BUS TICKET BOOKING SYSTEM

A PROJECT REPORT

Submitted by

SAI NITHICK ROSHAAN S (2303811724321094)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

JUNE- 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on "**Bus ticket Booking System** " is the bonafide work of **SAI NITHICK ROSHAAN S (2303811724321094)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

SIGNATURE	**SIGNATURE**
Dr.T. AVUDAIAPPAN, M.E.,Ph.D.,	Mrs.S. GEETHA, M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
ASSOCIATE PROFESSOR	ASSISTANT PROFESSOR
Department of Artificial Intelligence	Department of Artificial Intelligence
K.Ramakrishnan College of Technology (Autonomous)	K.Ramakrishnan College of Technology (Autonomous)
Samayapuram–621112.	Samayapuram–621112.

Submitted for the viva-voce examination held on …04.06.2025……………….

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

I declare that the project report on **“BUS TICKET BOOKING SYSTEM”** is the result of original work done by me and best of my knowledge, similar work has not been submitted to **“ANNA UNIVERSITY CHENNAI”** for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1221 – DATABASE MANAGEMENT SYSTEMS.**

.

Signature

.

SAI NITHICK ROSHAAN S

Place: Samayapuram

Date: 04.06.2025

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. T. AVUDAIAPPAN, M.E.,Ph.D.,** Head of the department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mrs.S.GEETHA, M.E.,** Department of **ARTIFICIAL INTELLIGENCE,** for her incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

INSTITUTE

Vision:

- To serve the society by offering top-notch technical education on par with global standards.

Mission:

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of industry and society.
- Be an institute with world class research facilities.
- Be an institute nurturing talent and enhancing competency of students to transform them as all – round personalities respecting moral and ethical values.

DEPARTMENT

Vision:

- To excel in education, innovation, and research in Artificial Intelligence and Data Science to fulfil industrial demands and societal expectations.

Mission

- To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.
- To collaborate with industry and offer top-notch facilities in a conducive learning environment.
- To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impactful research.
- To tackle the societal challenge of producing capable professionals by instilling employability skills and human values.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

- **PEO1:** Compete on a global scale for a professional career in Artificial Intelligence and Data Science.
- **PEO2:** Provide industry-specific solutions for the society with effective communication and ethics.
- **PEO3** Enhance their professional skills through research and lifelong learning initiatives.

PROGRAM SPECIFIC OUTCOMES (PSOs)

- **PSO1:** Capable of finding the important factors in large datasets, simplify the data, and improve predictive model accuracy.
- **PSO2:** Capable of analyzing and providing a solution to a given real-world problem by designing an effective program.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in

diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Bus Ticket Booking System is a user-friendly application developed using Python with Gradio as the frontend interface and SQLite as the backend database. This system is designed to simplify and digitize the process of managing buses and booking tickets, making it particularly useful for small-scale transport services or educational purposes. On the passenger side, users can book tickets by selecting an available bus, specifying the travel date, and choosing a seat number. The system automatically checks for seat availability and prevents double bookings through unique constraints on bus ID, travel date, and seat number. his system demonstrates how Python can be used to build full-stack applications efficiently with minimal external dependencies. It serves as a practical example of integrating GUI design with database operations and lays a foundation for future enhancements like user authentication, seat visualization, search filters. Overall, this project highlights the power of combining simple tools to build functional and scalable systems suitable for both learning and deployment in real-world scenarios.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD DATABASES FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The Bus Ticket Booking System is a Python-based application using Gradio for the interface and SQLite for data storage. It allows admins to add buses and passengers to book tickets with real-time seat validation. The system ensures data integrity with proper constraints and a user-friendly tab-based layout. It's ideal for small transport services or as a learning project in full-stack Python development.	PO1 -3 PO2 -3 PO3 -3 PO4 -3 PO5 -3 PO6 -2 PO7 -1 PO8 -2 PO9 -2 PO10 -2 PO11-2 PO12 -3	PSO1 -3 PSO2 -3

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	**viii**
1	**INTRODUCTION**	**1**
	1.1 OBJECTIVE	1
	1.2 OVERVIEW	1
	1.3 SQL AND DATABASE CONCEPTS	2
2	**PROJECT METHODOLOGY**	**4**
	2.1 PROPOSED WORK	3
	2.2 BLOCK DIAGRAM	4
3	**MODULE DESCRIPTION**	**5**
	3.1 ADMIN MODULE	5
	3.2 PASSENGER BOOKING MODULE	6
	3.3 BUS MANAGEMENT MODULE	6
	3.4 VIEW & REPORT MODULE	7
4	**CONCLUSION & FUTURE SCOPE**	**9**
	APPENDIX A SOURCE CODE	11
	APPENDIX B SCREENSHOTS	15
	REFERENCES	17

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The primary objective of the Bus Ticket Booking System is to develop a simplified, efficient, and user-friendly software solution for managing bus ticket bookings. In many transportation services, ticket booking is often handled manually or through fragmented systems, leading to issues such as double bookings, seat allocation errors, and inefficient record keeping. This project aims to automate the booking process using a digital platform that eliminates these challenges.

