iRTS-RAILWAY RESERVATION SYSTEM

## A MINI PROJECT REPORT

**Submitted By**

# RAKHUL PRAKASH S B (220701216) SABARISH M (220701234)

In partial fulfillment for the award of the degree of BACHELOR OF ENGINEERING

IN COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS) THANDALAM

CHENNAI-602105

2023-2024

**BONAFIDE CERTIFICATE**

Certified that this project report “**RAILWAY RESERVATION SYSTEM**” is the bonafide work of “**RAKHUL PRAKASH SB (220701216) , SABARISH M(220701234)**” who carried out

the project work under my supervision.

Submitted for practical examination held on

INTERNAL EXAMINER EXTERNAL EXAMINER

# ABSTRACT

This project presents the development of a Railway Ticket Booking System using Python, Streamlit, and MySQL, with a particular emphasis on the application of database normalization concepts. The primary objective of this system is to provide a streamlined and efficient platform for booking railway tickets, while ensuring the integrity and optimal organization of the underlying database.

The system leverages Streamlit, a modern web framework, to create an intuitive and user-friendly interface that facilitates interactions between users and the backend database.

Python is utilized for backend development, ensuring robust and scalable application logic. MySQL serves as the relational database management system, chosen for its reliability and performance.

A key focus of the project is the implementation of database normalization techniques to eliminate redundancy and improve data integrity. By applying normalization rules the database schema is optimized to reduce data anomalies, ensuring consistent and reliable data storage.

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S.No** | **CONTENTS** | **Pg No.** |
| **1** | **INTRODUCTION** | **5** |
| 1.1 | INTRODUCTION | 5 |
| 1.2 | OBJECTIVES | 5 |
| 1.3 | MODULES | 5 |
| **2** | **SURVEY OF TECHNOLOGIES** | **7** |
| 2.1 | SOFTWARE DESCRIPTION | 7 |
| 2.2 | LANGUAGES | 7 |
| 2.2.1 | PYTHON | 7 |
| 2.2.2 | SQL | 8 |
| **3** | **REQUIREMENTS AND ANALYSIS** | **9** |
| 3.1 | REQUIREMENT SPECIFICATION | 9 |
| 3.2 | HARDWARE AND SOFTWARE REQUIREMENTS | 12 |
| 3.3 | ARCHITECTURE DIAGRAM | 12 |
| 3.4 | ER DIAGRAM | 15 |
| 3.5 | NORMALIZATION | 15 |
| **4** | **PROGRAM CODE** | **19** |
| **5** | **RESULTS AND DISCUSSION** | **62** |
| **6** | **TESTING** | **71** |
| **7** | **CONCLUSION** | **72** |
| **8** | **REFERENCES** | **73** |
|  |  |  |

* 1. **INTRODUCTION :**

# INTRODUCTION

The Railway Ticket Booking System is an application designed to facilitate the process of booking, viewing, and canceling railway tickets. Utilizing Python, Streamlit, and MySQL, this system aims to provide a user-friendly interface and a robust backend that ensures efficient and secure operations. By integrating database normalization techniques, the system achieves optimal data management, reducing redundancy and maintaining data integrity. This project not only streamlines the ticket booking process for users but also offers a comprehensive administrative interface for managing train schedules, fares, and user data.

## OBJECTIVES :

The primary objective of the Railway Ticket Booking System is to develop a reliable and efficient platform for users to book railway tickets online. The system aims to:

* + 1. Provide a seamless and intuitive user interface for ticket booking, viewing, and cancellation.
    2. Ensure data integrity and optimal database performance through the application of normalization principles.
    3. Offer secure and scalable solutions for handling user data and transactions.
    4. Enable administrative control over train schedules, fare management, and user management.
    5. Enhance the overall user experience by minimizing booking errors and simplifying the ticket management process

## MODULES :

### User Registration and Authentication:

* + - * **Sign-Up**: Allows new users to create accounts.
      * **Login**: Enables existing users to log in securely.
      * **Password Management**: Provides functionalities for password recovery and changes.

o

### Ticket Booking:

* + - * **Search Trains**: Users can search for trains based on source, destination, and travel dates.
      * **Select Seats**: Users can view and select available seats.
      * **Calculate Fare**: Displays the total fare based on selected options.
      * **Payment Processing**: Integrates with payment gateways for secure transactions.

### View Ticket:

* + - * **Booking Confirmation**: Displays booking confirmation details including train information and seat numbers.
      * **Ticket Details**: Allows users to view detailed information about their booked tickets.

### Cancel Ticket:

* + - * **Cancellation Request**: Users can request ticket cancellations.
      * **Refund Processing**: Handles refunds according to cancellation policies.

### Admin Panel:

* + - * **Manage Train Schedules**: Admins can add, update, or remove train schedules.
      * **Manage Fares**: Admins can set and update fare information.
      * **User Management**: Admins can view and manage user accounts and their activities.

### Database Management:

* + - * **Normalized Schema**: Ensures the database is normalized up to the third normal form (3NF).
      * **Data Integrity**: Maintains consistent and reliable data storage.
      * **Scalability**: Ensures the database can handle large volumes of data efficiently.

This modular approach ensures a comprehensive and organized system, allowing for ease of use, efficient management, and robust performance

# 2.SURVEY OF TECHNOLOGY

## SOFTWARE DESCRIPTION :

### Visual studio Code

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging. First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit-build-debug cycle means less time fiddling with your environment, and more time executing on your ideas.

## LANGUAGES :

* + 1. **PYTHON :**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum. It was originally released in 1991. Designed to be easy as well as fun, the name "Python" is a nod to the British comedy group Monty Python. Python has a reputation as a beginner-friendly language, replacing Java as the most widely used introductory language because it handles much of the complexity for the user, allowing beginners to focus on fully grasping programming concepts rather than minute details.

Python is used for server-side web development, software development, mathematics, and system scripting, and is popular for Rapid Application Development and as a scripting or glue language to tie existing components because of its high-level, built-in data structures, dynamic typing, and dynamic binding. Program maintenance costs are reduced with Python due to the easily learned syntax and emphasis on readability. Additionally, Python's support of modules and packages facilitates modular programs and reuse of code. Python is an open source community language, so numerous independent programmers are continually building libraries and functionality for it.

## MYSQL :

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation. A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

# 2. REQUIREMENT AND ANALYSIS

## REQUIREMENTS SPECIFICATION :

**User Requirements :**

### User Registration and Authentication:

* + - Users must be able to create an account with a username, password, and email address.
    - Users must be able to log in securely using their credentials.
    - Users should have the option to recover or reset their password if forgotten.

### Ticket Booking:

* + - Users should be able to search for available trains by entering source, destination, and travel date.
    - Users must be able to view train schedules and seat availability.
    - Users should be able to select their preferred seats.
    - The system should calculate and display the total fare based on the selected train and class.
    - Users must be able to make payments securely through integrated payment gateways.

### View Ticket:

* + - Users should be able to view details of their booked tickets, including train number, travel date, departure and arrival times, and seat number.
    - Users must receive a booking confirmation upon successful ticket booking.

### Cancel Ticket:

* + - Users should be able to request ticket cancellations.
    - The system must process refunds according to the defined cancellation policy and display the refund status to the user.

### User Profile Management:

* + - Users must be able to update their personal information, such as name, contact number, and email address.
    - Users should have access to their booking history and details

## System Requirements :

### Functional Requirements

1. **User Management**:
   * The system must handle user registrations, logins, and profile management.
   * The system must ensure secure authentication and authorization processes.

### Train and Schedule Management:

* + The system should allow administrators to add, update, or remove train schedules.
  + The system should display real-time seat availability and fare information to users.

### Ticket Booking:

* + The system must provide a search functionality for users to find trains based on specific criteria.
  + The system must support seat selection and fare calculation.
  + The system must integrate with payment gateways for processing payments securely.

### Ticket Viewing and Cancellation:

* + The system should allow users to view their booked tickets and related details.
  + The system must support ticket cancellation requests and process refunds as per the defined policy.

### Admin Panel:

* + The system must provide administrative tools for managing train schedules, fares, and user accounts.
  + The system should enable administrators to monitor and manage user activities and bookings.

### Non-Functional Requirements

1. **Performance**:
   * The system should be able to handle multiple concurrent users without significant performance degradation.
   * The database queries should be optimized for fast data retrieval and update operations.

### Security:

* + The system must ensure data security through encryption, secure authentication, and authorization mechanisms.
  + The system should comply with relevant data protection regulations to safeguard user information.

### Scalability:

* + The system should be scalable to accommodate increasing numbers of users and transactions over time.

### Usability:

* + The system should have an intuitive and user-friendly interface, making it easy for users to navigate and perform tasks.
  + The user interface should be responsive and accessible on various devices, including desktops, tablets, and smartphones.

### Reliability:

* + The system should ensure high availability with minimal downtime.
  + The system should have robust error handling and recovery mechanisms to maintain data integrity and system stability.

### Maintainability:

* + The system should be designed with modularity and clean code practices to facilitate easy maintenance and updates.
  + Comprehensive documentation should be provided for both users and developers to ensure smooth operation and future enhancements.

## HARDWARE AND SOFTWARE REQUIREMENTS :

### Software Requirements :

* + Operating System Windows 10
  + Front End HTML , CSS , javascript
  + Back End PHP, MySQL

### Hardware Requirements :

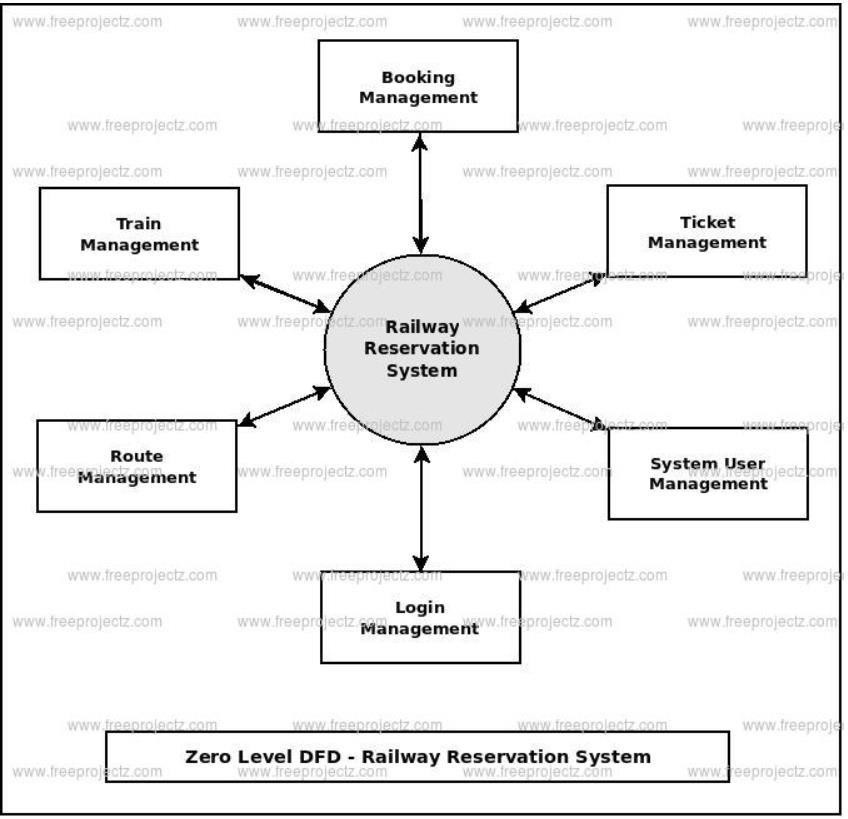
* Desktop PC or a Laptop
* Printer
* Operating System – Windows 10
* Intel® CoreTM i3-6006U CPU @ 2.00GHz
* 4.00 GB RAM
* 64-bit operating system, x64 based processor
* 1024 x 768 monitor resolution
* Keyboard and Mouse

