Tribhuvan University

Faculty of Humanities and Social Science

A PROJECT REPORT FOR

"Bus Ticketing System using Content Based, Occupancy and Price Automation Algorithms"

Submitted to

Department of Computer Application

Advanced College of Engineering and Management

Kalanki, Kathmandu

In partial fulfillment of the requirements for the bachelor's in computer application

Submitted by

Stephen G. Shrestha	**Rubek Maharjan**
37602025	**37602021**
BCA 8th Semester	**BCA 8th Semester**

Under the Supervision of

Er. Sakriya Maharjan

Tribhuvan University

Faculty of Humanities and Social Science

Advance College of Engineering and Management

Kalanki, Kathmandu

Bachelor in Computer Application (BCA)

SUPERVISOR'S RECOMMENDATION

I hereby recommend that this project prepared under my supervision by **Rubek Maharjan and Stephen G. Shrestha** entitled **"Bus Ticketing System using Content Based, Occupancy and Price Automation Algorithms"** in partial fulfillment of the requirements for the degree of Bachelor of Computer Application is recommended for the final evaluation.

....

Er. Sakriya Maharjan

SUPERVISOR

Department of Computer Application

Advanced College of Engineering and Management

Kalanki, Kathmandu

Tribhuvan University

Faculty of Humanities and Social Science

ADVANCED COLLEGE OF ENGINEERING AND MANAGEMENT

LETTER OF ACCEPTANCE

This is to certify that this project prepared by **“Rubek Maharjan and Stephen G. Shrestha”** entitled **“MahaBus”** in partial fulfillment of the requirements for the 8th semester of Bachelor in Computer Application has been evaluated. In our opinion it is satisfactory in the scope and quality as a project for the defense project.

Er. Sakriya Maharjan	Ajaya Shrestha
Project Supervisor	Head of Department (acem)
BCA Department (acem)	

External Examiner	Internal Examiner
Tribhuvan University	

ABSTRACT

The Bus Ticketing System is a comprehensive platform designed to streamline the booking and management of bus tickets. It allows users to search for bus routes, book tickets, and access travel schedules, ensuring a convenient and efficient booking experience. The system features a dual-panel architecture, with distinct capabilities for users and administrators. Users can easily view available routes, select preferred seats, and make payments, while administrators can manage schedules, allocate buses, and monitor ticket sales and occupancy rates. This organized approach enhances operational efficiency and minimizes manual errors, providing a seamless experience for all stakeholders. The system's architecture supports extensive data collection and processing, ensuring efficient management of both user interactions and administrative tasks. The software aims to improve the ticketing process by offering an intuitive interface and integrated communication features. Future developments will focus on advanced data analytics and machine learning techniques to optimize route planning, enhance user experience, and provide detailed insights into travel patterns and preferences. This innovative technology offers a forward-thinking solution for improving the organization and execution of bus ticketing in an increasingly digital world.

Keywords: *ReactJs, React Query, NodeJs, ExpressJs, Prisma ORM, PostgresSQL, GitHub, TailwindCSS*

ACKNOWLEDGEMENT

We take this opportunity to express our deepest and sincere gratitude to our **Supervisor Er. Sakriya Maharjan** for her insightful advice, motivating suggestions, invaluable guidance, help, and support in the successful completion of this project, and also for her constant encouragement and advice throughout our Bachelor's program.

We express our deep gratitude to the Head of the Computer Application Department, **Ajaya Shrestha** for her indispensable guidance and support to harness our skills.

The in-time facilities provided by the department throughout the Bachelor's program are also equally acknowledgeable.

We would like to convey our thanks to the teaching and non-teaching staff of the **Department of Computer Application, acem** for their invaluable help and support throughout the period of Bachelor's Degree. We are also grateful to all our classmates for their help, encouragement, and invaluable suggestions.

Yours Sincerely,

Rubek Maharjan

Stephen G. Shrestha

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

CASE	Computer Aided Software Engineering
RDBMS	Relational Database Management System
DFD	Data Flow Diagram
ERD	Entity Relationship Diagram
UI	User Interface
JS	JavaScript
HTML	Hypertext Markup Language
API	Application Programming Interface

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Introduction

In the current era of digital transformation, public transportation systems are undergoing significant changes to enhance service efficiency, accessibility, and passenger satisfaction. One critical aspect of these advancements is the modernization of bus ticketing systems. Traditional ticketing methods, often characterized by long queues, paper tickets, and manual processing, are increasingly becoming obsolete. [1]
The proposed Bus Ticketing System aims to address the challenges faced by both passengers and transportation providers. For passengers, the system will offer an intuitive platform to easily book, manage, and validate bus tickets through website. The system will contribute to sustainable mobility by reducing the reliance on paper tickets and supporting the development of smart cities.

1.2 Problem Statement:

The bus ticketing sector frequently encounters several persistent challenges that impact both operators and passengers. Common issues include overbooking, inaccurate schedule information, and a lack of real-time updates. These problems often lead to missed buses, extended waiting times, and confusion regarding delays or route changes. Additionally, inefficiencies in handling cancellations, refunds, and last-minute bookings contribute to user dissatisfaction, leading to a loss of trust and reduced customer retention. [2]

To address these challenges, modern bus ticketing platforms should integrate real-time tracking and dynamic scheduling systems. These tools provide passengers and operators with accurate, up-to-date information on bus locations, expected arrival times, and any unforeseen delays. Secure online payment gateways and automated booking confirmations can further streamline the reservation process, reducing human errors and preventing overbooking. A clean and intuitive user interface (UI) that allows passengers to easily search routes, reserve seats, and manage their bookings is essential for a seamless experience.

Equally important is the development of responsive and reliable customer support channels to handle queries, cancellations, and refunds quickly and effectively. Support via chat, phone, or email can enhance the overall experience, resolve issues faster, and build stronger relationships with customers. By adopting these improvements, bus ticketing platforms can boost efficiency, enhance user satisfaction, and foster long-term loyalty in a competitive market.

1.3 Objectives:

- To simplify ticket booking and seat selection through a user-friendly platform using Content-Based Algorithms for personalized route and bus recommendations, reducing manual errors, and eliminating physical counters.
- To enhance customer experience with Shortest Path Algorithms for optimal route suggestions, along with real-time bus updates, live tracking, and responsive support.

1.4 Scope & Limitation

1.4.1 Scope

The objective of this project is to design, develop, and deploy a comprehensive online platform for booking bus tickets. The system will serve as a centralized hub where users can search for bus routes, compare schedules, select seats, and complete their ticket purchases. The platform will provide detailed bus information, real-time seat availability, and payment options, delivering a seamless experience for both passengers and bus operators.

The project will involve the following key components:

User Registration and Authentication: Users will be able to create accounts, log in, and manage their profiles. They can also save routes, track bookings, and receive notifications about upcoming trips, special offers, and updates.

Bus Schedule and Route Listings: The system will provide a comprehensive database of bus routes and schedules. Users will be able to search for buses by departure and destination locations, dates, times, and bus type (luxury or regular). Detailed information on each route, including departure times, travel durations, stops, and end destination.

Ticket Booking and Payment: Users will be able to select their desired bus, route, time, and seats and proceed to book their tickets. The platform will support multiple payment options, including credit/debit cards, digital wallets, and online banking. Users will have the option to modify or cancel their bookings within the system’s cancellation policies.

1.4.2 Limitation

Dependence on Internet Connectivity: The system relies heavily on internet access for users to book tickets, view schedules, and check real-time seat availability. Limited or no internet access could prevent passengers from using the platform, especially in rural areas or during travel.

Real-Time Data Sync Issues: The system's accuracy depends on real-time updates from bus operators. Delays or failures in syncing data (seat availability, cancellations) could lead to inaccurate information being displayed, resulting in booking errors or double bookings.

Scalability Challenges: As the number of users, routes, and transactions grows, the system may face scalability challenges. If not built with scalability in mind, it could experience slowdowns or crashes during periods of high demand, such as holidays or peak travel times.

Refund and Cancellation Delays: While the system automates ticket cancellations and refunds, delays in processing refunds due to banking systems or payment gateway issues could lead to dissatisfaction among passengers, especially in urgent situations.

1.5 Development Methodology

In this project, we have adopted the Agile methodology to manage and streamline the software development process. Agile is an iterative and flexible approach that breaks down the development cycle into smaller, manageable units called sprints or iterations, typically lasting one to four weeks. Each sprint involves planning, designing, developing, and testing a set of prioritized features. This iterative process allows us to deliver working modules of the application frequently, enabling faster feedback and early identification of issues. Agile supports dynamic requirements, making it highly suitable for projects like ours where continuous improvement and adaptability are essential.