This system allows the admin to add buses with their routes and total seats, while passengers can book tickets by selecting a bus, travel date, and preferred seat number. Additionally, the system ensures no duplication of bookings and prevents double-booking of seats on the same bus for the same date. It also provides functionalities for viewing available buses, checking seat availability, and reviewing existing bookings in an organized manner.

Another major objective is to use Python and SQL (SQLite) to provide a practical example of database management integrated with a frontend UI through Gradio. This offers users and students hands-on experience in managing data with backend logic, implementing form validation, and creating interactive user interfaces for seamless ticket booking.

1.2 OVERVIEW

The Patient Appointment Scheduling System is designed as an integrated platform to streamline the management of medical appointments in clinics and healthcare centers. Built using the Gradio framework, it offers a clean, responsive, and user-friendly interface that allows easy interaction directly through Python functions. This makes it accessible for developers and students working in environments such as Google Colab. At its core, the system manages two main entities — Doctors and Appointments — stored securely in a backend database. The admin can add doctors along with their specializations, while patients can book appointments by selecting a doctor, date, and available time slot.

The system ensures no double-booking by validating appointment availability and providing real-time feedback to users.The system offers multiple functionalities to enhance usability and administration. Patients can book new appointments and view scheduled visits, while also having the option to edit existing appointments based on their unique IDs. Additionally, the system includes a database viewer feature that enables developers or administrators to inspect raw database tables, aiding in verification and debugging processes. Overall, this project demonstrates a practical integration of database management with an intuitive UI, delivering an efficient and reliable solution for appointment scheduling in healthcare settings.

1.3 SQL AND DATABASE CONCEPTS

This project extensively utilizes SQLite, a lightweight and serverless SQL database that is embedded into the application file. SQLite is highly suitable for small projects as it doesn’t require a separate server or installation and stores all data in a local file (appointments.db). It uses relational tables to store structured data.

There are two primary tables used in the system:

- **doctors(id, name, specialization):** Stores doctor information. Each doctor has a unique ID automatically generated using AUTOINCREMENT.
- **appointments(id, patient_name, doctor_id, date, time):** Stores appointment details, including the patient name, linked doctor ID, appointment date, and time.
- The foreign key relationship between the appointments and doctors tables ensures that no invalid doctor IDs can be used. SQL operations used include:
- INSERT for adding new records.
- SELECT for retrieving information such as available doctors or appointment history.