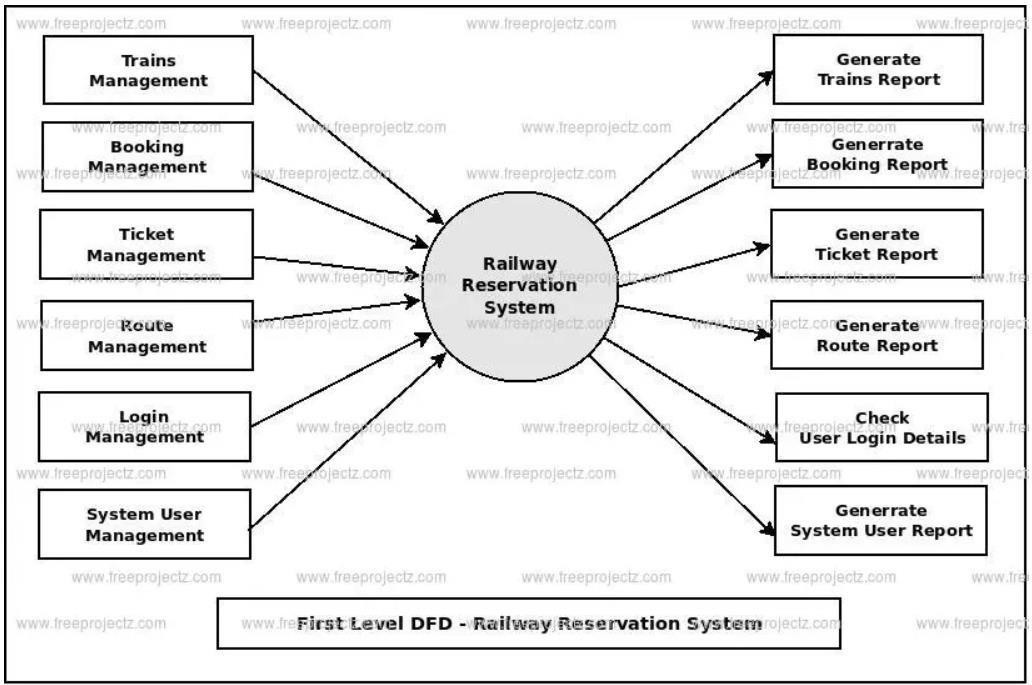
## DATA FLOW DIAGRAM :

DFD is an important tool used by system analysis. A data flow diagram model, a system using external entities from which data flows through a process which transforms the data and creates output data transforms which go to other processes external entities such as files. The main merit of DFD is that it can provide an overview of what data a system would process.

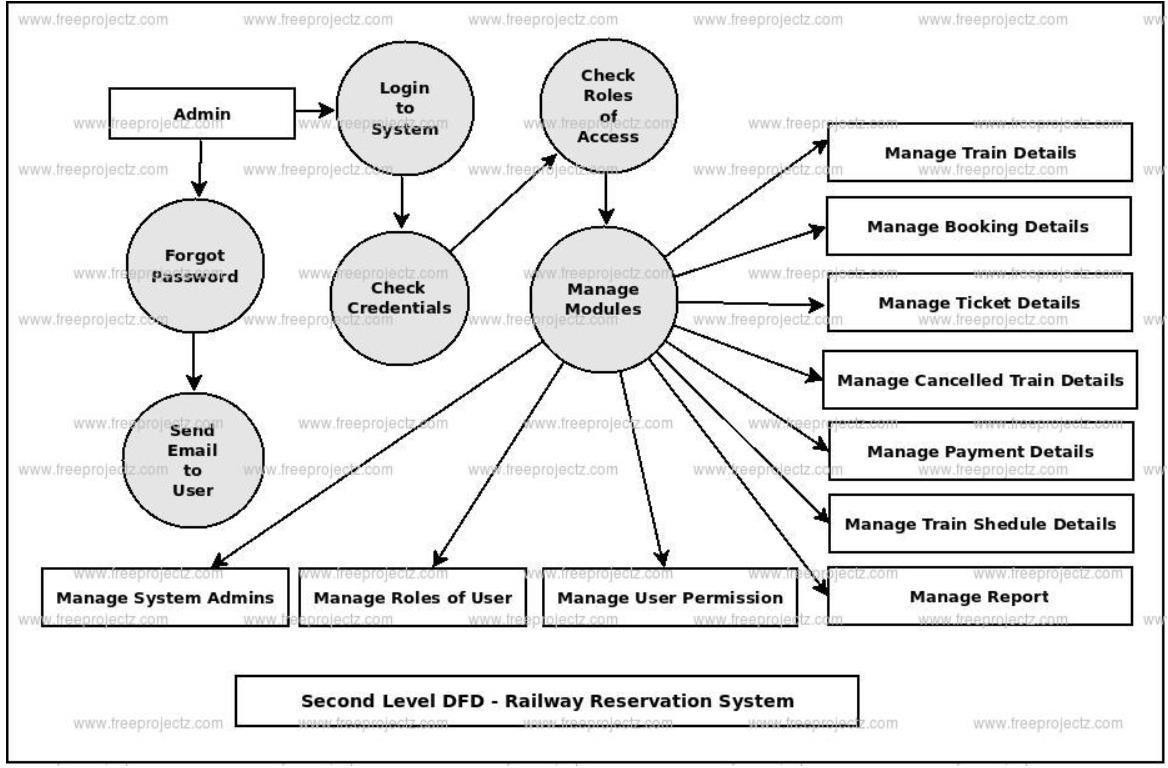
* + A data-flow diagram is a way of representing a flow of data through a process or a system.
  + The DFD also provides information about the outputs and inputs of each entity and the process itself.

## ZERO LEVEL :

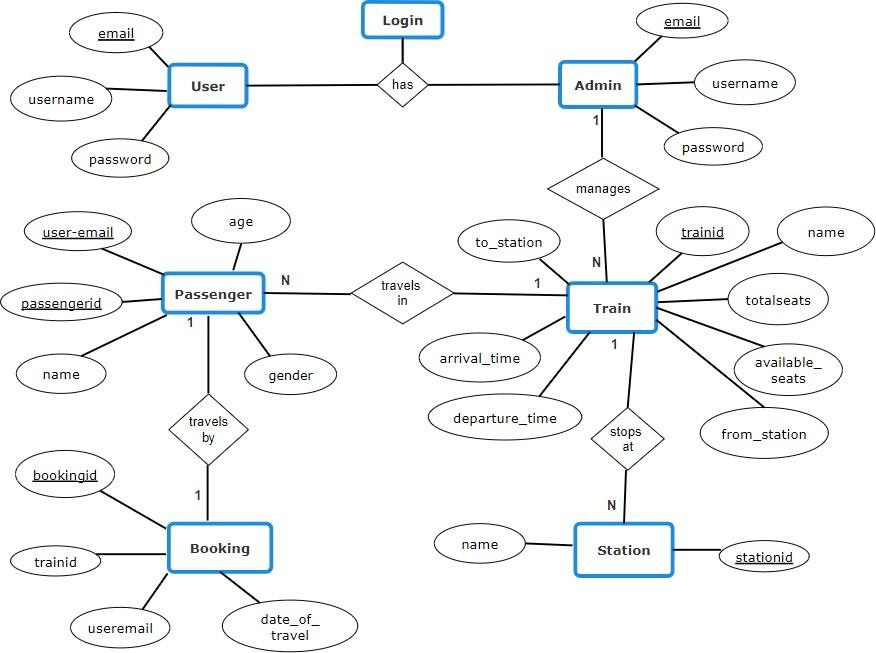


**FIRST LEVEL :**

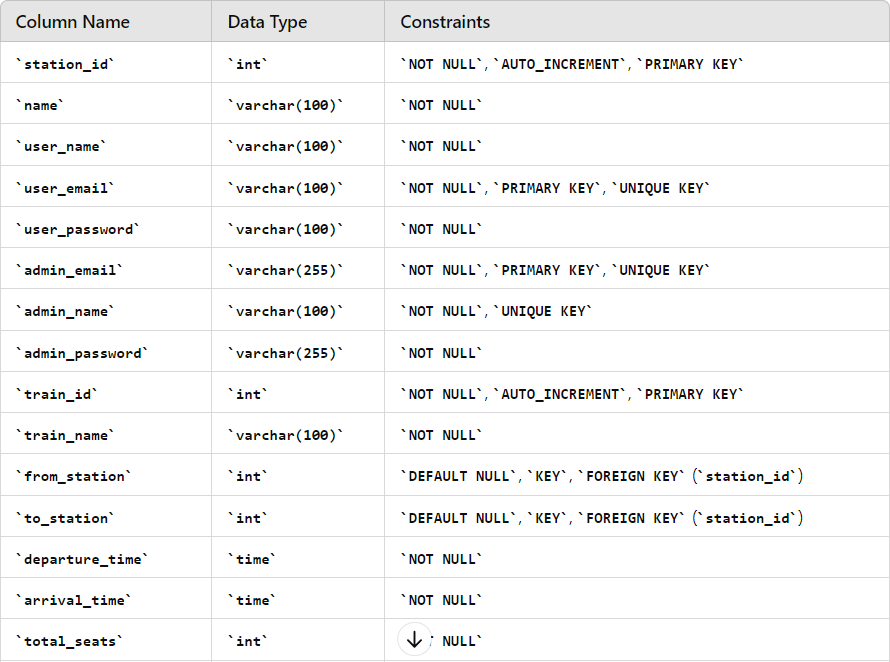
## SECOND LEVEL :