The Agile approach in this project promotes collaboration among cross-functional teams, including developers, testers, and stakeholders. Regular Agile practices such as daily stand-up meetings, sprint planning, sprint reviews, and retrospectives ensure clear communication, transparency, and ongoing process improvement. Customer or client feedback is gathered regularly to ensure that the final product aligns closely with user expectations. By using Agile, we can maintain a high level of flexibility, respond quickly to changes, reduce development risks, and improve the overall quality and relevance of the software being delivered.

1.6 Report Organization

Chapter 1 introduces the system, we outline its objectives, limitations, and the compelling reasons that drove its development. By providing a concise overview, we aim to offer a clear understanding of the system's purpose and the boundaries it operates within.

Chapter 2 summarizes system management; various applications have been developed with intriguing features.

Chapter 3 encompass various aspects, including functional and non-functional requirements, feasibility analysis, ER diagram, DFD, system design, system architecture, database schema, and interface design.

Chapter 4 talks about various tools that were used in the system development process to facilitate implementation. Detailed implementation procedures were followed, including thorough testing to assess the system's performance and functionality.

Chapter 5 highlights a summary of lessons learnt, outcome and conclusion of the whole project and explains what has been done and what further improvements could be done.

CHAPTER 2 BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

In many countries, the bus transportation system operates informally, with no centralized or standardized platform for ticketing, scheduling, or managing bus routes. Often, passengers must rely on physical ticket counters, bus terminals, or local agents to book tickets, leading to inconsistent pricing, lack of transparency, and frequent overbooking. Furthermore, pricing is often non-standardized, with rates varying based on demand or the operator's discretion.

Due to these challenges, several bus ticketing websites and mobile applications have emerged in the market. Some platforms favor larger bus operators, leaving smaller regional services underrepresented, while others may not provide users with the flexibility to modify or cancel bookings easily. [3]

In response to this situation, we are developing a comprehensive bus ticketing website that addresses the needs of both passengers and bus operators. The goal is to provide a user-friendly platform that offers a transparent, real-time view of bus schedules, seat availability, and pricing. The system will also allow bus operators to list and manage their routes, prices, and services in a competitive environment. By balancing the needs of passengers, operators, and local businesses, this platform aims to standardize the bus ticketing process, making it accessible, efficient, and equitable for all users involved.

2.2 Literature Review

A study by Sharma explores how user experience impacts the adoption and satisfaction of online bus ticket booking systems. Findings highlight that interface design, ease of use, and customer support significantly influence user behavior and system effectiveness, crucial for enhancing service quality and user satisfaction. [4]

According to a report by World Bank, technological innovations such as real-time data integration, mobile applications, and AI-driven analytics are transforming transportation management systems. These innovations improve operational efficiency, scheduling accuracy, and customer service within bus ticket booking platforms, fostering a more responsive and reliable travel experience. [5]

Research by J.D. Power examines customer satisfaction and loyalty in online travel booking, including bus ticket reservations. The study identifies key drivers such as service quality, price transparency, booking convenience, and post-purchase support, influencing customer retention and brand loyalty in competitive travel markets. [6]

MarketsandMarkets' report explores the integration of Internet of Things (IoT) devices, such as smart ticketing systems and vehicle trackers, in public transportation. IoT platforms and edge computing enable real-time data processing, enhancing operational efficiency and passenger experience. [7]

McKinsey & Company's analysis explores the application of AI and machine learning algorithms in personalizing travel recommendations and optimizing pricing strategies. AI-driven chatbots, predictive analytics, and dynamic pricing models enhance customer engagement and revenue generation in bus ticketing platforms. [8]

Studies by UC Berkeley assess the impact of ride-sharing services (e.g., Uber, Lyft) on bus travel demand and service utilization. Integration with multi-modal transportation apps and shared mobility solutions offer commuters flexible travel options and influence modal shift behaviors in urban areas. [9]

Security measures in bus ticketing systems include HTTPS protocol for secure data transmission, SSL/TLS encryption for data protection, and OAuth/JWT for secure authentication. Regular security audits and vulnerability assessments are conducted to detect and mitigate threats, complemented by firewall configurations and intrusion detection systems (IDS) to ensure compliance with regulatory standards like PCI-DSS, maintaining user trust and system integrity. [10]

According to Mbarika manual systems were prone to errors, with issues like overbooking or inconsistent pricing being common. Moreover, passengers lacked access to real-time information, such as bus schedules, delays, or cancellations. [11]

Grob and Bernon discuss how the introduction of digital ticketing revolutionized the transportation sector, as users could book tickets online or through mobile devices, reducing the reliance on physical ticket counters and improving operational efficiency. [12]

Shiroor analyze the development of online ticketing platforms, noting that the integration of web-based and mobile applications has become a game-changer for public transport. These platforms provide users with the ability to search for available buses, view schedules, select seats, and pay for tickets online. [13]

Kaur and Sandhu, online systems offer bus operators the ability to optimize route planning and dynamically adjust prices based on demand. [14]

According to Kumar, allowed users to book tickets on the go, view seat layouts, and receive real-time alerts. The convenience provided by these apps enhanced the user experience and widened the customer base, as passengers no longer needed access to desktop computers to make bookings. [15]

Sasidharan highlight the issue of digital divide, where passengers in rural areas or regions with poor internet connectivity may struggle to use online systems. In these cases, the reliance on manual or agent-based ticketing persists, which can limit the reach of digital platforms. [16]

CHAPTER 3 SYSTEM ANALYSIS AND DESIGN

3.1 System Analysis

3.1.1 Requirement Analysis

The requirement analysis for a Bus Ticket Booking System in Nepal involves identifying and documenting the functional and non-functional requirements necessary to develop a comprehensive and efficient system. This analysis aims to ensure that the system meets the needs of users, bus operators, and other stakeholders.

i. Functional Requirement

There are various basic as well as secondary functional requirements for a bus ticketing website. Some of them are user registration and authentication, manages buses, view buses, book seats, view ticket etc. User authentication should ensure that only authorized users can access certain features and information. Similarly, users should be able to search and view for buses based on various criteria, such as location and specific features.

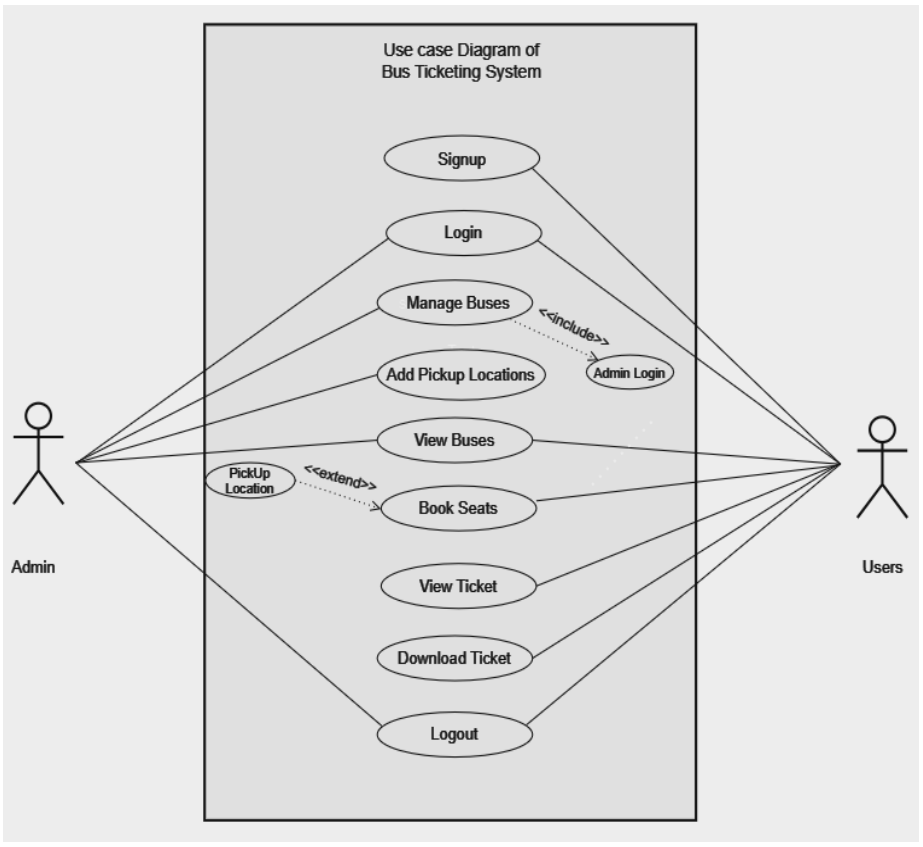


Figure 3.1 Use Case Diagram of Bus Ticketing System

In the above use case diagram, there are two actors (admin and users). The actors can login and logout. The admin initiates the manage buses, add pickup location and view buses. User's initiates view buses, book seats, view ticket and download ticket.

ii. Non-Functional Requirement

Performance, usability, user experience and security are important non-functional requirements for a website. The website should have fast loading times to ensure a smooth user experience. Content and features should adapt to smaller screens without loss of functionality or readability. The website should employ robust security measures to protect user data and prevent unauthorized access. User login and authentication processes should be secure and follow best practices.