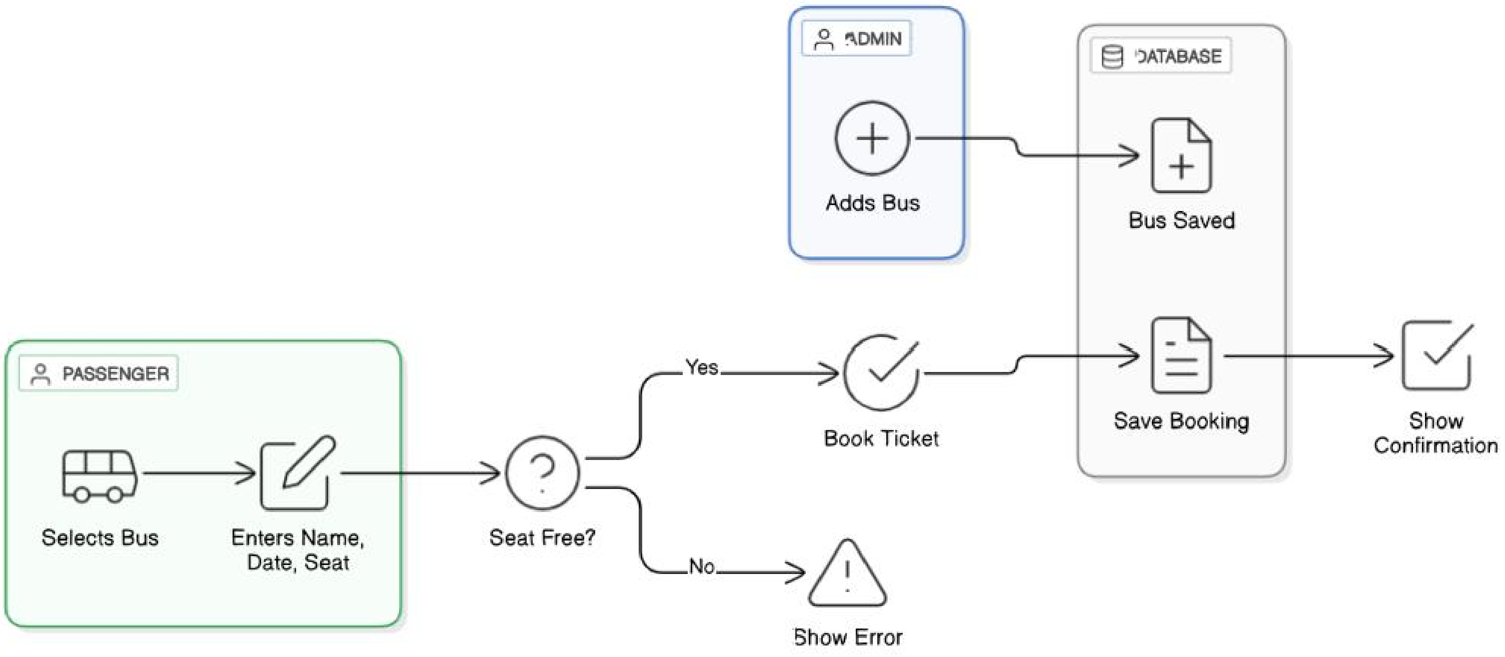
CHAPTER 2
PROJECT METHODOLOGY

2.1 PROPOSED WORK

The The development of the Patient Appointment Scheduling System follows a structured approach that emphasizes simplicity, usability, and real-time data management. The project begins with requirement analysis to identify key functionalities such as doctor management, appointment booking, time conflict resolution, and data transparency. After establishing the core requirements, the system is designed to operate entirely within a cloud-based environment using Python as the primary programming language, SQLite for database management, and Gradio for the user interface. This combination ensures that the system remains lightweight, accessible, and easy to maintain without the need for complex installations or costly infrastructure.

The implementation phase involves creating a centralized SQLite database to store information about doctors and appointments. Doctors' data, including names and specializations, are entered and managed by the administrator. Patients interact with a dynamic, interactive GUI built using Gradio, allowing them to select doctors, choose appointment dates from an auto-generated list for the upcoming week, and pick predefined time slots. Backend validation mechanisms are employed to prevent double-booking by checking if a selected time slot is already occupied for the chosen doctor on the specified date. Additionally, the system provides functionality for patients to edit their existing appointments, ensuring flexibility. To enhance transparency and ease of maintenance, a database viewer module is integrated, enabling administrators and developers to inspect raw data tables directly. Throughout development, testing and iterative improvements ensure the system performs reliably, efficiently managing appointment scheduling in real-time.

2.2 BLOCK DIAGRAM



CHAPTER 3

MODULE DESCRIPTION

The Patient Appointment Scheduling System includes modules for managing doctors, booking and editing appointments, and viewing schedules. It prevents double bookings by validating time slots and offers a database viewer for data transparency. Integrated within an easy-to-use Gradio interface, these modules work together to provide a smooth, efficient platform for managing healthcare appointments in real time.

3.1 ADMIN MODULE

- The Admin Module acts as the central control panel for managing the bus booking system. It provides authorized personnel with the ability to add new buses, view existing bus details, and monitor the underlying database for accuracy and troubleshooting.
- Add Bus: The admin enters the bus name, origin city, destination city, and total number of seats. The system validates that all fields are filled and that the seat count is a positive integer. Once validated, the information is inserted into the buses table using SQL commands.
- View Buses: This feature fetches and displays a list of all registered buses along with their routes and seating capacities. The output is presented clearly for easy understanding and quick reference.
- View Raw Database Tables: A database viewer is included to allow the admin to inspect raw data from both the buses and tickets tables. This helps in verifying records, identifying inconsistencies, and performing necessary audits or debugging.
-

3.2 PASSENGER BOOKING MODULE

This module is the core of the application, offering a straightforward and efficient interface for passengers to book bus tickets. It ensures a seamless booking experience by guiding users through each required step.

- **Passenger Information:** Passengers input their full name and contact details to personalize the booking and maintain accurate records for future reference.
- **Bus Selection:** Passengers choose a bus from the available options displayed with bus name, origin, destination, and available seats. This ensures they select the correct route and service.
- **Seat Selection and Booking:** Passengers select their preferred seats from the available ones. The system dynamically shows seat availability to avoid double bookings and maintain an updated seat count.
- **Conflict Detection:** Before confirming the booking, the system verifies that the selected seats are still available, preventing overbooking or seat conflicts.
- **Feedback and Confirmation:** Upon successful booking, the system provides a confirmation message with booking details. If issues arise, such as unavailable seats or missing information, clear error messages guide the passenger to correct the input.

3.3 BUS MANAGEMENT MODULE

The Bus Management Module plays a crucial role in organizing and maintaining all data related to bus operations, forming the foundation for an efficient transport booking system. This module is designed to streamline the process of adding, updating, and managing bus records, ensuring accurate scheduling and optimal seat allocation for passengers.

- **Bus Information Entry:** Administrators can add new buses to the system by entering essential details such as bus number, bus name, source and destination locations, total seating capacity, bus type (AC/Non-AC), and scheduled departure time. The system performs data validation checks to ensure there are no missing or incorrect entries before the details are inserted into the database using SQL commands.
- **Update Bus Details:** In the event of schedule changes, route updates, or maintenance requirements, admins can modify existing bus entries. This flexibility ensures the system reflects real-time operational data, which helps avoid confusion during booking and ensures customer satisfaction.
- **Bus Deletion:** Buses that are no longer operational or temporarily out of service can be safely removed from the database. Deletion operations are protected with confirmation prompts to minimize the risk of accidental data loss, maintaining database reliability and accuracy.
- **Data Storage and Integrity:** All bus-related data is stored using SQLite in a structured and relational format. Referential integrity is maintained between the bus records and passenger bookings to prevent inconsistencies, especially when a bus is associated with upcoming reservations.