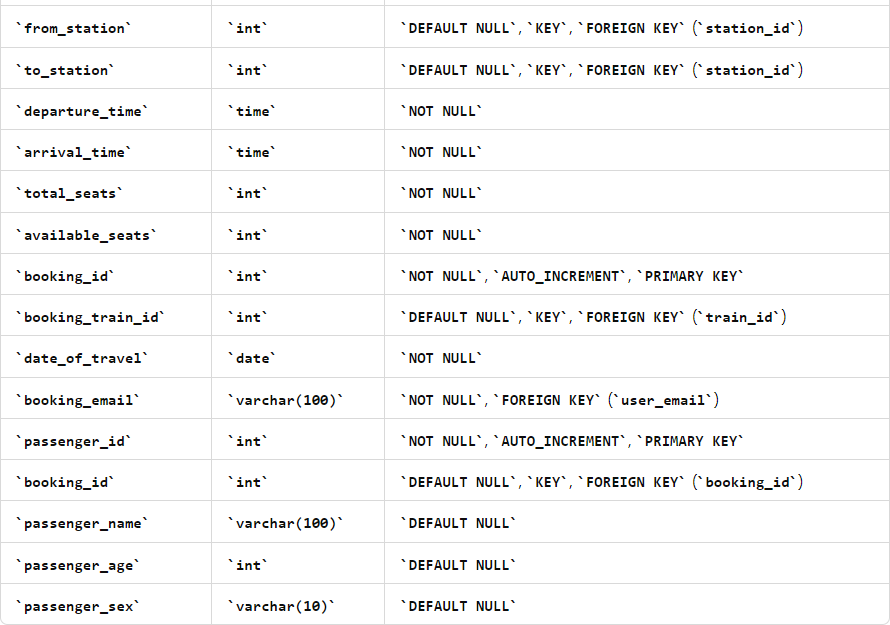


* 1. **ER DIAGRAM :**

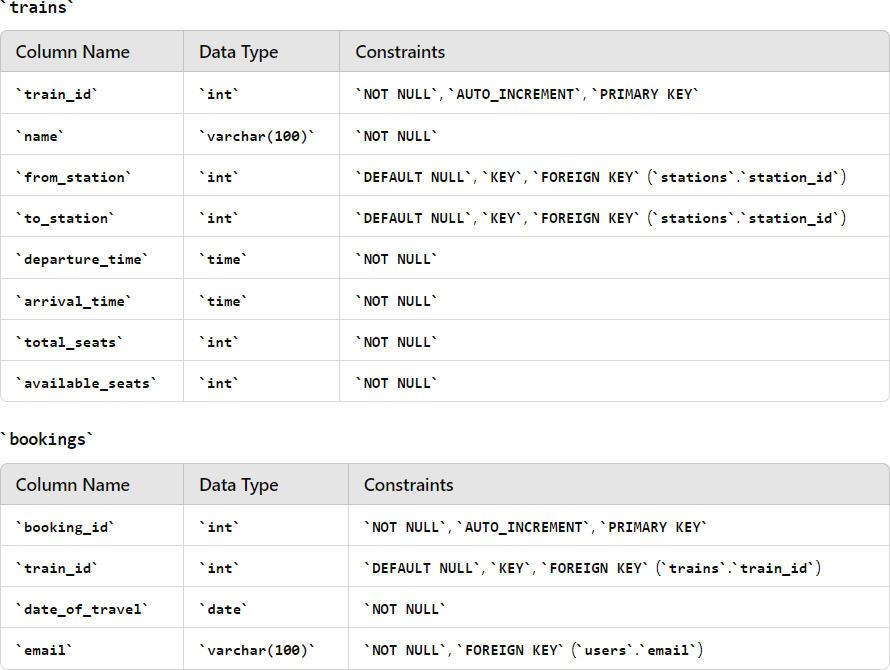
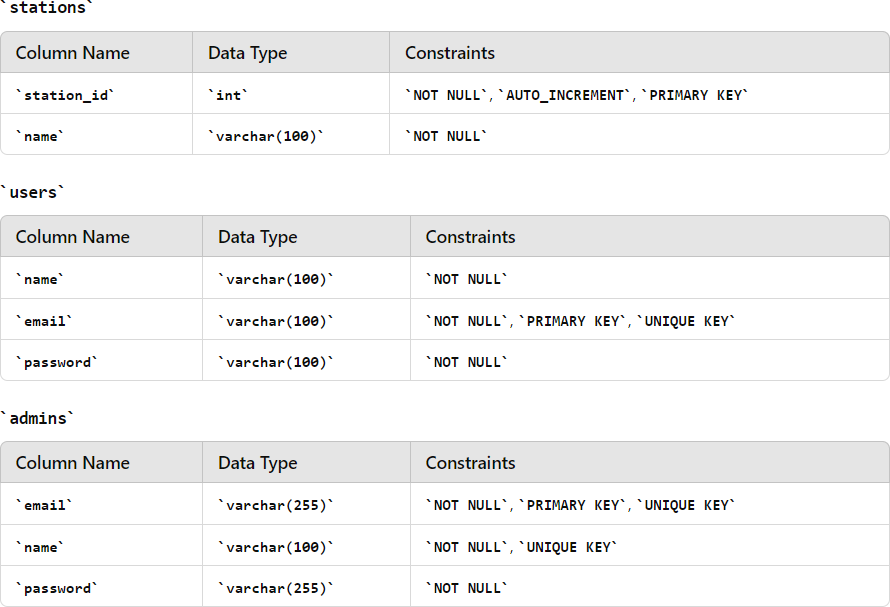


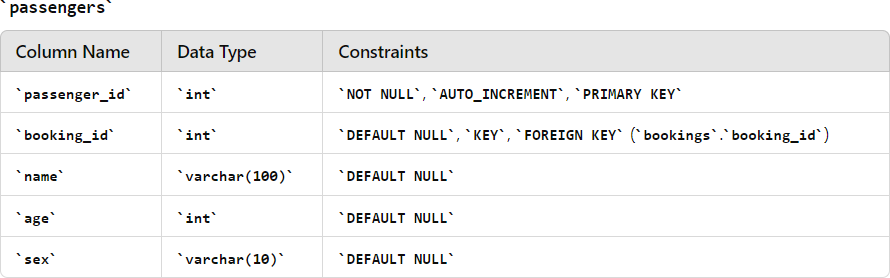
## NORMALIZATION : RAW DATABASE :





**NORMALIZED DATABASE :**





# PROGRAM CODE

## 4.1 CODE DETAILS AND CODE EFFICIENCY :

import streamlit as st

from streamlit\_option\_menu import option\_menu

from pages.userPages import userBookTicket, userContactUs, userHome, userViewTickets, userAboutUs

from pages.onBoarding\_and\_authentication import login, signup, onboardingLogin

from pages.adminPages import adminGuide, adminHome, adminSettings, adminManageTrains, adminManageStations

st.set\_page\_config(layout="wide")

def status\_shower(email, username, status):

text = f"""<button id="btn-message" class="button-message">

<div class="content-avatar">

<div class="status-user"></div>

<div class="avatar">

<svg class="user-img" xmlns="[http://www.w3.org/2000/svg"](http://www.w3.org/2000/svg) viewBox="0 0 24 24"><path d="M12,12.5c-3.04,0-5.5,1.73- 5.5,3.5s2.46,3.5,5.5,3.5,5.5-1.73,5.5-3.5-2.46-3.5-5.5-3.5Zm0-.5c1.66,0,3-

1.34,3-3s-1.34-3-3-3-3,1.34-3,3,1.34,3,3,3Z"></path></svg>

</div>

</div>

<div class="notice-content">

<div class="username">{username}</div>

<div class="lable-message">{status}<span class="number- message"></span></div>

<div class="user-id">{email}</div>

</div>

</button>"""

if status == "Logged in": online\_status = "#00da00"

else:

online\_status = "#880808"

css = f"""<style> #btn-message {{

--text-color: rgb(255, 255, 255);

--bg-color-sup: #5e5e5e;

--bg-color: #000000;

--bg-hover-color: #161616;

--online-status: {online\_status};

--font-size: 16px;

--btn-transition: all 0.2s ease-out;

}}

@media (prefers-color-scheme: dark) {{ #btn-message {{

--text-color: var(--text-color-dark);

--bg-color-sup: var(--bg-color-sup-dark);

--bg-color: var(--bg-color-dark);

--bg-hover-color: var(--bg-hover-color-dark);

}}

}}

@media (prefers-color-scheme: light) {{ #btn-message {{

--text-color: var(--text-color-light);

--bg-color-sup: var(--bg-color-sup-light);

--bg-color: var(--bg-color-light);

--bg-hover-color: var(--bg-hover-color-light);

}}

}}

.button-message {{ display: flex;

justify-content: center; align-items: center;

font: 400 var(--font-size) Helvetica Neue, sans-serif;

box-shadow: 0 0 2.17382px rgba(0,0,0,.049),0 1.75px 6.01034px rgba(0,0,0,.07),0 3.63px 14.4706px rgba(0,0,0,.091),0 22px 48px rgba(0,0,0,.14);

background-color: var(--bg-color); border-radius: 68px;

cursor: pointer;

padding: 6px 10px 6px 6px; width: fit-content;

height: 40px; border: 0;

margin: 10px 10px 10px 60px; overflow: hidden;

position: relative;

transition: var(--btn-transition);

}}

.button-message:hover {{ height: 48px;

padding: 8px 20px 8px 8px;

background-color: var(--bg-hover-color); transition: var(--btn-transition);

}}

.button-message:active {{ transform: scale(0.99);

}}

.content-avatar {{ width: 30px; height: 30px; margin: 0;

transition: var(--btn-transition); position: relative;

}}

.button-message:hover .content-avatar {{ width: 40px;

height: 40px;

}}

.avatar {{ width: 100%;

height: 100%;

border-radius: 50%; overflow: hidden;

background-color: var(--bg-color-sup);

}}

.user-img {{ width: 100%;

height: 100%; object-fit: cover;

}}

.status-user {{ position: absolute; width: 6px; height: 6px;

right: 1px; bottom: 1px;

border-radius: 50%;

outline: solid 2px var(--bg-color); background-color: var({online\_status}); transition: var(--btn-transition);

animation: active-status 2s ease-in-out infinite;

}}

.button-message:hover .status-user {{ width: 10px;

height: 10px; right: 1px; bottom: 1px;

outline: solid 3px var(--bg-hover-color);

}}

.notice-content {{ display: flex;

flex-direction: column; align-items: flex-start; justify-content: center; padding-left: 8px;

text-align: initial; color: var(--text-color);

}}

.username {{

letter-spacing: -6px; height: 0;

opacity: 0;

transform: translateY(-20px); transition: var(--btn-transition);

}}

.user-id {{

font-size: 12px; letter-spacing: -6px; height: 0;

opacity: 0;

transform: translateY(10px); transition: var(--btn-transition);

}}

.lable-message {{ display: flex;

align-items: center; opacity: 1; transform: scaleY(1);

transition: var(--btn-transition);

}}

.button-message:hover .username {{ height: auto;

letter-spacing: normal;

opacity: 1;

transform: translateY(0); transition: var(--btn-transition);

}}

.button-message:hover .user-id {{ height: auto;

letter-spacing: normal; opacity: 1;

transform: translateY(0); transition: var(--btn-transition);

}}

.button-message:hover .lable-message {{ height: 0;

transform: scaleY(0); transition: var(--btn-transition);

}}

.lable-message, .username {{ font-weight: 600;

}}

.number-message {{ display: flex;

justify-content: center; align-items: center; text-align: center; margin-left: 8px;

font-size: 12px; width: 16px; height: 16px;

background-color: #c62828; color: #ffffff;

border-radius: 50%;

animation: notification 1s ease-in-out infinite alternate;

}}

@keyframes active-status {{ from {{

opacity: 0.4;

}}

to {{

opacity: 1;

}}

}}

@keyframes notification {{ from {{

transform: scale(1) translateY(0);

}}

to {{

transform: scale(1.2) translateY(-2px);

}}

}}

</style>"""

st.markdown(css, unsafe\_allow\_html=True) st.markdown(text, unsafe\_allow\_html=True)

def main():

if not st.session\_state.get('role', False): onboardingLogin.display()

if st.session\_state.role == 'user':

if not st.session\_state.get('user\_logged\_in', False):

page = option\_menu(menu\_title=None, options=["Login", "Sign Up"], icons=["person", "person-plus"],

orientation="horizontal") if page == "Login":

login.display()

elif page == "Sign Up": signup.display()

else:

with st.sidebar:

st.markdown("<h1 style='text-align: center;'>User Dashboard</h1>", unsafe\_allow\_html=True)

selected = option\_menu("Main Menu", ["Home", "Book Ticket", "View Tickets", "Contact Us", "About Us"],

icons=['house', 'ticket', 'list', 'envelope', 'code-square'], menu\_icon="train-front", default\_index=0)

status\_shower(st.session\_state.email, st.session\_state.username,"Logged In" if st.session\_state.logged\_in else "Logged Out")

col1,col2,col3 = st.columns([1.5,2,1]) with col2:

if st.button("Logout", key="logout"): st.session\_state.role = None

st.session\_state.user\_logged\_in = None st.session\_state.email = None st.session\_state.username = None st.session\_state.logged\_in = None st.rerun()

if selected == "Home": userHome.display()

elif selected == "Book Ticket": userBookTicket.display()

elif selected == "View Tickets": userViewTickets.display()

elif selected == "Contact Us": userContactUs.display()

elif selected == "About Us": userAboutUs.display()

elif st.session\_state.role == 'admin':

if not st.session\_state.get('user\_logged\_in', False): login.display()

else:

with st.sidebar:

st.markdown("<h1 style='text-align: center;'>Admin Dashboard</h1>", unsafe\_allow\_html=True)

selected = option\_menu("Admin Menu", ["Home", "Manage Trains", "Manage Stations", "Settings","Guide"],

icons=['house', 'train-front', 'pin-map', 'gear','book'], menu\_icon="tools", default\_index=0)

status\_shower(st.session\_state.email, "Admin",

"Logged In" if st.session\_state.user\_logged\_in else "Logged

Out")

col1,col2,col3 = st.columns([1.5,2,1]) with col2:

if st.button("Logout", key="logout"): st.session\_state.role = None st.session\_state.user\_logged\_in = None st.session\_state.email = None st.session\_state.username = None st.session\_state.logged\_in = None st.rerun()

if selected == "Home": adminHome.display()

if selected == "Settings": adminSettings.display()

if selected == "Guide": adminGuide.display()

if selected == "Manage Trains": adminManageTrains.display()

if selected == "Manage Stations": adminManageStations.display()

if name == " main ":

main()

import mysql.connector import streamlit

from mysql.connector import Error import os

from dotenv import load\_dotenv load\_dotenv()

def create\_connection(): connection = None try:

connection = mysql.connector.connect( host=os.environ.get('DB\_HOST'), user=os.environ.get('DB\_USER'),

password=os.environ.get('DB\_PASSWORD'), database=os.environ.get('DB\_DATABASE')