3.1.2 Feasibility Analysis

Feasibility study is a process of analyzing technical, operational, and economic viability of a project. It ensures that the proposed system aligns with the team's goals, resources, and capabilities.

i. Technical

The selected technology stack is modern, scalable, and well-suited for web-based systems. React is used for building a fast and responsive user interface, while Golang is used on the backend for efficient and reliable API development. PostgreSQL is chosen as the database due to its stability and support for handling structured data. No extra hardware was needed, as the development was done using standard computers and internet access. The system runs smoothly on available infrastructure and supports future upgrades if user demand increases.

Additionally, the stack ensures high performance, maintainability, and easy integration with third-party services such as payment gateways and real-time tracking APIs. The use of RESTful APIs promotes modularity, enabling each component to be independently updated or scaled. This architecture is ideal for future enhancements like mobile app integration or machine learning-based recommendations.

ii. Operational

The system is built to be simple and easy to use. Both users and admins can use the website without needing technical knowledge. The user interface is clean and responsive, making it easy to browse buses, book tickets, and download them. The admin panel is also straightforward, allowing admins to manage buses and bookings without needing any special training.

iii. Economic

The project is cost-effective because it uses free and open-source tools such as React, Golang, PostgreSQL, Git, and Postman. These tools help reduce development and maintenance costs. There are no licensing fees, and the wide community support makes it easier to fix issues and improve the system in the future. Overall, the system is affordable to develop, run, and maintain.

iv. Schedule

The system is completed within the scheduled time and does not exceed the scheduled time. The system was developed within the 3 months as estimated before starting the developing process.

3.1.3 Object Modelling using Class and Object Diagrams

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP).

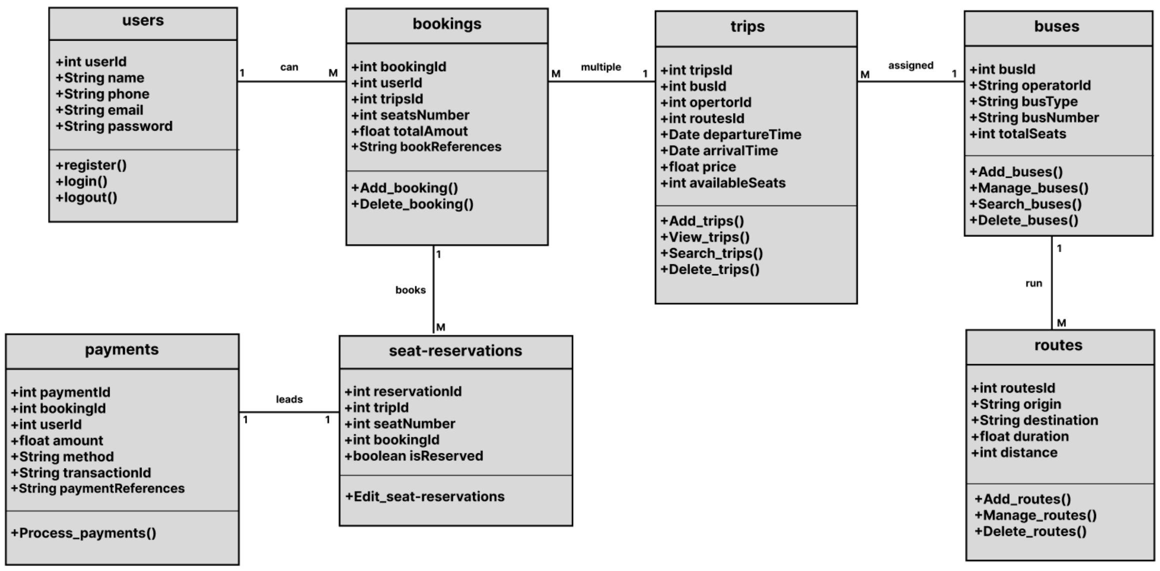


Figure 3.2 Object Modelling using Class and Object Diagrams for Bus Ticketing System

The given class diagram represents the object-oriented structure of a Bus Ticketing System using UML. It consists of several interconnected classes that define the key components and their relationships. The Users class contains user-specific attributes such as name, phone, email, and password, along with methods for registration, login, and logout. A user can make multiple Bookings, each linked to a specific Trip, and each booking includes seat numbers, total amount, and booking references. The Trips class holds trip-related information like bus ID, route ID, departure and arrival times, price, and available seats, and is linked to the Buses class, which manages information about the bus type, operator, and total seats. Each trip also follows a specific Route, defined by origin, destination, duration, and distance.

Similarly, the Seat-Reservations class records individual seat bookings, linking a seat number to a trip and a booking, with a boolean indicating if the seat is reserved. For each booking, there is a corresponding Payment containing details such as amount, method, and transaction ID, and processed through the Process_payments() method. All these components work together to form a complete system where users can search trips, book seats, and make secure payments, while the system manages buses, routes, and seat availability through clearly defined object relationships and operations.

3.1.4 Dynamic Modelling using State and Sequence Diagrams

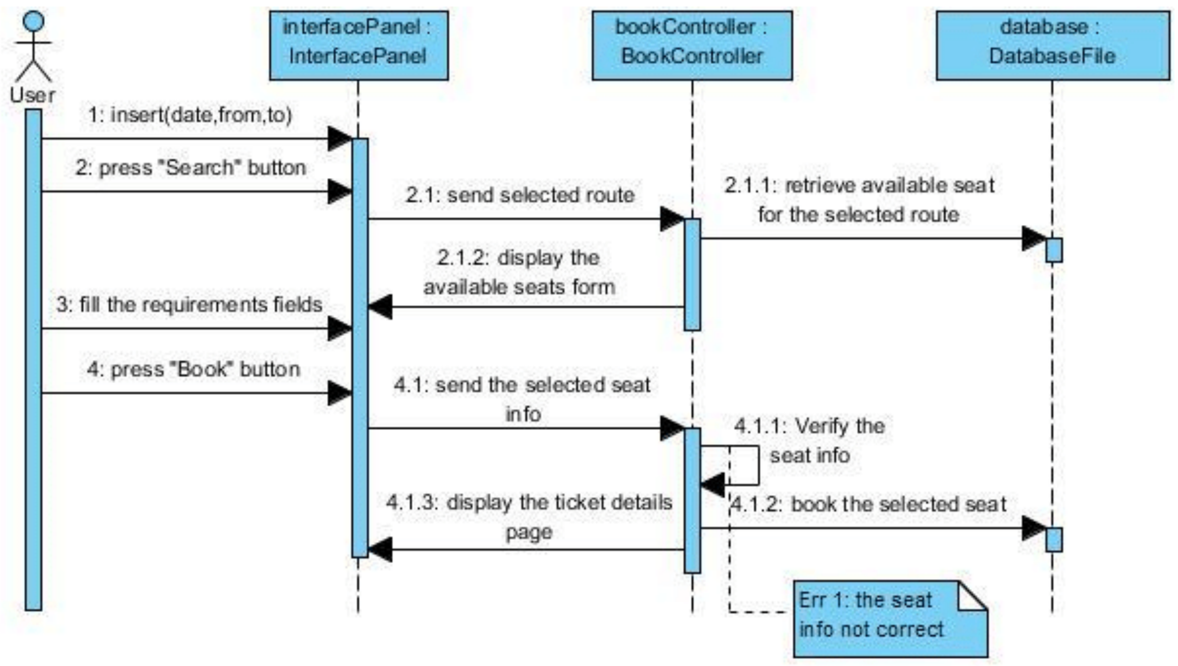

User
interfacePanel : InterfacePanel
bookController : BookController
database : DatabaseFile
1: insert(date,from,to)
2: press "Search" button
2.1: send selected route
2.1.1: retrieve available seat for the selected route
2.1.2: display the available seats form
3: fill the requirements fields
4: press "Book" button
4.1: send the selected seat info
4.1.1: Verify the seat info
4.1.2: book the selected seat
4.1.3: display the ticket details page
Err 1: the seat info not correct

Figure 3.3 Dynamic Modelling using State and Sequence Diagrams for Bus Ticketing System

The given figure is a sequence diagram, a type of dynamic modeling used in software engineering to depict the interaction between different objects or components in a system over time. This specific diagram illustrates the process of booking a seat through a ticket booking system. The sequence begins with the user inserting travel details (such as date, from, and to locations) and then pressing the "Search" button. The InterfacePanel sends the selected route to the BookController, which then queries the database for available seats.