3.4 VIEW & REPORT MODULE

The View & Report Module serves as the system's analytical interface, providing visibility into all appointment-related data. It enables users, especially admins, to track, manage, and analyze bookings efficiently while ensuring transparency in healthcare scheduling operations.

- **View Appointments:** Displays all appointments with key details like Appointment ID, Patient Name, Doctor, Date, and Time Slot in a structured format, aiding in schedule tracking and verification.
- **Filter and Search Options:** Users can filter data by doctor, date, or patient name, making it easier to retrieve relevant information quickly and manage a high volume of records.
- **Edit and Update Records:** Existing appointments can be modified using their unique ID. The system ensures updated entries do not conflict with other bookings, maintaining data integrity.
- **Database Reports and Insights:** Provides basic reports such as daily appointment counts and doctor workload summaries. These help identify patterns and optimize resources.
- **Raw Database Viewer:** Offers direct access to doctors and appointments tables, allowing admins to validate, audit, or debug the stored data efficiently.
- **Feedback and Error Handling:** Displays success or error messages for each action, guiding users through the system and confirming updates or highlighting issues when needed.

CHAPTER 4

CONCLUSION AND FUTURE SCOPE

CONCLUSION

The Patient Appointment Scheduling System developed in this project provides a robust, efficient, and user-friendly approach to automating medical appointments, especially in small to mid-level clinics. By eliminating the drawbacks of manual scheduling-such as double-booking, human errors, and fragmented data management-the system introduces streamlined appointment handling that enhances both patient satisfaction and administrative workflow.Built using **Python**, **SQLite**, and **Gradio**, the system combines backend logic with an intuitive frontend interface, enabling real-time scheduling, validation, editing, and database interaction.A significant strength of this system is its cloud-compatibility through **Google Colab**, making it accessible without additional infrastructure. It demonstrates key principles of software development such as modular programming, clean UI design, and database normalization, making it ideal for both academic purposes and real-world deployment. Moreover, this project stands as a practical implementation of theoretical knowledge in database management, and software validation. It serves as a learning model for students and developers alike by bridging academic concepts with a real-time application that solves a common real-world problem. This reinforces not just technical proficiency, but also the importance of human-centered design in software engineering.

FUTURE SCOPE :

While the current version offers core functionalities, it can be expanded to meet the evolving needs of modern clinics and healthcare systems. Potential future enhancements include:

- **User Authentication and Role Management:** Introduce secure logins with distinct roles (Admin, Doctor, Patient) to manage access and improve security.

- **Notification System:** Integrate SMS/email APIs to send appointment confirmations, reminders, and alerts to patients.

- **Doctor Portal:** Develop a dedicated dashboard for doctors to view schedules, manage availability, and update appointments.

- **Patient Health Records:** Include features to track and display medical history for each patient, aiding in long-term treatment planning.

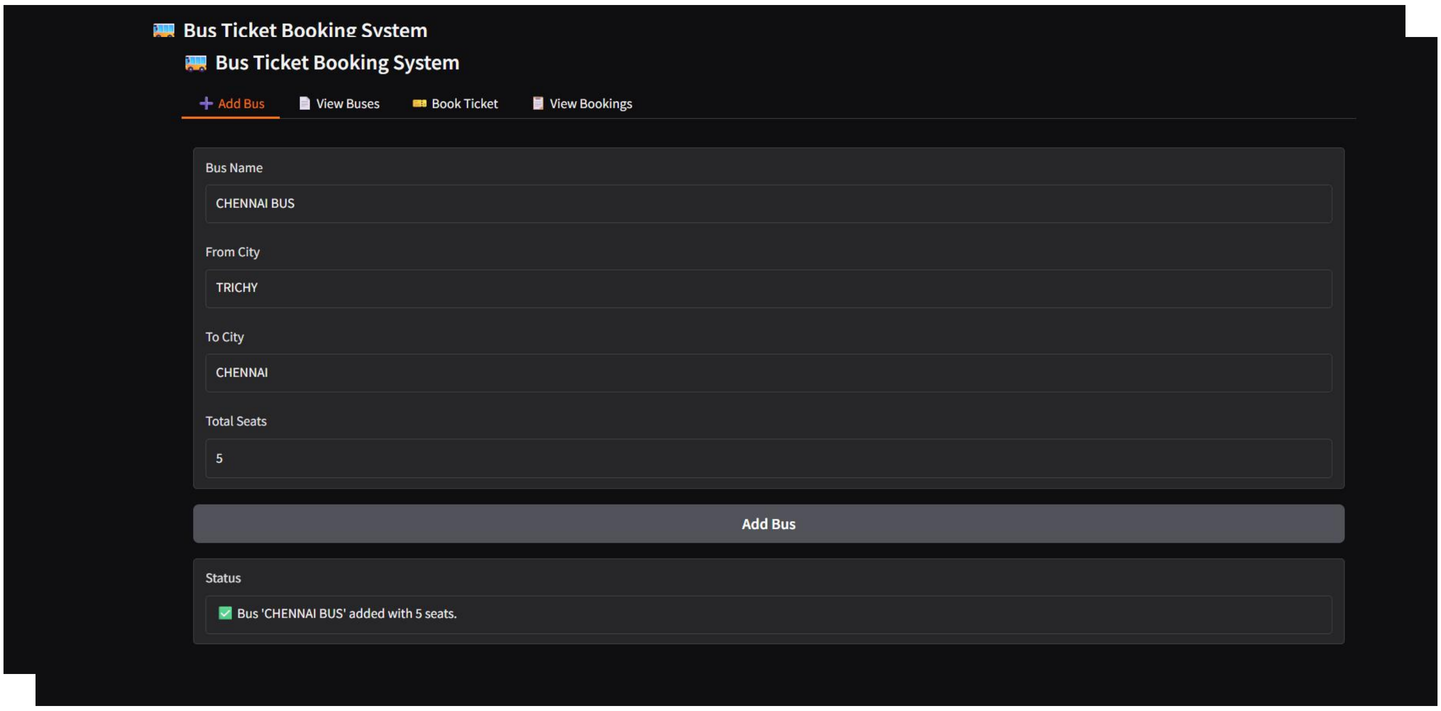
- **Data Export and Reporting:** Enable exporting of appointment lists and reports in formats like CSV or PDF for administrative use.