)

except Error as e:

print(f"The error '{e}' occurred") return connection

def authenticate\_user(table, usermail, password): connection = create\_connection()

cursor = connection.cursor() try:

query = f"SELECT \* FROM {table} WHERE email = %s AND password = %s" cursor.execute(query, (usermail, password))

user = cursor.fetchone() return user

except Error as e:

print(f"The error '{e}' occurred") finally:

cursor.close() connection.close()

def register\_user(username, usermail, password): connection = create\_connection()

cursor = connection.cursor() try:

query = "INSERT INTO users (name ,email , password) VALUES (%s ,%s, %s)"

cursor.execute(query, (username, usermail, password)) connection.commit()

return True except Error as e:

print(f"The error '{e}' occurred") return False

finally:

cursor.close() connection.close()

def register\_admin(username, usermail, password): connection = create\_connection()

cursor = connection.cursor() try:

query = "INSERT INTO admins (name ,email , password) VALUES (%s ,%s,

%s)"

cursor.execute(query, (username, usermail, password)) connection.commit()

return True except Error as e:

print(f"The error '{e}' occurred") return False

finally:

cursor.close() connection.close()

def execute\_query(query, params=()):

connection = create\_connection() cursor = connection.cursor()

try:

cursor.execute(query, params) connection.commit()

return cursor.lastrowid except Error as e:

print(f"The error '{e}' occurred")

def execute\_read\_query(query, params=()): connection = create\_connection()

cursor = connection.cursor(dictionary=True) try:

cursor.execute(query, params) result = cursor.fetchall() return result

except Error as e:

print(f"The error '{e}' occurred") finally:

cursor.close() connection.close()

def get\_stations():

query = "SELECT \* FROM stations" return execute\_read\_query(query)

def add\_station(station\_name):

query = "INSERT INTO stations (name) VALUES (%s)" return execute\_query(query, (station\_name,))

def update\_station(old\_name, new\_name):

query = "UPDATE stations SET name = %s WHERE name = %s" return execute\_query(query, (new\_name, old\_name))

def delete\_station(station\_name):

query = "DELETE FROM stations WHERE name = %s" return execute\_query(query, (station\_name,))

def get\_trains\_by\_station(station\_name): query = """

SELECT t.\*

FROM trains t

JOIN stations s ON t.from\_station = s.station\_id OR t.to\_station = s.station\_id WHERE s.name = %s

"""

return execute\_read\_query(query, (station\_name,))

def update\_train\_station(train\_id, new\_station\_name):

new\_station\_id\_query = "SELECT station\_id FROM stations WHERE name =

%s"

new\_station\_id = execute\_read\_query(new\_station\_id\_query, (new\_station\_name,))[0]['station\_id']

query = "UPDATE trains SET from\_station = %s WHERE train\_id = %s" return execute\_query(query, (new\_station\_id, train\_id))

def get\_users():

query = "SELECT \* FROM users" users = execute\_read\_query(query) return users

def get\_trains():

query = "SELECT \* FROM trains" trains = execute\_read\_query(query) return trains

def search\_trains(from\_station, to\_station, date\_of\_travel): query = """

SELECT \* FROM trains

WHERE from\_station IN (SELECT station\_id FROM stations WHERE name =

%s)

AND to\_station IN (SELECT station\_id FROM stations WHERE name = %s) """

params = (from\_station, to\_station)

trains = execute\_read\_query(query, params) return trains

def get\_user\_by\_email(email):

query = "SELECT \* FROM users WHERE email = %s" user = execute\_read\_query(query, (email,))

return user[0] if user else None

def get\_username\_by\_email(table, email):

query = f"SELECT \* FROM {table} WHERE email = %s" user = execute\_read\_query(query, (email,))

return user[0]['name'] if user else None

def create\_user(name, email):

query = "INSERT INTO users (name, email) VALUES (%s, %s)" user\_id = execute\_query(query, (name, email))

return user\_id

def book\_ticket(train\_id, email, date\_of\_travel):

query = "INSERT INTO bookings (train\_id, email, date\_of\_travel) VALUES (%s,

%s, %s)"

params = (train\_id, email, date\_of\_travel) booking\_id = execute\_query(query, params) return booking\_id

def add\_passenger(booking\_id, name, age, sex):

query = "INSERT INTO passengers (booking\_id, name, age, sex) VALUES (%s,

%s, %s, %s)"

return execute\_query(query, (booking\_id, name, age, sex))

def get\_tickets(email): query = """

SELECT b.booking\_id, b.train\_id, b.date\_of\_travel, u.name FROM bookings b

JOIN users u ON b.email = u.email WHERE u.email = %s

"""

params = (email,)

tickets = execute\_read\_query(query, params) return tickets

def delete\_ticket(booking\_id):

# First, delete all passengers associated with the booking

delete\_passengers\_by\_booking\_id(booking\_id)

# Then, delete the booking from the 'bookings' table query = "DELETE FROM bookings WHERE booking\_id = %s" execute\_query(query, (booking\_id,))

def delete\_passengers\_by\_booking\_id(booking\_id):

query = "DELETE FROM passengers WHERE booking\_id = %s" execute\_query(query, (booking\_id,))

def get\_passengers\_by\_booking\_id(booking\_id):

query = "SELECT \* FROM passengers WHERE booking\_id = %s" return execute\_read\_query(query, (booking\_id,))

def delete\_passenger(passenger\_id):

query = "DELETE FROM passengers WHERE passenger\_id = %s" execute\_query(query, (passenger\_id,))

def get\_trains(): query = """

SELECT t.train\_id, t.name, s1.name AS from\_station, s2.name AS to\_station, t.departure\_time, t.arrival\_time, t.total\_seats, t.available\_seats

FROM trains t

JOIN stations s1 ON t.from\_station = s1.station\_id JOIN stations s2 ON t.to\_station = s2.station\_id """

return execute\_read\_query(query)

def add\_train(name, from\_station, to\_station, departure\_time, arrival\_time, total\_seats, available\_seats):

query = """

INSERT INTO trains (name, from\_station, to\_station, departure\_time, arrival\_time, total\_seats, available\_seats)

VALUES (%s, %s, %s, %s, %s, %s, %s) """

return execute\_query(query, (name, from\_station, to\_station, departure\_time, arrival\_time, total\_seats, available\_seats))

def update\_train(train\_id, name, from\_station, to\_station, departure\_time, arrival\_time, total\_seats, available\_seats):

query = """ UPDATE trains

SET name = %s, from\_station = %s, to\_station = %s, departure\_time = %s, arrival\_time = %s, total\_seats = %s, available\_seats = %s

WHERE train\_id = %s """

return execute\_query(query, (name, from\_station, to\_station, departure\_time, arrival\_time, total\_seats, available\_seats, train\_id))

def delete\_train(train\_name):

query = "DELETE FROM trains WHERE name = %s" return execute\_query(query, (train\_name,))

def delete\_trains\_by\_station(station\_name): query = """

DELETE FROM trains

WHERE from\_station = (SELECT station\_if from stations where name=%s) OR to\_station = (SELECT station\_id from stations where name=%s)

"""

execute\_query(query, (station\_name, station\_name))

//A consolidated SQL commands to create all the table of the project

//TOOL - MySQL Workbench

CREATE DATABASE train\_reservation\_db; USE train\_reservation\_db;

CREATE TABLE `admins` (

`email` varchar(255) NOT NULL,

`name` varchar(100) NOT NULL,

`password` varchar(255) NOT NULL, PRIMARY KEY (`email`),

UNIQUE KEY `username` (`name`)

)

CREATE TABLE `passengers` (

`passenger\_id` int NOT NULL AUTO\_INCREMENT,

`booking\_id` int DEFAULT NULL,

`name` varchar(100) DEFAULT NULL,

`age` int DEFAULT NULL,

`sex` varchar(10) DEFAULT NULL, PRIMARY KEY (`passenger\_id`), KEY `booking\_id` (`booking\_id`),

CONSTRAINT `passengers\_ibfk\_1` FOREIGN KEY (`booking\_id`) REFERENCES

`bookings` (`booking\_id`)

)

CREATE TABLE `stations` (

`station\_id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(100) NOT NULL, PRIMARY KEY (`station\_id`)

)

CREATE TABLE `bookings` (

`booking\_id` int NOT NULL AUTO\_INCREMENT,

`train\_id` int DEFAULT NULL,

`date\_of\_travel` date NOT NULL,

`email` varchar(100) NOT NULL, PRIMARY KEY (`booking\_id`), KEY `train\_id` (`train\_id`),

CONSTRAINT `bookings\_ibfk\_2` FOREIGN KEY (`train\_id`) REFERENCES `trains` (`train\_id`)

)

CREATE TABLE `users` (

`name` varchar(100) NOT NULL,

`email` varchar(100) NOT NULL,

`password` varchar(100) NOT NULL, PRIMARY KEY (`email`),

UNIQUE KEY `email` (`email`)

)

CREATE TABLE `trains` (

`train\_id` int NOT NULL AUTO\_INCREMENT,

`name` varchar(100) NOT NULL,

`from\_station` int DEFAULT NULL,

`to\_station` int DEFAULT NULL,

`departure\_time` time NOT NULL,

`arrival\_time` time NOT NULL,

`total\_seats` int NOT NULL,

`available\_seats` int NOT NULL, PRIMARY KEY (`train\_id`),

KEY `from\_station` (`from\_station`), KEY `to\_station` (`to\_station`),

CONSTRAINT `trains\_ibfk\_1` FOREIGN KEY (`from\_station`) REFERENCES

`stations` (`station\_id`),

CONSTRAINT `trains\_ibfk\_2` FOREIGN KEY (`to\_station`) REFERENCES

`stations` (`station\_id`)

)

import streamlit as st

from pages.database import database

def display():

st.title(f"Login as {st.session\_state.role}") if st.session\_state.role == "user":

table = "users" else:

table = "admins"

usermail = st.text\_input("Email")

password = st.text\_input("Password", type="password") if st.button("Login"):

user = database.authenticate\_user(table, usermail, password) if user:

st.success("Logged in successfully!") st.session\_state.email = usermail st.session\_state.logged\_in = True

st.session\_state.username = database.get\_username\_by\_email(table,

usermail)

st.session\_state.user\_logged\_in = True st.rerun()

else:

st.error("Invalid username or password") import streamlit as st

# Declare the main function for onboarding as user and admin def display():

if 'role' not in st.session\_state: st.session\_state['role'] = None

image\_path = "assets\\"

# Show the logo and ask what kind of login

st.header("iRTS - Railway Reservation System", divider='red') col1, col2 = st.columns([1, 1])

with col1:

container = st.container() with container:

cola, colb, colc = st.columns([1, 2, 1]) with colb:

st.markdown("<h3 style='text-align: center;'>User</h3>", unsafe\_allow\_html=True)

st.image(f"{image\_path}Mobile inbox-pana.png", use\_column\_width=True)

if st.button("Enter", key='user'): st.session\_state['role'] = 'user' st.rerun()

with col2:

container = st.container() with container:

cola, colb, colc = st.columns([1, 2, 1]) with colb:

st.markdown("<h3 style='text-align: center;'>Admin</h3>", unsafe\_allow\_html=True)

st.image(f"{image\_path}Server-cuate.png", use\_column\_width=True) if st.button("Enter", key='admin'):

st.session\_state['role'] = 'admin' st.rerun()

import streamlit as st

from pages.database import database

def display():

st.title(f"Sign Up as {st.session\_state.role}") username = st.text\_input("Username") usermail = st.text\_input("Email")

password = st.text\_input("Password", type="password") confirm\_password = st.text\_input("Confirm Password", type="password") if st.button("Sign Up"):

if password == confirm\_password:

if database.register\_user(username, usermail, password): st.success("Sign up successful!")

st.session\_state.email = usermail st.session\_state.logged\_in = True st.session\_state.username = username st.session\_state.user\_logged\_in = True st.rerun()

else:

st.error("Failed to register user") else:

st.error("Passwords do not match") import streamlit as st

def display(): st.title("About Us") st.markdown("---")

st.header("Our Team")

col1, col2 = st.columns(2)

with col1:

with st.container(): st.subheader("Rakhul",divider='red')

if st.button("Know More About Rakhul"): with st.expander("Details"):

st.markdown("""

\*\*Student at Rajalakshmi Engineering College, II Year - CSE\*\*

Second-year Computer Science and Engineering student at Rajalakshmi Engineering College. Talented developer with a keen eye for detail and a passion for learning. Expertise in various technologies and the ability to quickly adapt to new challenges have been invaluable to the team's success.