Once the available seats are retrieved, the InterfacePanel displays them to the user. The user then fills in the required fields and presses the "Book" button. This triggers the sending of selected seat information to the BookController, which verifies the seat info. If the seat is valid, it is booked through the database, and a ticket details page is displayed to the user. However, if the seat information is incorrect, an error is thrown. This diagram helps in understanding the flow of control and data among system components during the booking process, making it a crucial tool for dynamic system design.

3.1.5 Process Modelling using Activity Diagrams

Process modeling is the graphical representation of business processes or workflows. One common tool for this is the activity diagram, which captures the dynamic aspects of a system. Rather than focusing on implementation details, activity diagrams illustrate the flow of control between activities within the system. They model both sequential and concurrent processes, making them ideal for visualizing complex workflows.

Activity diagrams emphasize the conditions and order of operations, supporting different flow types such as linear, branching, and parallel flows through elements like forks and joins. Often referred to as object-oriented flowcharts, these diagrams represent activities as sets of actions or operations and are commonly used in behavioral modeling.

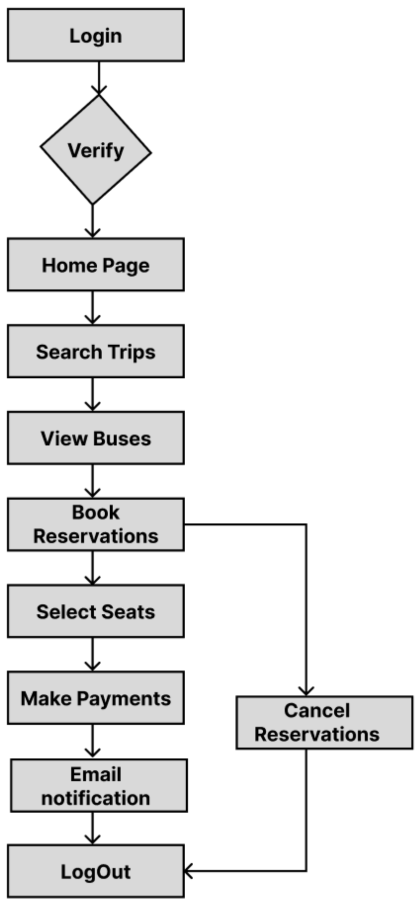


Figure 3.4 Process Modelling using Activity Diagrams for Bus Ticketing System (User)

As illustrated in the activity diagram above, the process begins when a user logs into the system. The user first browses the contents of the home page, where they can search for and view available trips. After selecting a trip, the user proceeds to choose a bus and book a seat. Following the booking, payment is made, and the user receives an email notification confirming their ticket.

However, if the user decides to cancel the reservation, the steps involving seat selection and payment are skipped. Instead, the user proceeds to log out, thereby completing the process.

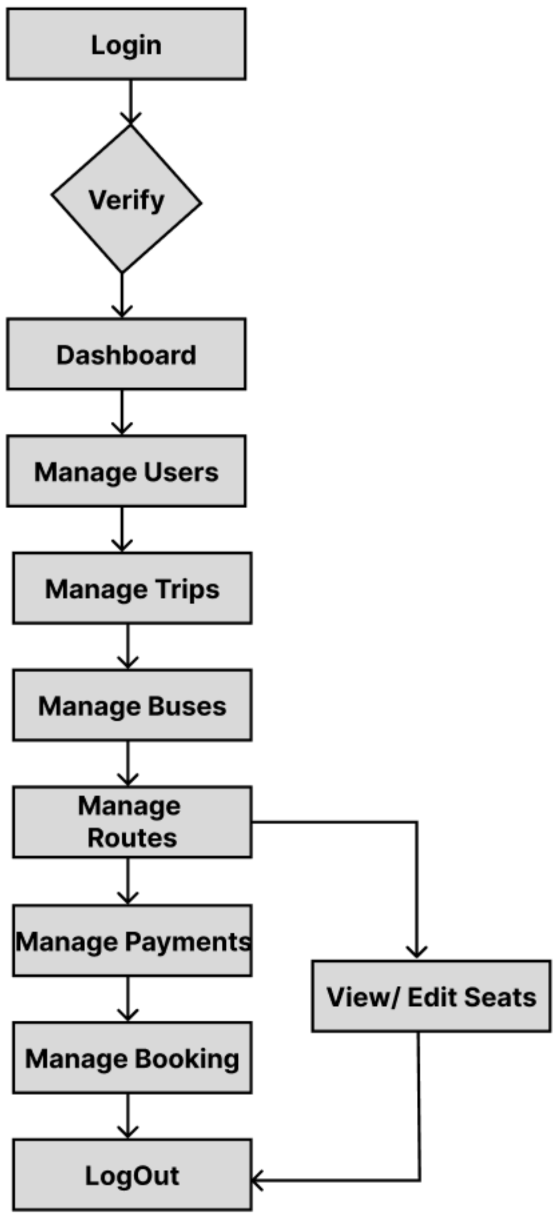


Figure 3.5 Process Modelling using Activity Diagrams for Bus Ticketing System (Admin)

As illustrated in the activity diagram above, the admin process begins with logging into the system, followed by a verification step. Once verified, the admin is directed to the dashboard, where they can manage various aspects of the system. These include managing users, trips, buses, routes, payments, and bookings. However, the admin may choose to skip the payment and booking management steps and instead proceed to view or edit seat arrangements. After completing the necessary administrative tasks, the admin logs out, marking the end of the process.

3.2 System Design

System Design is the process of defining the architecture, modules, components, and their interfaces, along with the data flow within a system. It involves breaking down the system into smaller, manageable components to clearly specify how these components interact and work together to meet the defined requirements.

3.2.1 Refinement of Class, Object, State, Sequence and Activity Diagrams

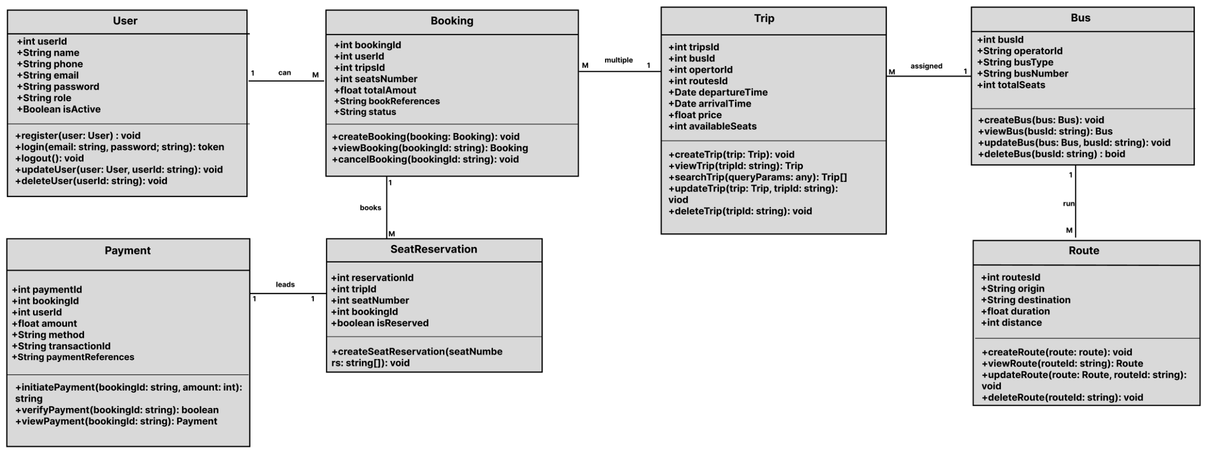

User
+int userId
+String name
+String phone
+String email
+String password
+String role
+Boolean isActive
+register(user: User) : void
+login(email: string, password; string): token
+logout(): void
+updateUser(user: User, userId: string): void
+deleteUser(userId: string): void
1
can
M
Booking
+int bookingId
+int userId
+int tripsId
+int seatsNumber
+float totalAmout
+String bookReferences
+String status
+createBooking(booking: Booking): void
+viewBooking(bookingId: string): Booking
+cancelBooking(bookingId: string): void
M
multiple
1
Trip
+int tripsId
+int busId
+int opertorId
+int routesId
+Date departureTime
+Date arrivalTime
+float price
+int availableSeats
+createTrip(trip: Trip): void
+viewTrip(tripId: string): Trip
+searchTrip(queryParams: any): Trip[]
+updateTrip(trip: Trip, tripId: string): viod
+deleteTrip(tripId: string): void
M
assigned
1
Bus
+int busId
+String operatorId
+String busType
+String busNumber
+int totalSeats
+createBus(bus: Bus): void
+viewBus(busId: string): Bus
+updateBus(bus: Bus, busId: string): void
+deleteBus(busId: string) : boid
1
run
M
Route
+int routesId
+String origin
+String destination
+float duration
+int distance
+createRoute(route: route): void
+viewRoute(routeId: string): Route
+updateRoute(route: Route, routeId: string): void
+deleteRoute(routeId: string): void
1
books
M
Payment
+int paymentId
+int bookingId
+int userId
+float amount
+String method
+String transactionId
+String paymentReferences
+initiatePayment(bookingId: string, amount: int): string
+verifyPayment(bookingId: string): boolean
+viewPayment(bookingId: string): Payment
leads
1
1
SeatReservation
+int reservationId
+int tripId
+int seatNumber
+int bookingId
+boolean isReserved
+createSeatReservation(seatNumbers: string[]): void

Figure 3.6 Refinement Class Diagram for Bus Ticketing System

The given class diagram represents a refined object-oriented model of an online bus booking system. In terms of class and object refinement, it illustrates how abstract entities such as users, bookings, and trips are transformed into concrete classes with specific attributes and behaviors. The User class handles personal information and authentication processes, while the Booking class links users to specific trips, storing details like seat numbers and booking status. The Trip class connects to a Bus and Route, detailing departure and arrival times, available seats, and pricing. Each trip is associated with a specific bus, defined in the Bus class, which contains bus type, number, and seat capacity.