- **Mobile-Friendly Interface:** Transition from Gradio to a full-fledged responsive web app using Flask/Django for the backend and React for the frontend.

APPENDIX A SOURCE CODE

```
import sqlite3
import gradio as gr

# connect to a file-based SQLite DB
conn = sqlite3.connect("bus_booking.db", check_same_thread=False)
cursor = conn.cursor()

# create tables if they don't exist
cursor.execute('''
CREATE TABLE IF NOT EXISTS buses (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    bus_name TEXT NOT NULL UNIQUE,
    from_city TEXT NOT NULL,
    to_city TEXT NOT NULL,
    total_seats INTEGER NOT NULL CHECK(total_seats > 0)
)
''')
cursor.execute('''
CREATE TABLE IF NOT EXISTS tickets (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    passenger_name TEXT NOT NULL,
    bus_id INTEGER NOT NULL,
    travel_date TEXT NOT NULL,
    seat_number INTEGER NOT NULL,
    FOREIGN KEY(bus_id) REFERENCES buses(id),
    UNIQUE(bus_id, travel_date, seat_number)
)
''')
conn.commit()

# 2. Admin: add a new bus
# def add_bus(bus_name, from_city, to_city, total_seats):
#     if not (bus_name and from_city and to_city and total_seats):
#         return "❌ All fields are required."
#     try:
#         seats = int(total_seats)
#         if seats <= 0:
#             return "❌ Total seats must be a positive integer."
#         cursor.execute(
#             "INSERT INTO buses (bus_name, from_city, to_city, total_seats)
VALUES (?, ?, ?, ?)",
#             (bus_name.strip(), from_city.strip(), to_city.strip(), seats)
#         )
#         conn.commit()
#         return f"✅ Bus '{bus_name}' added with {seats} seats."
#     except ValueError:
#         return "❌ Total seats must be an integer."
#     except sqlite3.IntegrityError:
#         return "❌ That bus name already exists."

def add_bus(bus_name, from_city, to_city, total_seats):
    if not (bus_name and from_city and to_city):
        return "❌ All fields are required."

    try:
        seats = int(float(total_seats))  # Converts from float (from gr.Number) to
int
        if seats <= 0:
            return "❌ Total seats must be a positive integer."

        cursor.execute(
            "INSERT INTO buses (bus_name, from_city, to_city, total_seats)
VALUES (?, ?, ?, ?)",
            (bus_name.strip(), from_city.strip(), to_city.strip(), seats)
        )
        conn.commit()
        return f"✅ Bus '{bus_name}' added with {seats} seats."
    except (ValueError, TypeError):
        return "❌ Total seats must be a valid number."
    except sqlite3.IntegrityError:
        return "❌ That bus name already exists."


# helper to get list of buses for dropdown
def get_bus_choices():
    cursor.execute("SELECT bus_name FROM buses ORDER BY bus_name")
    return [row[0] for row in cursor.fetchall()]

# 3. Passenger: book a ticket
def book_ticket(passenger_name, bus_name, travel_date, seat_number):
    if not (passenger_name and bus_name and travel_date and seat_number):
        return "❌ All fields are required."
    try:
        seat = int(seat_number)
    except ValueError:
        return "❌ Seat number must be an integer."

    cursor.execute("SELECT id, total_seats FROM buses WHERE bus_name
= ?", (bus_name,))
    bus = cursor.fetchone()
    if not bus:
        return "❌ Selected bus not found."
    bus_id, total_seats = bus
    if not (1 <= seat <= total_seats):
        return f"❌ Seat must be between 1 and {total_seats}."

    try:
        cursor.execute(
            "INSERT INTO tickets (passenger_name, bus_id, travel_date,
seat_number) VALUES (?, ?, ?, ?)",
            (passenger_name.strip(), bus_id, travel_date.strip(), seat)
        )
        conn.commit()
        return f"✅ Ticket booked on '{bus_name}' for {travel_date}, Seat
#{seat}."
    except sqlite3.IntegrityError:
        return "❌ That seat is already booked on that date."

# 4. View functions
def view_buses():
    buses = cursor.execute("SELECT bus_name, from_city, to_city, total_seats
FROM buses").fetchall()
    if not buses:
        return "No buses available."
    return "\n".join([
        f"{b[0]}: {b[1]} → {b[2]} ({b[3]} seats)"
        for b in buses
    ])

def view_bookings():
    rows = cursor.execute('''
        SELECT t.id, t.passenger_name, b.bus_name, t.travel_date, t.seat_number
        FROM tickets t
        JOIN buses b ON t.bus_id = b.id
        ORDER BY t.travel_date, b.bus_name, t.seat_number
    ''').fetchall()
    if not rows:
        return "No bookings yet."
    return "\n".join([
        f"#{r[0]} {r[1]} → {r[2]} on {r[3]} (Seat {r[4]})"
        for r in rows
    ])

# 5. Build Gradio interface
with gr.Blocks() as app:
    gr.Markdown("##     Bus Ticket Booking System")

    # Admin tab
    with gr.Tab("➕ Add Bus"):
        name_in  = gr.Text(label="Bus Name")
        from_in  = gr.Text(label="From City")
        to_in    = gr.Text(label="To City")
        seats_in = gr.Number(label="Total Seats", precision=0)
        add_btn  = gr.Button("Add Bus")
        add_out  = gr.Textbox(label="Status")
        add_btn.click(add_bus, [name_in, from_in, to_in, seats_in], add_out)

    # View buses tab
    with gr.Tab("    View Buses"):
        show_buses = gr.Button("Show All Buses")
        buses_out  = gr.Textbox(label="Available Buses", lines=8)
        show_buses.click(view_buses, None, buses_out)

    # Booking tab
    with gr.Tab("    Book Ticket"):
        passenger_in = gr.Text(label="Passenger Name")
        bus_dd       = gr.Dropdown(choices=get_bus_choices(), label="Select Bus")
        refresh_btn  = gr.Button("    Refresh Bus List")
        refresh_btn.click(lambda: gr.update(choices=get_bus_choices()), None,
bus_dd)

        date_in      = gr.Text(label="Travel Date (YYYY-MM-DD)")
        seat_in      = gr.Number(label="Seat Number", precision=0)
        book_btn     = gr.Button("Book Ticket")
        book_out     = gr.Textbox(label="Status")
        book_btn.click(book_ticket,
                       [passenger_in, bus_dd, date_in, seat_in],
                       book_out)

        show_bookings = gr.Button("Show All Bookings")
        book_list     = gr.Textbox(label="Bookings", lines=10)
        show_bookings.click(view_bookings, None, book_list)
    app.launch()
```