Significant contributions have been made to both the front-end and back-end connections of the project.

""")

st.balloons()

with col2:

with st.container(): st.subheader("Sabarish",divider='red')

if st.button("Know More About Sabarish"): with st.expander("Details"):

st.markdown("""

\*\*Student at Rajalakshmi Engineering College, II Year - CSE\*\*

Second-year Computer Science and Engineering student at Rajalakshmi Engineering College. Skilled in backend development, database

normalization, and problem-solving. Strong analytical thinker with a passion for coding and creating efficient systems. Contributions have been pivotal in enhancing system performance and reliability, demonstrating dedication and diligence in every project undertaken.

""")

st.snow()

st.markdown("---") st.header("Project Stack") st.markdown("""

Our project leverages a combination of powerful technologies to deliver an efficient and user-friendly train reservation system. Here's a look at the technology stack we used:

""")

with st.expander("Streamlit"): st.markdown("""

* \*\*Streamlit\*\*: An open-source app framework used to create the front- end interface. Streamlit allows for quick and easy development of interactive web applications.

""")

with st.expander("MySQL"): st.markdown("""

* \*\*MySQL\*\*: A reliable and scalable relational database management system used for storing and managing data. MySQL's robust features enable us to handle complex queries and transactions efficiently.

""")

st.markdown("""

Our team has worked diligently to integrate these technologies seamlessly, ensuring a smooth and enjoyable user experience. We are proud of what we have achieved and look forward to continuing to improve and expand our system.

""")

import streamlit as st import pandas as pd

from pages.database import database

from pages.userPages import userBookingForm

def display():

st.title("Book Train Tickets")

st.caption(f"\*\*Logged in as:\*\* {st.session\_state.email}") # Search filter

st.subheader("Search Filter") col1, col2, col3 = st.columns(3) st.subheader("Search Filter") with col1:

from\_station = st.selectbox("From", options=userBookingForm.get\_station\_names(), index=0)

with col2:

to\_station = st.selectbox("To", options=userBookingForm.get\_station\_names(), index=0)

with col3:

date\_of\_travel = st.date\_input("Date of Travel") # Fetch trains based on search filter

trains = database.search\_trains(from\_station, to\_station, date\_of\_travel)

# Display filtered trains in a table if trains:

st.subheader("Available Trains") df\_trains = pd.DataFrame(trains) st.dataframe(df\_trains)

# Book button for each train

for index, row in df\_trains.iterrows(): userBookingForm.display\_booking\_form(row,date\_of\_travel)

else:

st.error("No trains available for the selected route and date.") import streamlit as st

from pages.database import database

def display\_booking\_form(train,date\_of\_travel): st.title("Enter Passenger Details")

st.write("You selected the following train:") st.subheader(

f"Train ID: {train['train\_id']}, Train Name: {train['name']}, Departure:

{train['departure\_time']}, Arrival: {train['arrival\_time']}") st.header("Passenger Details")

# Initialize with 6 passenger fields passenger\_fields = [

{'name': '', 'age': '', 'sex': ''}

for \_ in range(6)

]

for i, passenger in enumerate(passenger\_fields): st.markdown(f"\*\*Passenger {i + 1}\*\*")

col1, col2, col3 = st.columns(3) # Create 3 columns for layout with col1:

name = st.text\_input("Name", key=f"name\_{i}", value=passenger['name'])

with col2:

age = st.text\_input("Age", key=f"age\_{i}", value=passenger['age']) with col3:

sex = st.selectbox("Sex", ["Male", "Female", "Other"], key=f"sex\_{i}",

index=0 if passenger['sex'] == 'Male' else 1 if passenger['sex']

== 'Female' else 2)

# Update passenger\_fields list

passenger\_fields[i] = {'name': name, 'age': age, 'sex': sex}

# Filter out empty fields

passenger\_fields = [passenger for passenger in passenger\_fields if all(passenger.values())]

if st.button("Submit"): if passenger\_fields:

# Book tickets using the database function

booking\_id = book\_tickets(train['train\_id'], st.session\_state['email'], date\_of\_travel, passenger\_fields)

if booking\_id:

st.success(f"Tickets booked successfully! Your booking ID is

{booking\_id}.") else:

st.error("Please fill in at least one passenger's details.")

def get\_station\_names():

stations = database.get\_stations()

return [station['name'] for station in stations]

def book\_tickets(train\_id, user\_mail, date\_of\_travel, passengers):

# Assuming this function is implemented in the database module to book tickets

booking\_id = database.book\_ticket(train\_id, user\_mail, date\_of\_travel) for passenger in passengers:

database.add\_passenger(booking\_id, passenger['name'], passenger['age'], passenger['sex'])

return booking\_id import streamlit as st

def display():

# Title and Subtitle st.title("📞 Contact Us")

st.subheader("We're here to help!") # Add an image/banner col1,col2,col3 = st.columns([1,2,1]) with col2:

st.image('assets//train-pana.png', width=450)

# Contact Information st.markdown("""

<style>

.contact-info { font-size: 20px;

margin-bottom: 25px;

}

.contact-icon { font-size: 22px;

margin-right: 10px;

}

</style>

<div class="contact-info">

<p><span class="contact-icon">📧</span>Email: <a hre[f="ma](mailto:support@trainbookingsystem.com)ilt[o:support@trainbookingsystem.com">](mailto:support@trainbookingsystem.com)support@trainbookingsyste m.com</a></p>

<p><span class="contact-icon">📞</span>Phone: +1 234 567 890</p>

<p><span class="contact-icon">🏢</span>Address: 123 Train St, Booking City, Country</p>

</div>

""", unsafe\_allow\_html=True)

# Contact Form

st.markdown("### Send Us a Message") with st.form("contact\_form"):

name = st.text\_input("Your Name") email = st.text\_input("Your Email")

subject = st.text\_input("Subject") message = st.text\_area("Message")

submitted = st.form\_submit\_button("Send")

if submitted:

st.success("Thank you for reaching out to us! We'll get back to you soon.")

# Social Media Links st.markdown("""

<style>

.social-media { font-size: 25px;

}

.social-media a { margin: 0 15px;

}

</style>

<div class="social-media">

<a href="https://facebook.com" target="\_blank">🌐 Facebook</a>

<a href="https://twitter.com" target="\_blank">🌐 Twitter</a>

<a href="https://linkedin.com" target="\_blank">🌐 LinkedIn</a>

<a href="https://instagram.com" target="\_blank">🌐 Instagram</a>

</div>

""", unsafe\_allow\_html=True) # Add a decorative line

st.markdown("<hr style='border: 1px solid #f0f0f0;'>", unsafe\_allow\_html=True)

# Call the function to display the page import streamlit as st

from pages.database import database import os

import pandas as pd

def display():

# Title with icon

st.title("🚄 iTRS - Train Reservation System")

# Centered image with columns

col1, col2, col3 = st.columns([0.4, 0.45, 0.4]) with col2:

image\_path = "assets\\"

st.image(f"{image\_path}Train-bro.png", use\_column\_width=True)

# Fetch and display trains in a table trains = database.get\_trains()

if trains:

st.subheader("🚂 Available Trains") df\_trains = pd.DataFrame(trains)

st.dataframe(df\_trains.style.set\_properties(\*\*{'text-align': 'center'})) else:

st.info("No trains available at the moment.")

# Book Tickets Section st.subheader("🎟️ Book Tickets") st.markdown(

"Book tickets for your journey by navigating to the \*\*Book Tickets\*\* page using the main menu in the sidebar.")

st.markdown("---")

# About Us Section st.header("📖 About Us") st.write("""

Welcome to the Train Ticket Booking System. Our aim is to provide a seamless and user-friendly experience for booking train tickets.

With our platform, you can easily search for trains, view available tickets, and book your journey with just a few clicks.

""")

st.markdown("---")

# Contact Us Section st.header("📞 Contact Us") st.write(

"For inquiries, please email us at [[sup](mailto:support@trainbookingsystem.com)p[ort@trainbookingsystem.com](](mailto:support@trainbookingsystem.com)mailto:support@trainbookingsystem.com)

.")

# Footer

st.markdown("<hr>", unsafe\_allow\_html=True)

st.markdown("<p style='text-align: center;'>© 2024 iTRS - Train Reservation System. All rights reserved.</p>",

unsafe\_allow\_html=True)

if name == " main ": display()

import streamlit as st

from pages.database import database

def display():

st.title("🎟️ View and Manage Your Tickets") st.markdown("## Welcome to your ticket dashboard!") st.caption(f"\*\*Logged in as:\*\* {st.session\_state.email}")

search\_query = st.text\_input("🔍 Search Tickets", help="Search by Booking ID or Train ID",

placeholder="Enter Booking ID or Train ID") tickets = database.get\_tickets(st.session\_state.email)

if search\_query:

tickets = [ticket for ticket in tickets if

search\_query.lower() in str(ticket['booking\_id']).lower() or search\_query.lower() in str(

ticket['train\_id']).lower()]

if tickets:

st.markdown(f"### Found {len(tickets)} ticket(s):") for ticket in tickets:

with st.expander(f"📄 \*\*Booking ID:\*\* {ticket['booking\_id']}", expanded=True):

st.write(f"\*\*Train ID:\*\* {ticket['train\_id']}") st.write(f"\*\*Date of Travel:\*\* {ticket['date\_of\_travel']}") st.write(f"\*\*Name:\*\* {ticket['name']}") st.write(f"\*\*Email:\*\* {st.session\_state.email}")

passengers = database.get\_passengers\_by\_booking\_id(ticket['booking\_id'])

if passengers:

st.markdown("#### Passengers:") for passenger in passengers:

col1, col2, col3, col4 = st.columns([2, 1, 1, 1]) col1.write(f"\*\*Name:\*\* {passenger['name']}") col2.write(f"\*\*Age:\*\* {passenger['age']}") col3.write(f"\*\*Sex:\*\* {passenger['sex']}") with col4:

if st.button("❌ Delete Passenger", key=f"delete\_passenger\_{passenger['passenger\_id']}"):

database.delete\_passenger(passenger['passenger\_id']) st.success(f"Passenger {passenger['name']} deleted.") st.rerun() # Refresh the page to show updated passengers

if st.button("🗑️ Delete Booking", key=f"delete\_booking\_{ticket['booking\_id']}"):

database.delete\_ticket(ticket['booking\_id'])

