T.4-PO3	MEDIUM	Prepare Design solutions for complex engineering problems and create technical report and deliver presentation.
100003/CS6 22T.4-PO4	MEDIUM	Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions and prepare technical report and deliver presentation.
100003/CS6 22T.4-PO5	MEDIUM	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and Prepare technical report and deliver presentation.
100003/CS6 22T.4-PO8	HIGH	Prepare technical report and deliver presentation by applying ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
100003/CS6 22T.4-PO9	HIGH	Prepare technical report and deliver presentation effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
100003/CS6 22T.4-PO10	HIGH	Communicate effectively with the engineering community and with society at large by prepare technical report and deliver presentation.
100003/CS6 22T.4-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work by prepare technical report and deliver presentation.
100003/CS6 22T.4-PO12	HIGH	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change by prepare technical report and deliver presentation.
100003/CS6 22T.4-PSO1	MEDIUM	Prepare a technical report and deliver presentation to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas.
100003/CS6 22T.4-PSO2	MEDIUM	To acquire programming efficiency by designing algorithms and applying standard practices in software project development and to prepare technical report and deliver presentation.
100003/CS6 22T.4-PSO3	MEDIUM	To apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs by preparing technical report and deliver presentation.
100003/CS6 22T.5-PO1	HIGH	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
100003/CS6 22T.5-PO2	HIGH	Identify, formulate, review research literature, and analyze complex engineering problems by applying engineering and management principles to achieve the goal of the project.

100003/CS6 22T.5-PO3	HIGH	Apply engineering and management principles to achieve the goal of the project and to design solutions for complex engineering problems and design system components or processes that meet the specified needs.
100003/CS6 22T.5-PO4	MEDIUM	Apply engineering and management principles to achieve the goal of the project and use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
100003/CS6 22T.5-PO5	MEDIUM	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO6	MEDIUM	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities by applying engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO7	MEDIUM	Understand the impact of the professional engineering solutions in societal and environmental contexts, and apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO8	HIGH	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice and to use the engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO9	MEDIUM	Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO11	MEDIUM	Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PO12	HIGH	Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PSO1	MEDIUM	The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas. Apply engineering and management principles to achieve the goal of the project.

100003/CS6 22T.5-PSO2	MEDIUM	The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry and to apply engineering and management principles to achieve the goal of the project.
100003/CS6 22T.5-PSO3	MEDIUM	The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur and apply engineering and management principles to achieve the goal of the project.