The Route class describes the journey's origin, destination, duration, and distance. Seat selections are managed by the SeatReservation class, which tracks reserved seats for each booking and trip. Lastly, the Payment class facilitates transaction processes, linking each payment to a booking and storing details like amount and transaction ID. These classes interact with one another through well-defined relationships, showing a detailed and organized refinement of objects, ensuring system modularity, maintainability, and clarity in software design.

3.2.2 Component Diagrams

A component diagram is a specific type of UML (Unified Modeling Language) diagram used to represent the structural organization of a software system. It shows the interconnections and dependencies between various components, such as modules, libraries, executables, and other key building blocks. This diagram helps visualize how components interact, how they are grouped, and how they contribute to the overall system architecture. It is especially useful in modeling large and complex systems by breaking them down into manageable, interconnected parts.

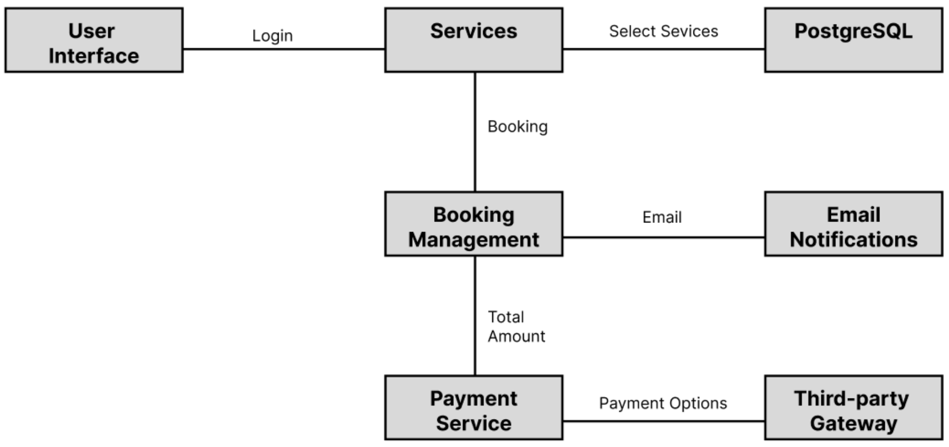


Figure 3.7 Component Diagrams for Bus Ticketing System

The components illustrated in the above diagram are interconnected to form a complete and functional system. The system comprises several key components, including a service module that allows users to browse and select from a range of available services, a booking management system that handles ticket reservations, and a payment service that integrates with third-party providers such as Khalti to securely process transactions.

3.3.3 Deployment Diagrams

Deployment diagrams are a type of UML structural diagram that illustrate the physical arrangement of hardware and software components within a system. They show how these components are distributed and connected in a real-world environment, effectively mapping out where and how software elements are deployed across hardware nodes. This helps in understanding the infrastructure setup and the interaction between different parts of the system.

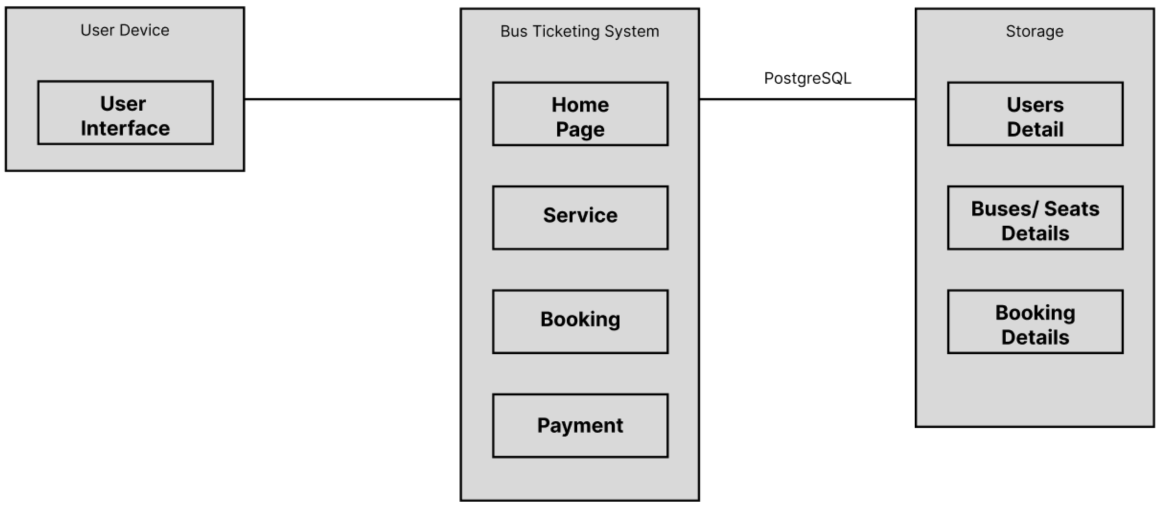


Figure 3.8 Deployment Diagram for Bus Ticketing System

In the deployment diagram, the system is divided into three primary nodes: the user's device, the bus ticketing system, and the storage server. The user's device hosts the user interface, which allows users to interact with the system by browsing services, selecting trips, booking seats, and making payments. These actions are processed by the bus ticketing system, which serves as the application server and includes core modules such as the home page, service module, booking system, and a payment module that integrates with third-party gateways like Khalti. All application logic and service handling occur within this central system. On the backend, a storage server powered by PostgreSQL is used to manage persistent data. This includes user details, bus, and seat information, and booking records. The components are interconnected, enabling smooth communication between the user interface, application logic, and data storage to ensure the system functions effectively.

3.3 Algorithms

1. User Preference Analysis and Trips Sorting Algorithm Breakdown of the Algorithm:

Breakdown of the Algorithm:

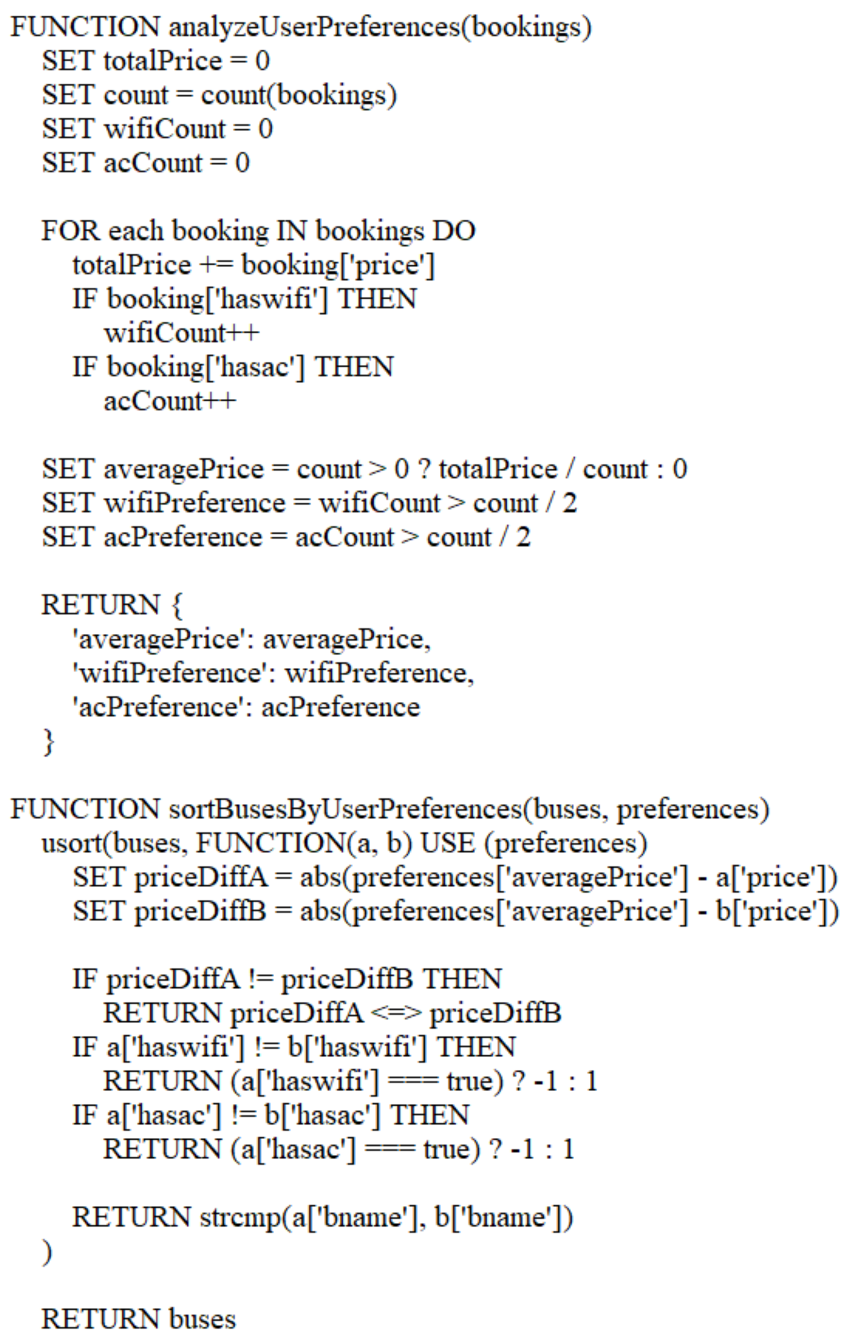
User Preference Analysis: The system checks the user's booking history to calculate their average ticket price and determine. If most of their past bookings included these features, they're marked as preferred.