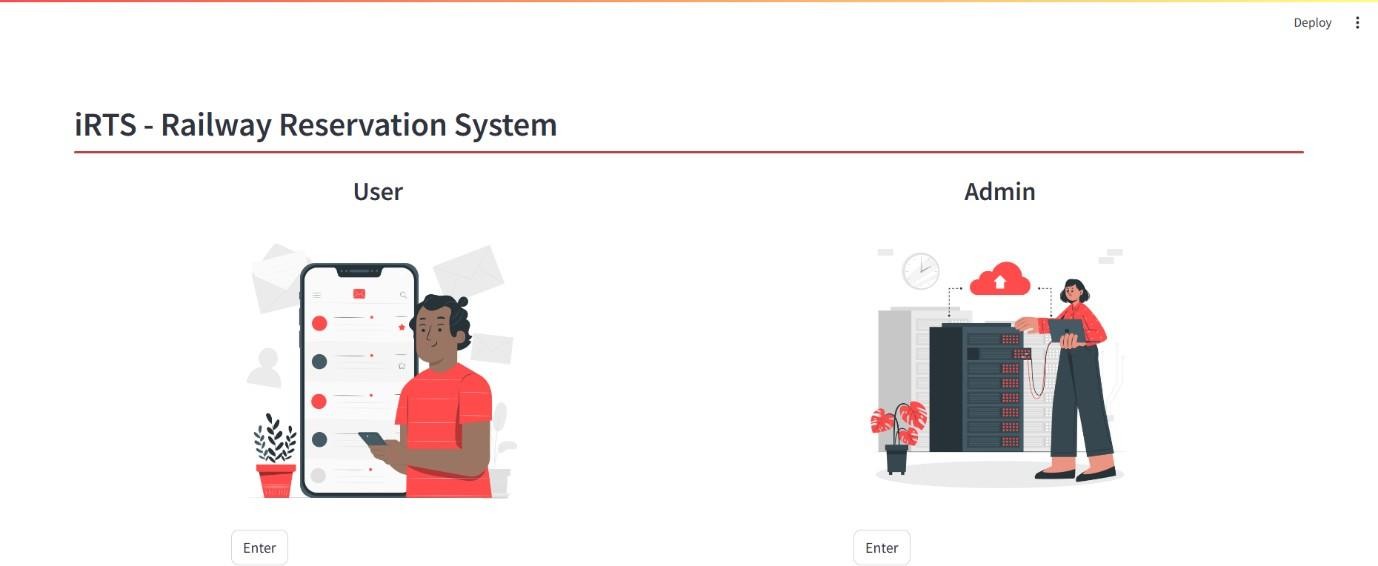
st.success(f"Booking ID {ticket['booking\_id']} deleted.") st.rerun() # Refresh the page to show updated tickets

else:

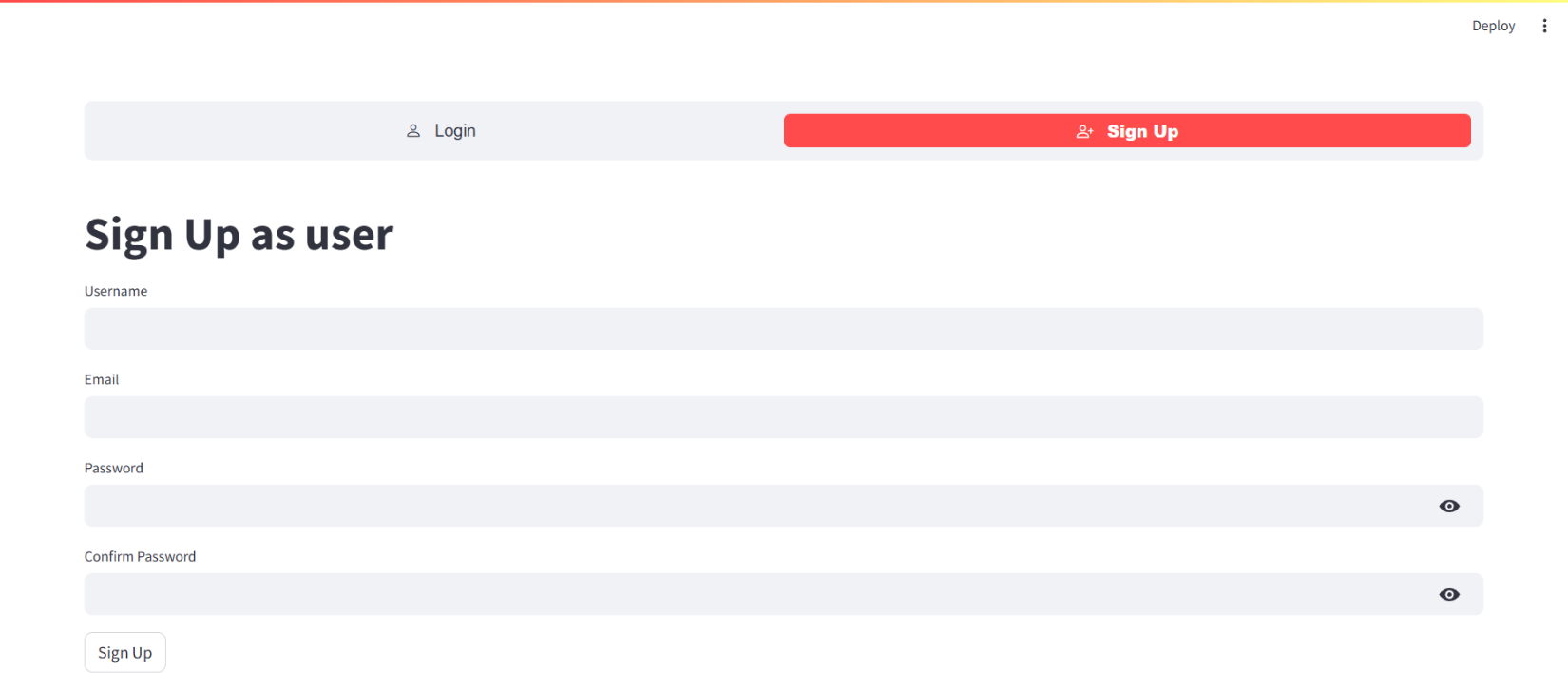
st.warning("No tickets found. Please check your search query or book a new ticket.")

# RESULT AND DISCUSSION

## USER DOCUMENTATION : On boarding page :

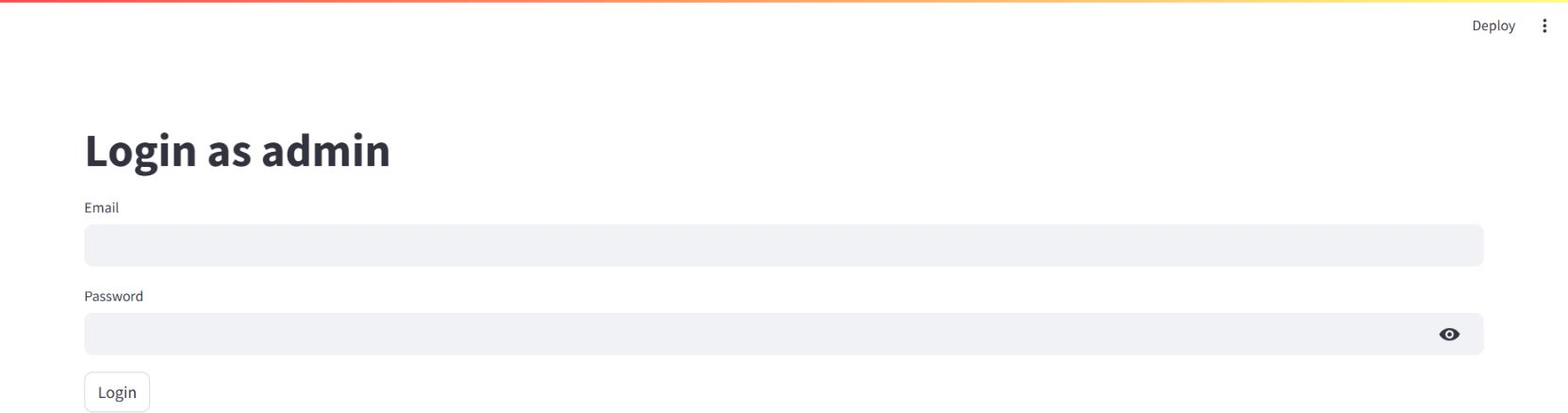


**Sign up page :**



## User login :

**Admin login :**



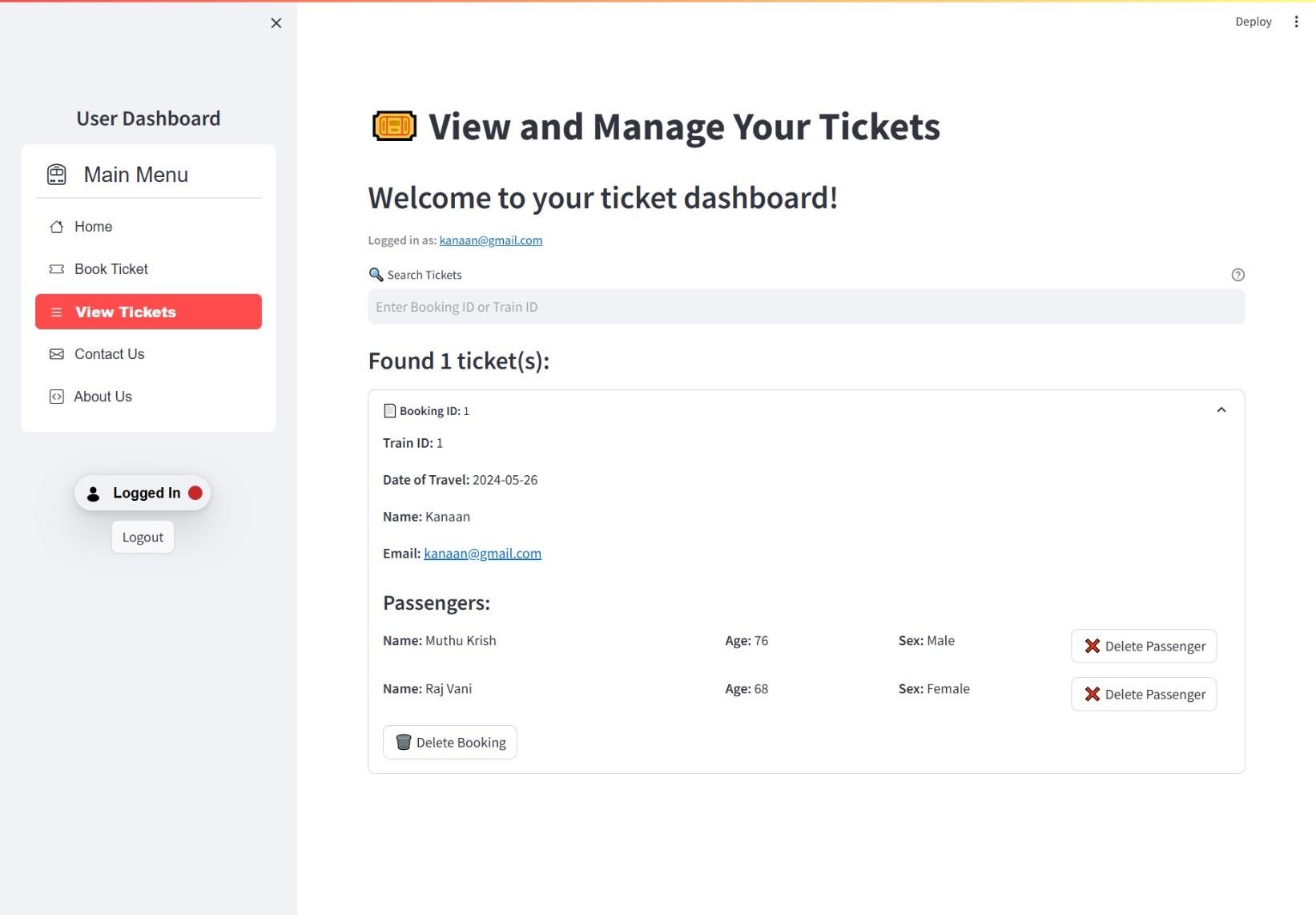
## Home page :