APPENDIX B SCREENSHOTS



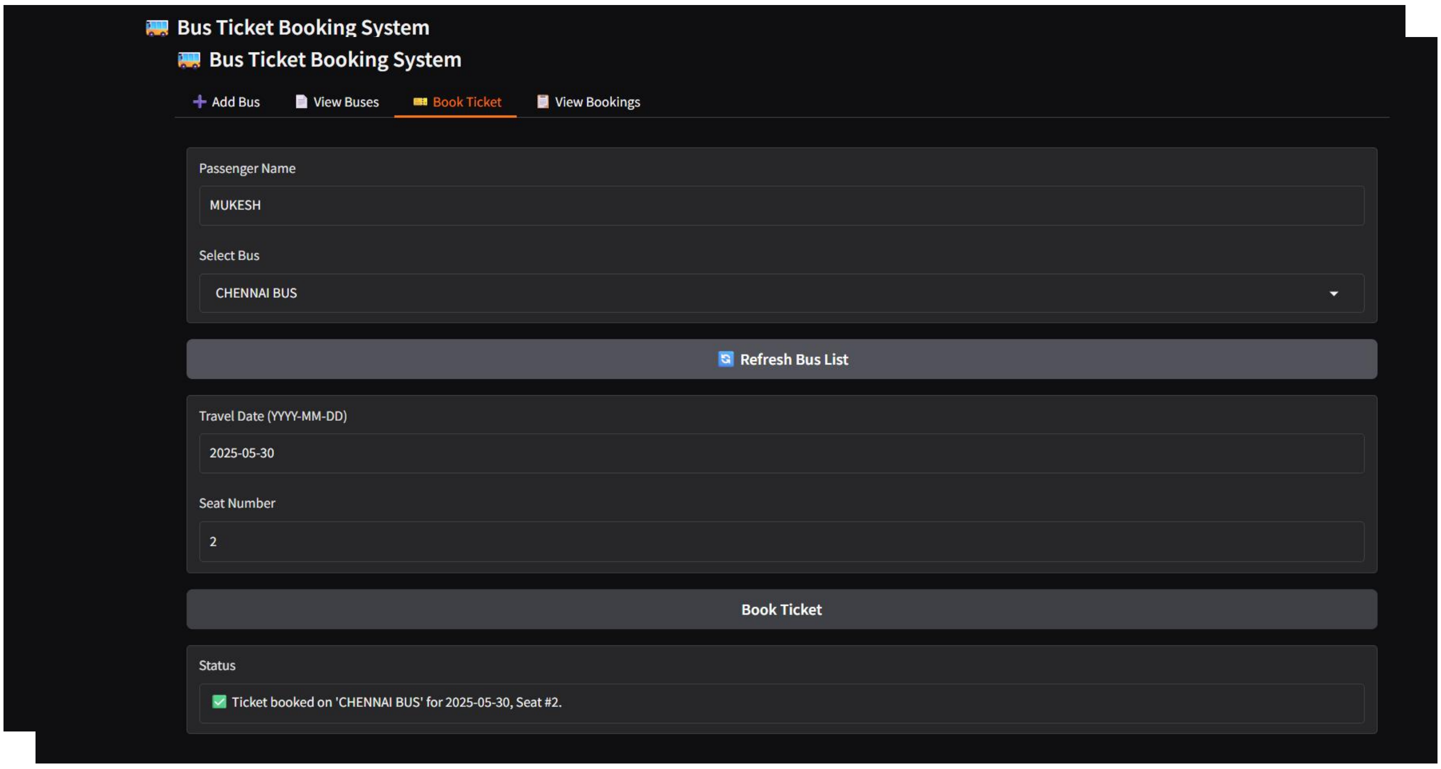
Bus Name: CHENNAI BUS

From City: TRICHY

To City: CHENNAI

Total Seats: 5

Status: ✓ Bus 'CHENNAI BUS' added with 5 seats.