Bus Sorting by Preferences: When displaying buses, the system sorts them based on the user's preferences:

1. Buses closest to the average price come first.
2. Among those, buses with **Wi-Fi** and **AC** are prioritized.
3. If still tied, buses are sorted alphabetically by name.

This helps users quickly find buses that match their usual choices. [24]

Pseudo Code of Algorithm:



```
FUNCTION analyzeUserPreferences(bookings)
   SET totalPrice = 0
   SET count = count(bookings)
   SET wifiCount = 0
   SET acCount = 0

   FOR each booking IN bookings DO
      totalPrice += booking['price']
      IF booking['haswifi'] THEN
         wifiCount++
      IF booking['hasac'] THEN
         acCount++

   SET averagePrice = count > 0 ? totalPrice / count : 0
   SET wifiPreference = wifiCount > count / 2
   SET acPreference = acCount > count / 2

   RETURN {
      'averagePrice': averagePrice,
      'wifiPreference': wifiPreference,
      'acPreference': acPreference
   }

FUNCTION sortBusesByUserPreferences(buses, preferences)
   usort(buses, FUNCTION(a, b) USE (preferences)
      SET priceDiffA = abs(preferences['averagePrice'] - a['price'])
      SET priceDiffB = abs(preferences['averagePrice'] - b['price'])

      IF priceDiffA != priceDiffB THEN
         RETURN priceDiffA <=> priceDiffB
      IF a['haswifi'] != b['haswifi'] THEN
         RETURN (a['haswifi'] === true) ? -1 : 1
      IF a['hasac'] != b['hasac'] THEN
         RETURN (a['hasac'] === true) ? -1 : 1

      RETURN strcmp(a['bname'], b['bname'])
   )

   RETURN buses
```

Figure 3.9 Pseudo Code for Bus Sorting Algorithm

The analyzeUserPreferences function processes a user's past bookings to extract insights. It calculates the average price of booked buses, as well as the user's preference for Wi-Fi and air conditioning (AC) based on the count of buses with these features. Wi-Fi or AC is marked as preferred if more than half of the bookings include it. This information provides a baseline for understanding the user's needs.

The sortBusesByUserPreferences function organizes buses according to the user's preferences. It prioritizes buses with ticket prices closer to the user's average spending. If two buses have the same price difference, Wi-Fi and AC preferences are considered, giving priority to buses with these amenities. If all else is equal, the buses are sorted alphabetically by name, ensuring a consistent ranking order.

The contentBased function combines the above functionalities to recommend buses tailored to the user's preferences. It first analyzes past bookings to understand user behavior, then sorts the available buses using the derived preferences. The result is a list of buses ranked in order of suitability, helping users find options that best match their past choices and needs.

2. Occupancy Optimization Algorithm:

Breakdown of the Algorithm:

Occupancy Analysis:

The algorithm calculates how full each bus trip is by dividing the number of booked seats by the total number of available seats. This occupancy rate becomes the basis for scoring each trip. Trips that are more than 60% full get a 1.5× boost in score, encouraging the system to prioritize them. If the trip has less than 20% occupancy, its score is reduced by 50%. Trips that have less than 10% occupancy are filtered out entirely as low performing.

Content Score Balance:

If a trip also has a content score (which might be based on ratings, images, or completeness), the algorithm blends it with the occupancy score using a weighted average—60% content score and 40% occupancy score—to create a final sorting score.

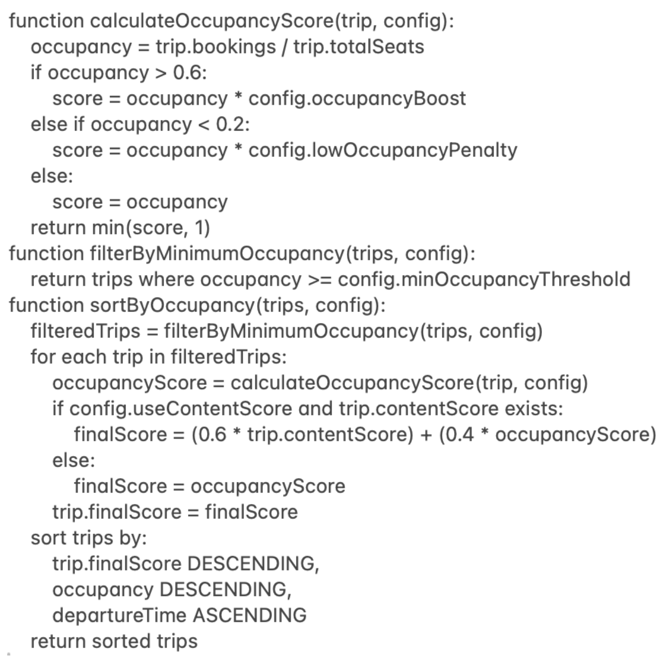
Sorting Trips:

After scoring, trips are sorted using a three-step logic:

1. Final Score (Occupancy + Content)
2. Occupancy Percentage
3. Departure Time (earlier trips first)

This ensures trips with better booking rates, verified quality, and sooner departure are shown first. An additional smart filtering feature selects a balanced variety of trips by checking for route and operator diversity, ensuring users see high-performing and varied results.

Pseudo Code of Algorithm:



```
function calculateOccupancyScore(trip, config):
    occupancy = trip.bookings / trip.totalSeats
    if occupancy > 0.6:
        score = occupancy * config.occupancyBoost
    else if occupancy < 0.2:
        score = occupancy * config.lowOccupancyPenalty
    else:
        score = occupancy
    return min(score, 1)
function filterByMinimumOccupancy(trips, config):
    return trips where occupancy >= config.minOccupancyThreshold
function sortByOccupancy(trips, config):
    filteredTrips = filterByMinimumOccupancy(trips, config)
    for each trip in filteredTrips:
        occupancyScore = calculateOccupancyScore(trip, config)
        if config.useContentScore and trip.contentScore exists:
            finalScore = (0.6 * trip.contentScore) + (0.4 * occupancyScore)
        else:
            finalScore = occupancyScore
        trip.finalScore = finalScore
    sort trips by:
        trip.finalScore DESCENDING,
        occupancy DESCENDING,
        departureTime ASCENDING
    return sorted trips
```

Figure 3.10 Pseudo Code for Occupancy Optimization Algorithm

This algorithm starts by analyzing each trip's occupancy percentage. Based on set thresholds, it boosts or penalizes the trip's score. If content scoring is enabled, it merges content and occupancy scores using weighted logic. After filtering out trips below the minimum threshold **(10%)**, the remaining trips are sorted based on their combined final score, ensuring that the most efficient and high-quality trips are shown first to users. The algorithm can also apply smart filtering to include diverse operators and routes for better representation.

3.Dynamic Price Automation

Breakdown of the Algorithm:

Time-Based Pricing:

The algorithm checks how far away the trip is from its departure time:

- If it's more than 48 hours away, a 15% discount is applied.
- If it's less than 6 hours away, a 25% price increase is applied.