64

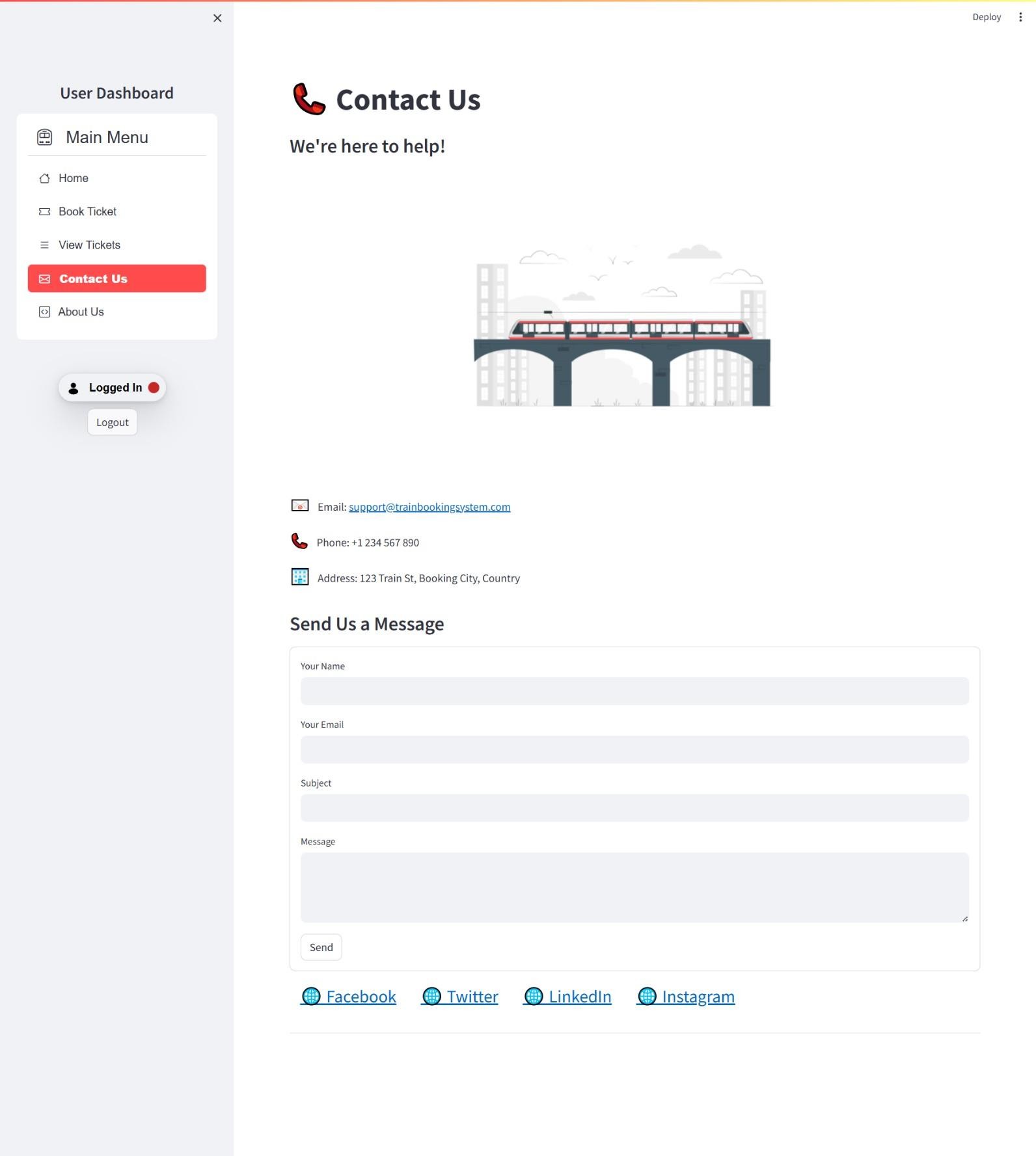
## Book ticket page :

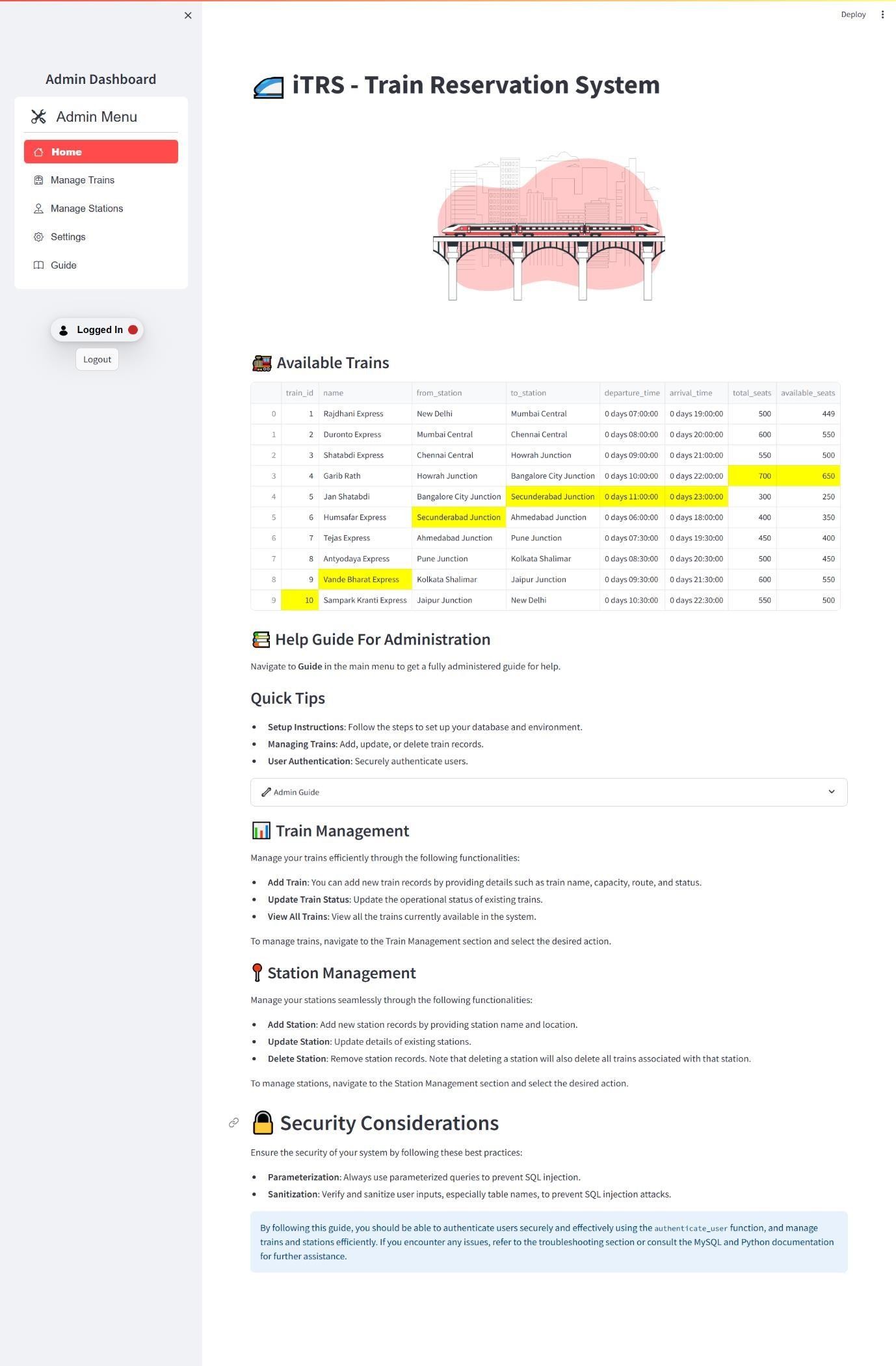
65

View ticket page :



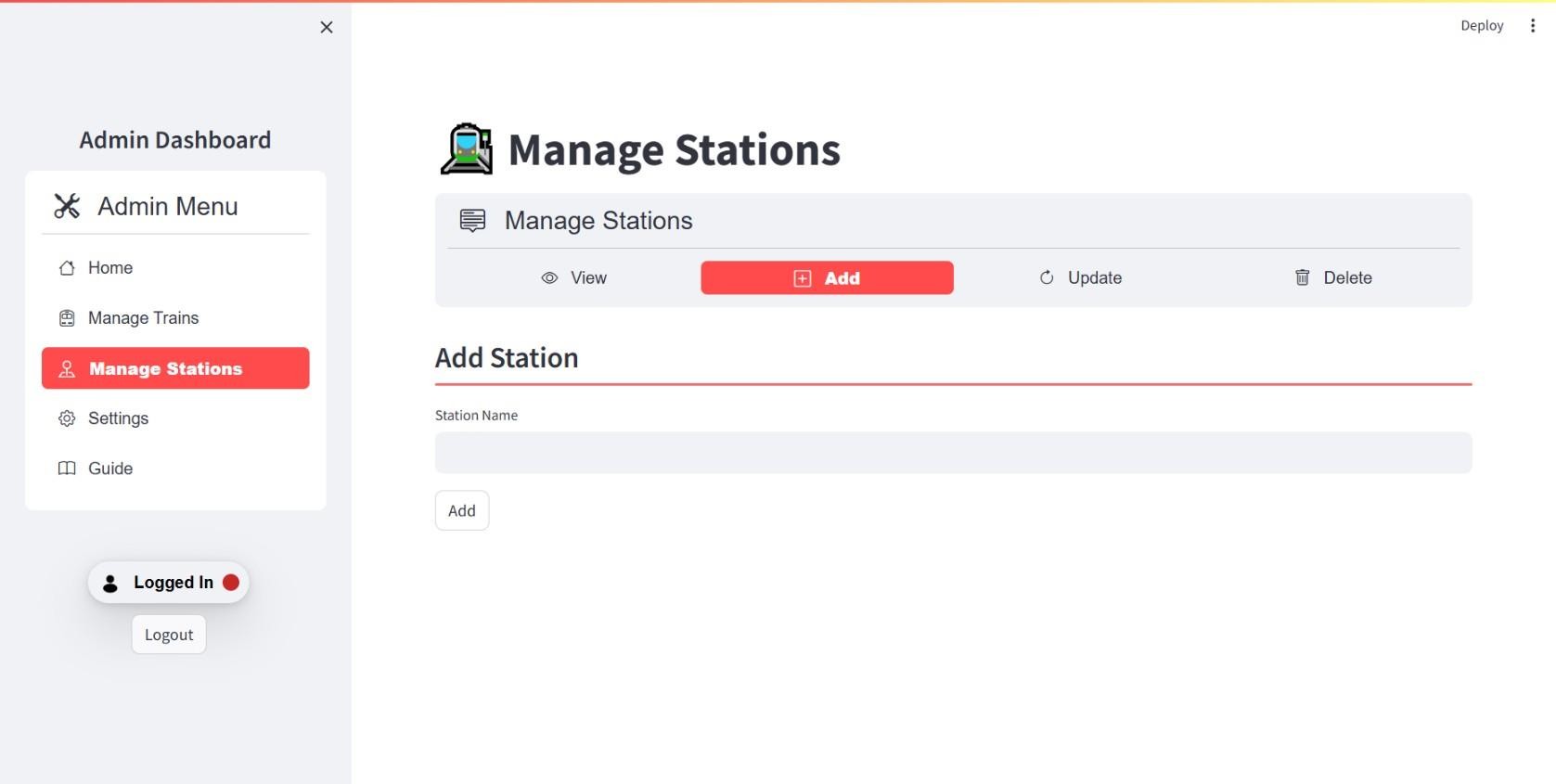
## Contact us page :