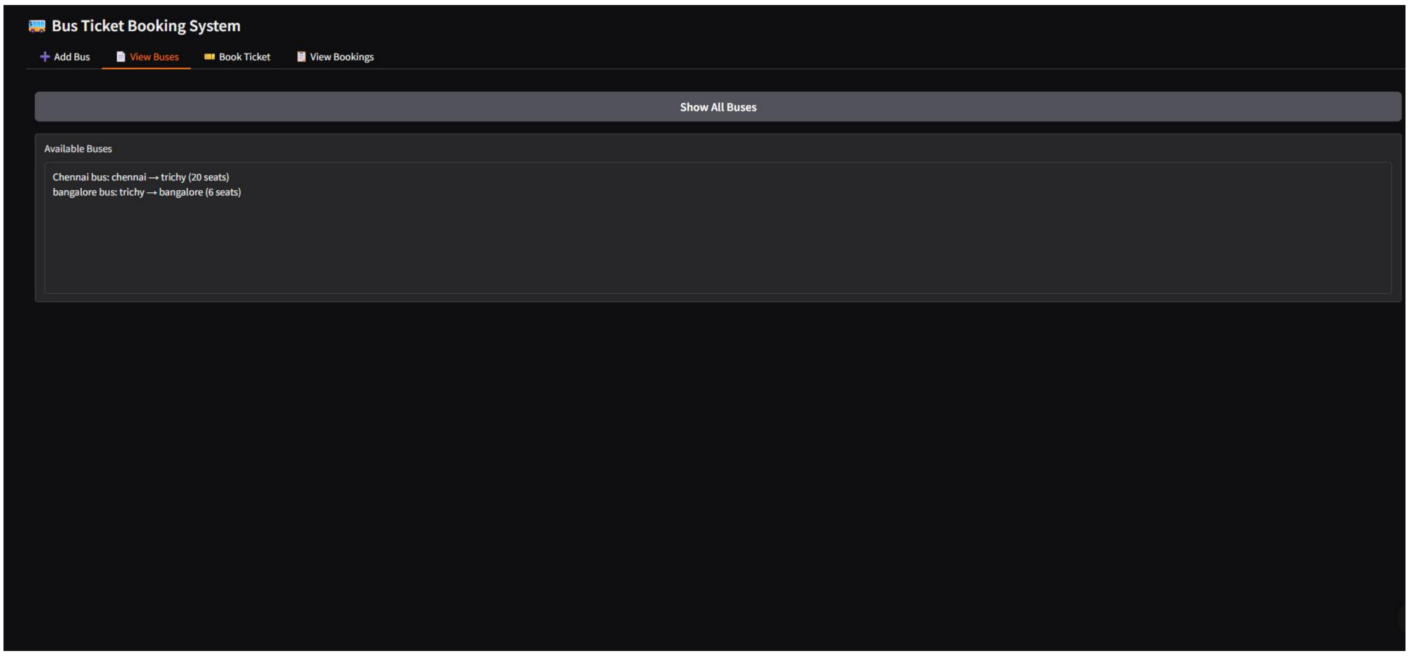
Passenger Name: MUKESH

Select Bus: CHENNAI BUS

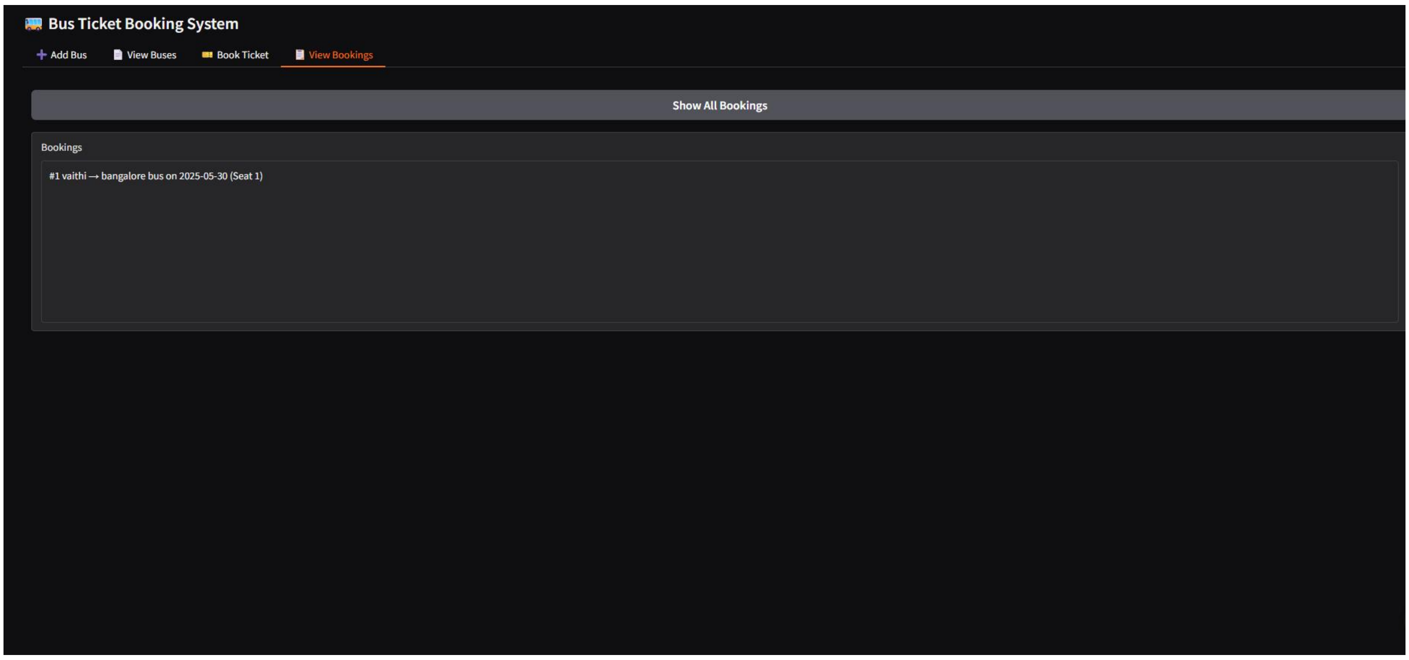
Travel Date (YYYY-MM-DD): 2025-05-30

Seat Number: 2

Status: ✓ Ticket booked on 'CHENNAI BUS' for 2025-05-30, Seat #2.



```
Chennai bus: chennai → trichy (20 seats)
bangalore bus: trichy → bangalore (6 seats)
```



```
#1 vaithi → bangalore bus on 2025-05-30 (Seat 1)
```

REFERENCES

1. Python Official Documentation
   https://docs.python.org/3/
   → Used for understanding core Python syntax, built-in functions, and modules for backend logic.
2. SQLite Official Documentation
   https://sqlite.org/docs.html
   → Referred for database creation, querying techniques, and data management using SQL.
3. Gradio Documentation
   https://www.gradio.app/docs/
   → Used to build the interactive user interface and connect Python functions to the frontend.
4. Stack Overflow
   https://stackoverflow.com/
   → Utilized for solving development issues, debugging errors, and finding community-verified solutions.
5. GeeksforGeeks – Python & SQLite Tutorials
   https://www.geeksforgeeks.org/
   → Helped in understanding the integration of SQLite with Python for building CRUD functionalities.
6. Real Python – SQLite with Python
   https://realpython.com/python-sqlite-sqlalchemy/
   → Provided best practices for SQLite usage in Python applications.
7. GitHub – Open Source Project Repositories
   https://github.com/
   → Referenced for architecture ideas, coding patterns, and database schema inspiration.

8. W3Schools – Python Tutorial
   https://www.w3schools.com/python/
   → Used to refresh Python fundamentals and explore examples of data handling and control structures.
9. UI/UX Design Principles – NNGroup & Smashing Magazine
   https://www.nngroup.com/articles/
   → Consulted to understand best practices for creating user-friendly, minimal, and responsive interfaces for improved usability.
10. YouTube – Project-Based Learning (Gradio & Python Integration)
   https://www.youtube.com/
   → Referred for tutorials on implementing real-time appointment systems, Gradio deployment, and practical UI workflows.