Occupancy-Based Pricing:

Trips with more than 70% occupancy are in high demand and get a 30% price increase, while those with less than 30% occupancy get a 20% discount.

Peak Hour and Velocity Adjustment:

If a trip departs during peak hours (7–10 AM or 5–8 PM)**,** a 15% increase is added. Additionally, if a trip is being booked quickly after creation, it gets a 5–10% increase**,** while trips that are old and not being booked get a small discount**.**

Price Boundaries:

All final prices are restricted between a maximum of 50% increase and 30% decrease from the original price to avoid extreme changes.

Pseudo Code of Algorithm

```
function getTimeFactor(trip, config):
   hoursUntilDeparture = (trip.departureTime - now) in hours
   if hoursUntilDeparture > config.earlyBirdThreshold:
      return config.earlyBirdDiscount
   else if hoursUntilDeparture < config.lastMinuteThreshold:
      return config.lastMinuteMultiplier
   else:
      return 1
function getOccupancyFactor(trip, config):
   occupancy = trip.bookings / trip.totalSeats
   if occupancy > 0.7:
      return config.highDemandMultiplier
   else if occupancy < 0.3:
      return config.lowDemandDiscount
   else:
      return 1
function getPeakHourFactor(trip, config):
   departureHour = trip.departureTime.hour
   if departureHour in any config.peakHours:
      return config.peakHourMultiplier
   else:
      return 1
function getVelocityFactor(trip):
   tripAge = hours since trip.createdAt
   occupancy = trip.bookings / trip.totalSeats
   if tripAge < 24 and occupancy > 0.2:
      return 1.1
   else if tripAge > 72 and occupancy < 0.2:
      return 0.95
   else:
      return 1
function calculateDynamicPrice(trip, config):
   basePrice = trip.originalPrice or trip.price
   timeFactor = getTimeFactor(trip, config)
   occupancyFactor = getOccupancyFactor(trip, config)
   peakFactor = getPeakHourFactor(trip, config)
   velocityFactor = getVelocityFactor(trip)
   finalPrice = basePrice * timeFactor * occupancyFactor * peakFactor * velocityFactor
   boundedPrice = clamp(finalPrice, basePrice * config.maxPriceDecrease, basePrice * config.maxPriceIncrease)
   return roundToNearest5(boundedPrice)
```

Figure 3.11 Pseudo Code for Dynamic Price Automation

The Dynamic Price Automation Algorithm calculates a price multiplier based on four main factors: time to departure, current occupancy, peak hour, and booking velocity. Each factor contributes to increasing or decreasing the base price. The algorithm ensures that price adjustments stay within allowed bounds, preventing unexpected price spikes or drops. It promotes early booking with discounts, maximizes revenue for popular or urgent trips, and encourages sales for underperforming ones. By automatically balancing supply and demand, the system can optimize both user affordability and operator profitability.

CHAPTER 4 IMPLEMENTATION AND TESTING

4.1 Implementation

Data was gathered in the initial phase. Data collection takes longer than the other phases. The entire project's physical design has been translated into functional computer code. The chapter before addressed a variety of techniques and technologies that were used to construct the system.

4.1.1 Tools Used

The tools that are used while designing this web application are:

Frontend: ReactJS, React Query
Backend: NodeJS, ExpressJS, Prisma ORM
Database: PostgreSQL
Documentation tools: Microsoft Word, Microsoft Excel, Microsoft PowerPoint

1. **Visual Studio Code (VS Code)**

Visual Studio Code (VS Code) as the primary source-code editor throughout the development of the project. VS Code is a lightweight, yet powerful code editor developed by Microsoft that supports extensions, syntax highlighting, IntelliSense, and integrated terminal. It improved our productivity by offering extensions such as Prettier for code formatting, Tailwind CSS IntelliSense for class name suggestions, and Prisma and React extensions for schema and component support. The integrated Git tools and terminal also made it convenient to manage version control and run development servers directly within the editor. [17]

2. **ReactJS and React Query**

In this project, we used React.js to build the frontend user interface of the Bus Ticketing System. React.js is an open-source JavaScript library developed by Facebook for building fast and interactive user interfaces. It allows us to create reusable components and manage dynamic data using state and props. With React, we built different pages such as the homepage, bus search and listing, seat selection, booking confirmation, and user profile. React's component-based architecture made the UI modular, maintainable, and scalable. [18]

We also used React Query to manage and synchronize server state with the frontend. React Query is a powerful data-fetching library for React applications that handles fetching, caching, updating, and error handling of asynchronous data. It allowed us to efficiently retrieve and manage data such as available buses, routes, booking details, and user profiles from the backend API. React Query automatically caches responses, reduces unnecessary network calls, and provides built-in support for background refetching, which enhanced performance and improved the overall user experience of the application.

3. Tailwind CSS

In this project, we used Tailwind CSS as the utility-first CSS framework to design and style the frontend components of the Bus Ticketing System. Tailwind CSS provides a set of low-level utility classes that allow developers to build responsive, customizable, and consistent user interfaces directly in the HTML or JSX markup. It helped us quickly style pages such as the homepage, bus listing, seat selection, and booking summary with minimal custom CSS. Features like responsive breakpoints, hover states, and dark mode were easily implemented, allowing for a modern and mobile-friendly design. [19]

4. Node JS & Express JS

In this project, we used Node.js as the runtime environment for building the backend server. Node.js is an open-source, event-driven JavaScript runtime that allows us to build scalable network applications. It enables asynchronous I/O operations, which makes it ideal for handling multiple simultaneous ticket booking requests in real-time. [20]

We used Express.js, a minimal and flexible Node.js web application framework, to define and manage various API routes such as user authentication, bus listing, ticket booking, and cancellation. Express provides middleware support, routing, and error-handling mechanisms, making backend development structured and maintainable. [21]

5. Prisma ORM

For interacting with the database, we used Prisma ORM. Prisma is a next-generation Object Relational Mapper that simplifies database operations by allowing developers to interact with the database using TypeScript or JavaScript. It provides a type-safe API to perform queries, updates, and deletions. Prisma was used to manage models such as User, Bus, Route, Booking, and Seat, and handle relationships between them efficiently.

6. PostgreSQL

In this project, we used PostgreSQL as the primary relational database to store and manage all application data. PostgreSQL is an open-source, powerful, and feature-rich relational database

management system (RDBMS) that supports advanced data types, complex queries, and strong data integrity. It is highly reliable and well-suited for handling structured data at scale.
We designed various relational tables in PostgreSQL such as users, buses, routes, seats, bookings, and payments. Each table stores structured data in rows and columns and maintains relationships through primary and foreign keys. For example, the bookings table includes foreign keys that link it to the users, buses, and seats tables, ensuring referential integrity.
PostgreSQL enabled us to efficiently execute SQL queries for tasks like retrieving available buses on a route, checking seat availability, recording user bookings, and tracking transaction history. Its support for indexing and joins improved the performance of complex queries and ensured fast data retrieval during high-traffic scenarios such as peak ticket booking hours.

7. Microsoft Word

Microsoft Word is a computer program that helps you create and edit documents. It's like a digital notepad where you can type letters, stories, reports, or anything you want to write. In this project, we used it for documentation purposes and report making purpose. You can change the text's appearance, like making it bold or colorful, and add pictures to your documents. It's handy for making things look neat and organized before you print them or share them with others. [22]

8. Microsoft Excel

Microsoft Excel is like a digital spreadsheet. We used excel to record timeline of the project in the form of Gantt chart. It helps you organize and calculate numbers. Imagine a table where you can put numbers and do math with them easily. You can make lists, budgets, or charts to show information in a clear way. It's great for managing finances, keeping track of data, and making things like graphs and tables.

9. Microsoft PowerPoint

Microsoft PowerPoint is like a digital tool for making presentations. It helps you create slides that show pictures, text, and even videos to explain or share something. It's like making a picture story to tell people about a topic. You can make each slide look nice and add transitions to make the presentation interesting. In our project, we used PowerPoint to create presentations.

4.1.2 Implementation details of modules (Description of Procedures/Functions)

a. Registration Module

When creating an account, the form's fields for name, email and password must all be filled out. The user is required to enter all the necessary information when filling out the input box. After filling out registration form when creating an account, user data is kept in the database. The technology allows users to log in once an account has been successfully created. While registering, the user needs to compulsorily provide username, email and password and phone number. After the user is registered, they can further process to log in.

b. Login Module

After successfully creating an account, a user can log in to the system using the login module. It has two fields, including an email and password field. Only user can access when the email address and password match with those in the database. An error message is displayed if the user enters the invalid email address and password.

c. List Buses Module

After logging into the system, the user will be able to access the List Buses module. In this module, the user can search for available buses by entering relevant criteria such as source and destination locations. The system will then display a list of buses that match the entered criteria.