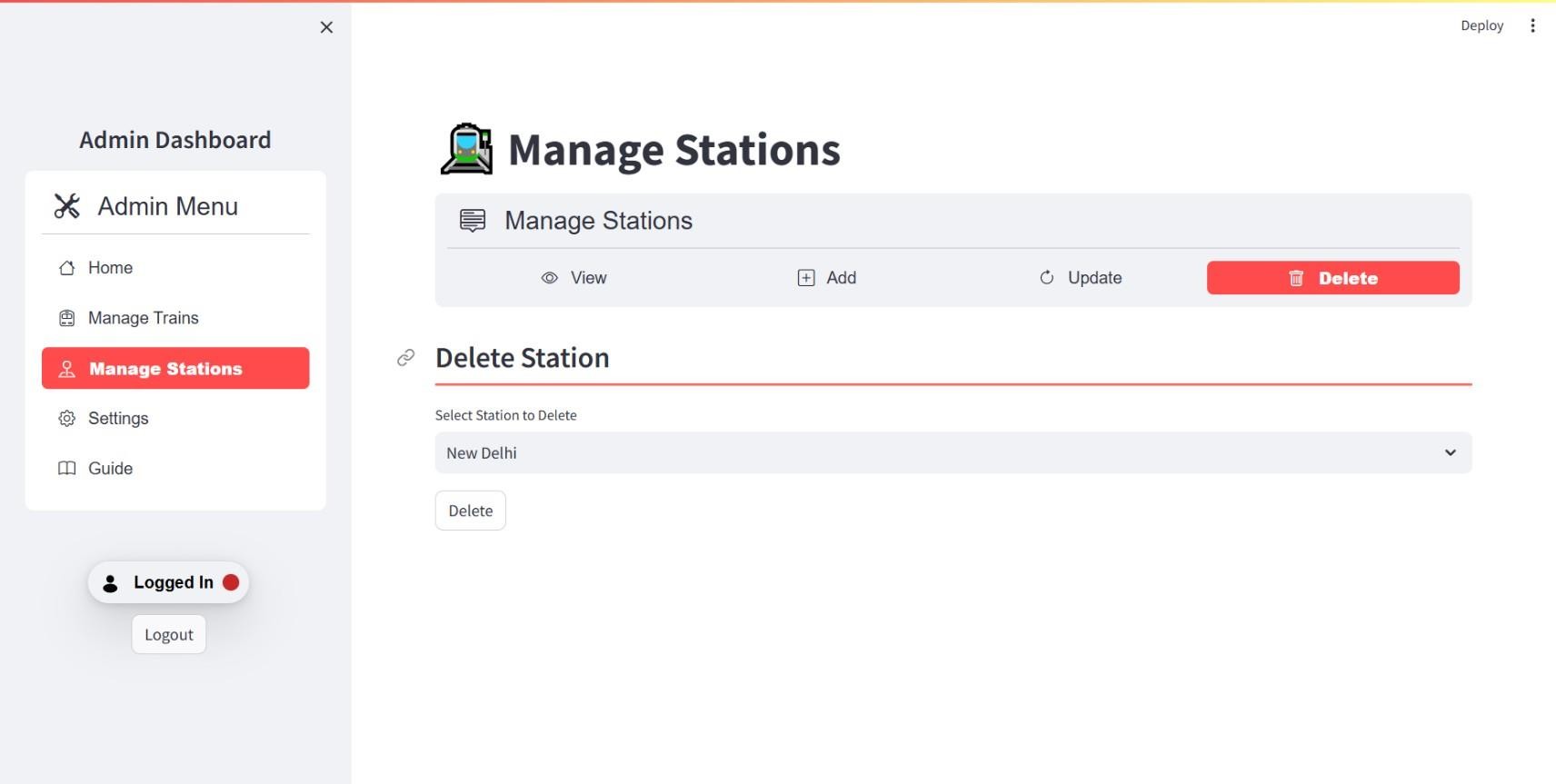


**Admin home page :**

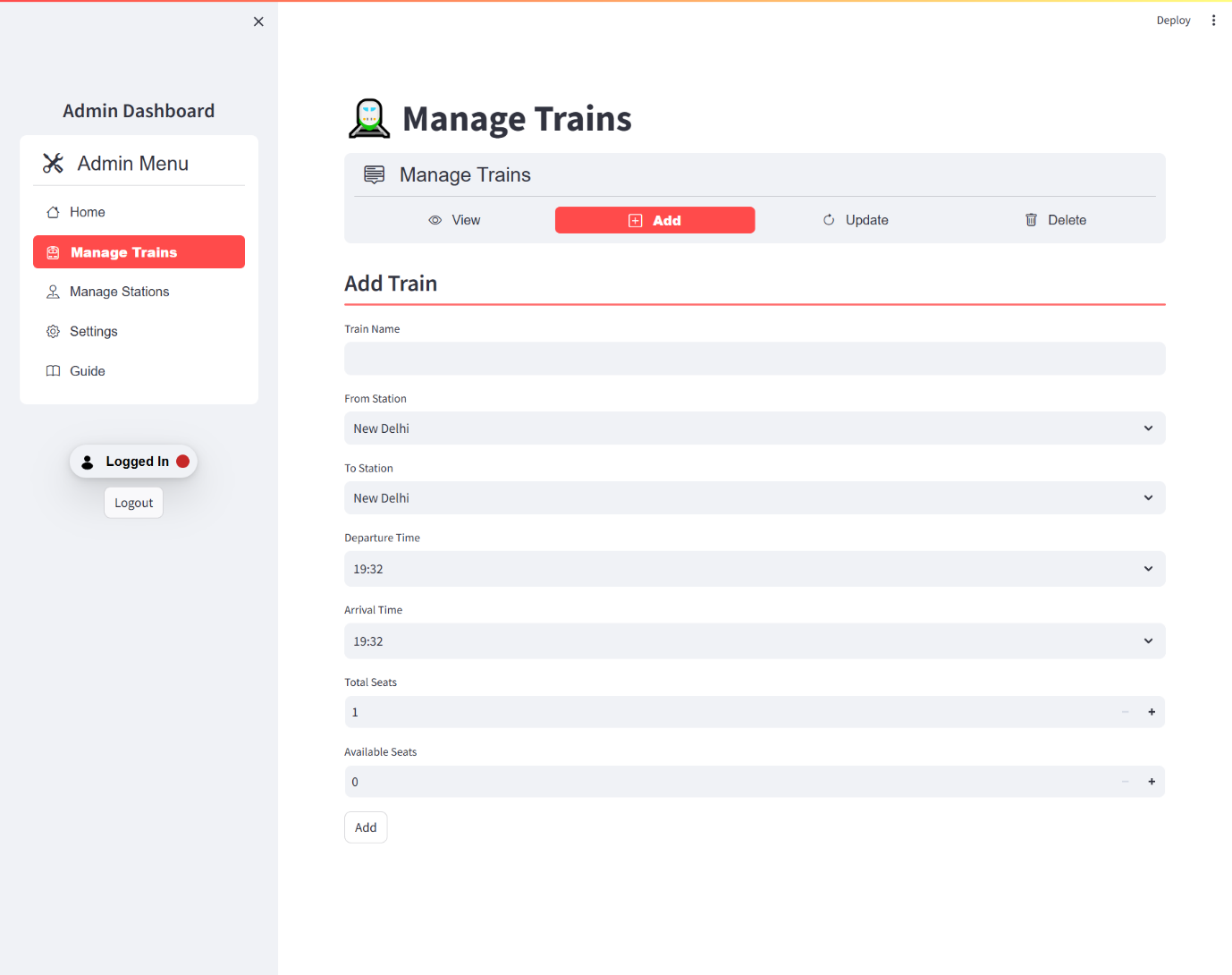
68

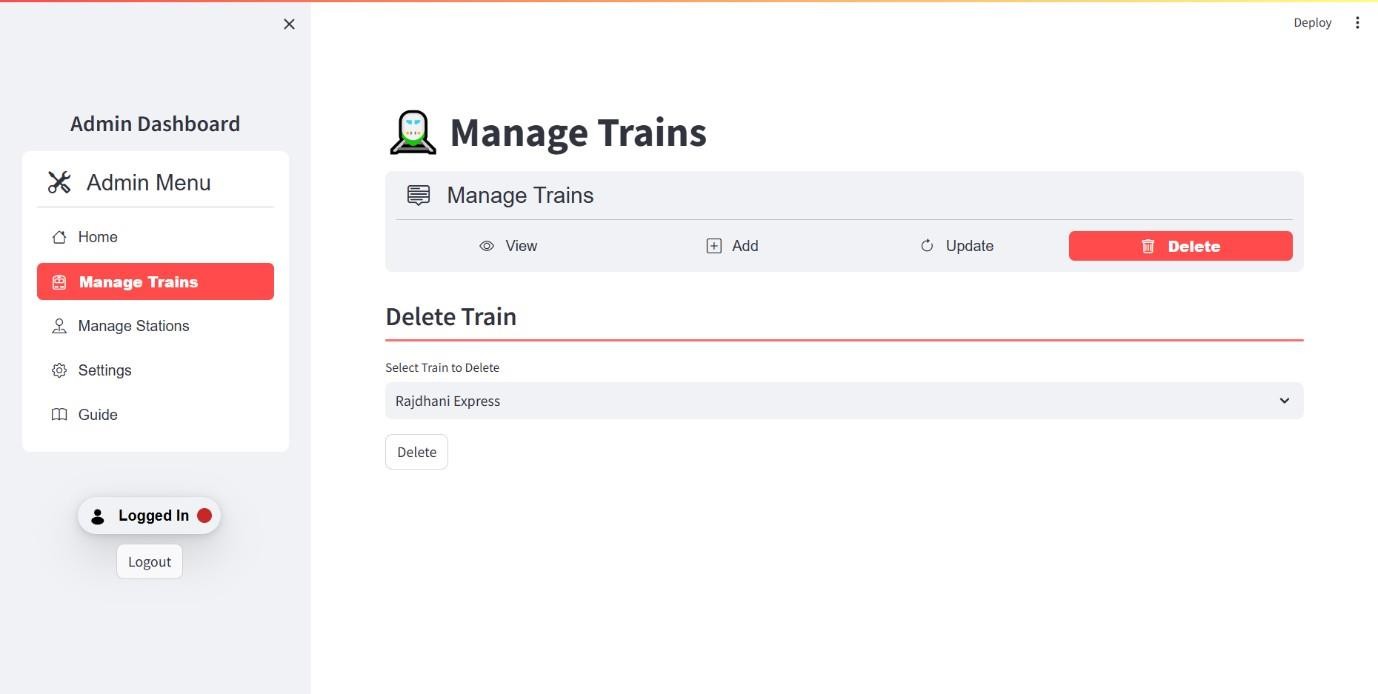
## Manage station page :





**Manage train page :**





# 6. TESTING

## Unit Testing

Unittesting is a testing technique in which modules are tested individually.

Small

individual units of source code are tested to determine whether it is fit to use or not.

Different modules of games are put to test while the modules are being developed. Here

modules refer to individual levels, players, scenes

## Integration Testing

Integration testing is the technique in which individual components or modules are

grouped together and tested. It occurs after testing. The inputs for the integrated testing

are the modules that have already been unit tested.

## System Testing

System testing is conducted on the entire system as a whole to check whether the system

meets its requirements or not. software was installed on different systems and any errors

or bugs that occurred were fixed.

## Acceptance Testing

User Acceptance is defined as a type of testing performed by the Client to certify the

system with respect to the requirements that was agreed upon. Thistesting happens in the

final phase of testing before moving the software application to the Market or Production

environment.

# 7.CONCLUSION

The Railway Ticket Booking System project successfully implements a comprehensive and user-friendly platform for booking, viewing, and canceling railway tickets. Utilizing Python, Streamlit, and MySQL, the system ensures efficient data management, secure transactions, and a smooth user experience. By incorporating database normalization techniques, the project achieves optimal data storage and integrity, minimizing redundancy and enhancing performance.

Key features such as secure user authentication, detailed train schedules, real- time seat availability, and integrated payment processing provide a seamless ticket booking experience. The administrative panel allows for effective management of train schedules, fares, and user accounts, ensuring that the system remains up-to-date and efficient.

Overall, the project demonstrates a robust and scalable solution for railway ticket booking, addressing the needs of both users and administrators. The implementation of best practices in database design and web application development ensures that the system is reliable, secure, and easy to maintain, making it a valuable tool for modern railway operations.

# 8.REFERENCES

### Python and Streamlit Documentation:

* Python: <https://docs.python.org/3/>
* Streamlit: https://docs.streamlit.io/

### MySQL Documentation:

* MySQL: <https://dev.mysql.com/doc/>

### Database Design and Normalization:

* Database Normalization Basics:

https://[www.essentialsql.com/get-ready-to-learn-sql-database-](http://www.essentialsql.com/get-ready-to-learn-sql-database-) normalization-explained-in-simple-english/

* Database Design Documentation: https://[www.guru99.com/database-design.html](http://www.guru99.com/database-design.html)

These references provide a solid foundation for understanding the technologies and best practices used in the iRTS-Railway Ticket Booking System project. They cover the essential aspects of programming, database design, security, payment integration, and user interface design that are critical to the successful implementation and maintenance of the system.