For each listed bus, details such as price and available amenities (e.g., Wi-Fi, air conditioning) will be shown. The user can then select a bus to proceed. All this information is pulled from the database and displayed in real time.

d. View Seats Module

Once a user selects a bus from the List Buses module, they can access the View Seats module. This module provides the user with a visual representation of the seating layout of the selected bus.

Seats that are already booked will be marked as unavailable (often shown in grey), while available seats will be selectable by the user. The seat availability information is retrieved from the database and displayed in real time. If there are no available seats for the selected bus, an error message will inform the user.

e. Book Seat Module

After selecting a seat from the View Seats module, the user is redirected to the Book Seat module. In this module, the selected seat and bus details (e.g., seat number, bus route, departure) are displayed. The user will need to confirm their booking by reviewing the details. The displayed ticket can be downloaded also on user's device.

4.2 Testing

The different training and testing datasets are used to test the system. The purpose of this test is to determine whether the system is delivering the correct summaries. Our system is continually tested since it is in the development phase. Following are the tests that were run:

Table 4.1 Test Cases for Unit Testing

Test case ID	Test Scenario	Test Steps	Test Data	Expected Result	Pass/ Fail
T1	User Register new account	Email: shyam@gmail.com Password: shyam	Redirect to success page	As expected	Pass
T2	User enters wrong email and password	Email: shyamm@gmail.com Password: shyaam	Redirect to error page	As expected	Pass
T3	User enters right email and password	Email: shyam@gmail.com Password: shyam	Redirect to home page	As expected	Pass

Table 4.2 Search Buses

Test case ID	Test Scenario	Test Steps	Test Data	Expected Result	Pass/ Fail
T1	Search buses with pickup and destination	Pickup Destination: Kathmandu To Destination: Pokhara	List the available bus data	As expected	Pass
T2	Search buses with wrong pickup and destination	Pickup Destination: Kathmandu To Destination: Delhi	No tickets found	As expected	Pass

Table 4.3 Book a Seat

Test case ID	Test Scenario	Test Steps	Test Data	Expected Result	Pass/ Fail
T1	Book seat/s while logged in	Login and book the seats	The seat/s gets booked and updated to the database	As expected	Pass
T2	Book seat/s while logged out	Book the seat/s while logged out	The book button is not displayed to the user	As expected	Pass

Test Cases for System Testing

In the following table, different test cases have been tested in following:

Table 4.4 Test Plan for System Testing

S.N	Test Plan
1	To check if the user registration module works properly.
2	To check if the user login module works properly.
3	To check if the admin upload module works properly.
4	To check if the search module works properly.

Table 4.5 Test Case 1 for Registration Module

Test no.	1
Test case	To check if user is created or not
Expected result	Users should be created.
Actual result	User was created.
Conclusion	Expected and actual results matched.

Table 4.6 Test Case 2 for Login Module

Test no.	2
Test case	To check if the login module works.
Expected result	Users should be redirected to the Dashboard.
Actual result	User redirected to the Dashboard.
Conclusion	Expected and actual results matched.

Table 4.7 Test Case 3 for Admin Upload Module

Test no.	3
Test case	To check if upload module works
Expected result	Uploaded data should be added.
Actual result	Uploaded data added.
Conclusion	Expected and actual results matched.

Table 4.8 Test Case 4 for Search Module

Test no.	4
Test case	To check if the search module works.
Expected result	Buses should be listed according to the search address.
Actual result	Buses listed according to search address.
Conclusion	Expected and actual results matched.

Test Cases for Algorithm

In the following table, different test cases have been tested in following:

Table 4.9 Test Case for Closest Pickup Algorithm

Test no.	1
Test case	To check if closest pickup location is suggested.
Expected result	Closest pickup location should be suggested.
Actual result	Closest pickup location is suggested.
Conclusion	Expected and actual results matched.

Table 4.10 Test Case for Content Base Algorithm

Test no.	1
Test case	To check if buses are suggested according to the user preferences.
Expected result	Buses should be suggested according to the user preferences.
Actual result	Buses suggested according to the user preferences.
Conclusion	Expected and actual results matched.

CHAPTER 5 CONCLUSION AND FUTURE RECOMMENDATION

5.1 Lesson Learnt/Outcome

5.1.1 Lesson Learnt

This project's completion made it feasible to accomplish the project's objective. The user can view & search the seats through a web browser after completing the registration form. Every project forces us to grow and learn in a variety of ways. With this project, we have mastered the problem-solving skills independently, writing skills, making correct use of instructions, communicating effectively, time management, and team leadership. Most importantly, we have learnt version control tools.

5.1.1.1 Soft Skills

The soft skills we acquired include carrying out the mission, setting goals, displaying self-assurance through effective communication, having a positive outlook, participating in teamwork, problem-solving abilities, writing abilities, correctly using instructions, time management, and team leadership.

5.1.1.2 Hard Skills

This project required and strengthened several key technical skills. Frontend development was achieved using React.js, enabling the creation of a dynamic, component-based user interface, while React Query was utilized for efficient data fetching, caching, and synchronization with the backend. On the server side, Node.js and Express.js were used to build scalable RESTful APIs and handle routing, authentication, and business logic. For database interaction, Prisma ORM provided a type-safe and structured approach to querying and managing the PostgreSQL database. Additional skills such as debugging, troubleshooting, version control (with GitHub), and working with JSON APIs were essential to ensure the stability and maintainability of the full-stack application.

5.1.2 Outcomes

In the project that follows, we have mastered the problem-solving skills independently, writing skills, making correct use of instructions, communicating effectively, time management, and team leadership.

5.2 Conclusion

This project successfully delivered a full-stack Bus Ticketing System where users can search, view, and book seats in real-time through a simple and responsive web interface. The integration of modern technologies such as React.js, Node.js, Prisma, and PostgreSQL ensured the system is both scalable and user-friendly, meeting the goals of the project.

The backend efficiently manages data and API communication, while the frontend offers a seamless booking experience. By implementing features like real-time seat availability, user authentication, and dynamic search, the system not only improves operational efficiency but also enhances overall user satisfaction.

5.3 Future Recommendation

Future additions to this website can enhance several aspects, including portability and user experience. There is still work to be done, so this application might be thought of as the beginning of something much bigger. While each of these will need more time and resources to fulfill, they are all still very attainable and realistic objectives.

- Greater User Experience
- Add Forget Password Options
- Add Promo Codes

REFERENCES

[1] "ResearchGate," researchcom, January 2020. [Online]. Available: https://www.researchgate.net/publication/353793655_DESIGN_AND_IMPLEMENTATION_OF_AN_ONLINE_BUS_TICKETING_SYSTEM.

[2] S. T, "NepalDatabase," 27 April 2023. [Online]. Available: https://www.nepaldatabase.com/problem-of-bus-tickets-in-nepal.

[3] RUDLIM, "Ticketing System," Perak, 2012.

[4] al.Sharma, "Adopation and Satisfaction of Tickietng System," 2020.

[5] D. Raj, "Technological Innvoation," 2019.

[6] J. Power, Customer Satisfaction, 2021.

[7] MarketsandMarkets, "Integration of Internet," 2020.

[8] McKinsey, "Application of AI," *Comapny Analysis,* 2020.

[9] Berkeley, "Imapct of Ride Sharing Services," Berlin, 2018.

[10] P. D.K, "Security," May 2016. [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-https/.

[11] Mbarika, "Evolution of System," 2002.

[12] G. a. Bernon, "Introduction of Digital Ticketing," 2006.

[13] S. e. al., "Development of Online System," March 2014. [Online]. Available: https://www.digitalpolicy.gov.hk/en/our_work/digital_infrastructure/methodology/system_development/.

[14] K. a. Sandhu, Interviewee, *Digital Online.* [Interview]. June 2015.

[15] K. e. al., Mobile Ticketing, Anna Publisher, 2016.

[16] S. e. al., Issue of Digital Divide, 2017.

[17] "Edu," Microsoft, July 2014. [Online]. Available: https://vscodeedu.com/.

[18] Benny, "Medium," 29 Sep 2023. [Online]. Available: https://medium.com/bina-nusantara-it-division/understanding-react-query-11e56960e90c.

[19] S. D., "Intro to Tailwind," 10 Jul 2025. [Online]. Available: https://www.geeksforgeeks.org/css/introduction-to-tailwind-css/.

[20] "nodejs.org," OpenJS Foundation, [Online]. Available: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs.

[21] "Developer Mozilla," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction.

[22] Gates, "Microsoft 365," microsoft, [Online]. Available: https://www.microsoft.com/en-us/microsoft-365/word.

[23] "StackOverflow," 2019. [Online]. Available: https://stackoverflow.com/questions/56844336/what-approach-would-be-suitable-for-clustering-customers-based-on-pickup-and-dro.

[24] Henry, "Application of User Preference," 2022.

[25] "Web Docs," mozilla.org, April 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML.