

# **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

<b>S.No</b>	<b>CONTENTS</b>	<b>Pg No.</b>
<b>1</b>	<b>INTRODUCTION</b>	<b>5</b>
1.1	INTRODUCTION	5
1.2	OBJECTIVES	5
1.3	MODULES	5
<b>2</b>	<b>SURVEY OF TECHNOLOGIES</b>	<b>7</b>
2.1	SOFTWARE DESCRIPTION	7
2.2	LANGUAGES	7
2.2.1	PYTHON	7
2.2.2	SQL	8
<b>3</b>	<b>REQUIREMENTS AND ANALYSIS</b>	<b>9</b>
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
<b>4</b>	<b>PROGRAM CODE</b>	<b>19</b>
<b>5</b>	<b>RESULTS AND DISCUSSION</b>	<b>62</b>
<b>6</b>	<b>TESTING</b>	<b>71</b>
<b>7</b>	<b>CONCLUSION</b>	<b>72</b>
<b>8</b>	<b>REFERENCES</b>	<b>73</b>

# 1.INTRODUCTION

## 1.1 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.

## 1.2 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

## 1.3 MODULES :

1. **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.

- 

## 2. 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.

## 3. 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.

## 4. Cancel Ticket:

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

## 5. 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.

## 6. 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**

### **2.1 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.

### **2.2 LANGUAGES :**

#### **2.2.1 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.

### **2.2.2 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

### 3.1 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.

#### **2. 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.

#### **3. 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.

#### **4. 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.

#### **5. 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.

#### **2. 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.

#### **3. Scalability:**

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

#### **4. 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.

#### **5. 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.

#### **6. 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.

### **3.2 HARDWARE AND SOFTWARE REQUIREMENTS :**

#### **Software Requirements :**

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

#### **Hardware Requirements :**

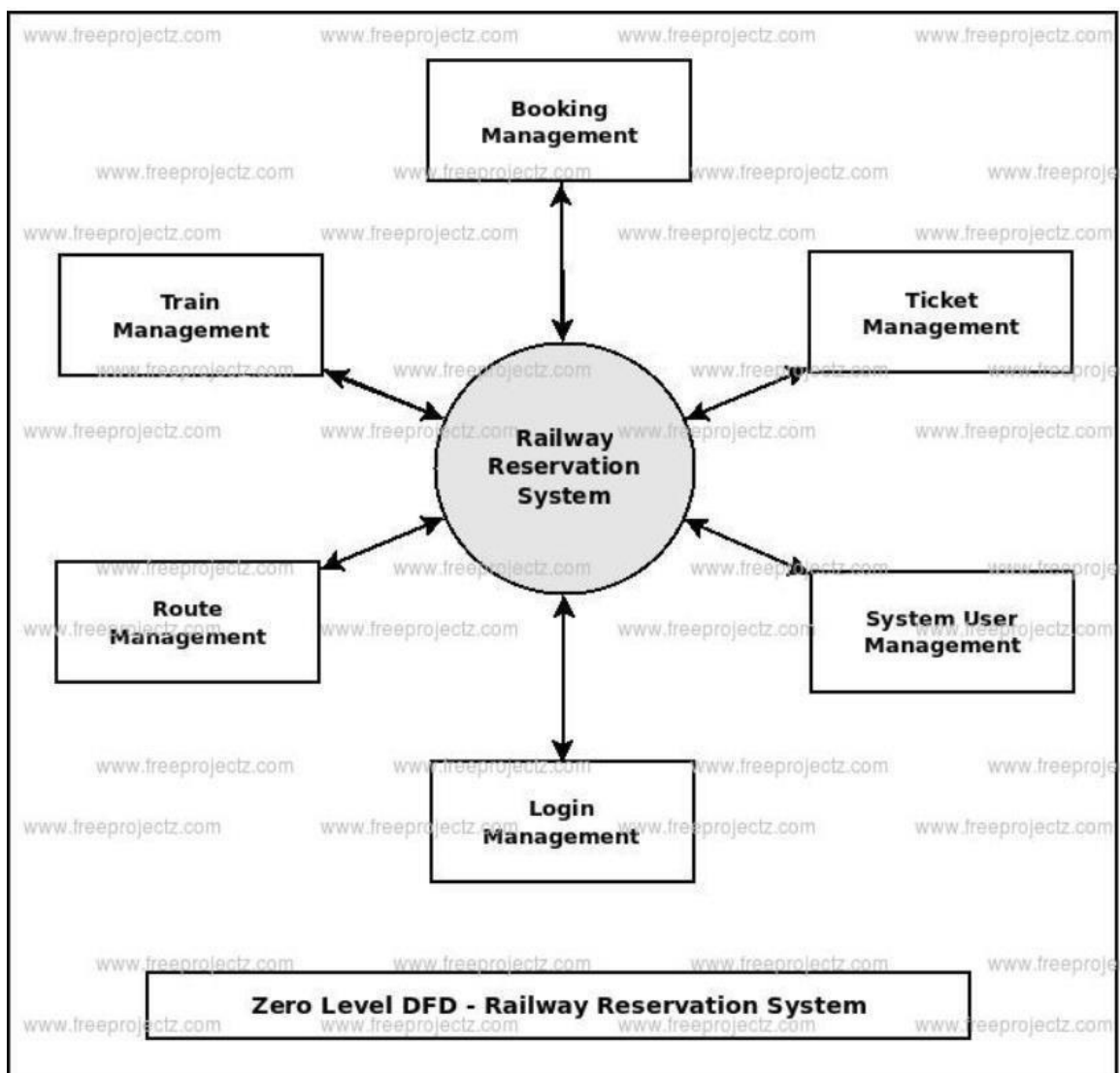
- Desktop PC or a Laptop
- Printer
- Operating System – Windows 10
- Intel® Core™ i3-6006U CPU @ 2.00GHz
- 4.00 GB RAM
- 64-bit operating system, x64 based processor
- 1024 x 768 monitor resolution
- Keyboard and Mouse

### **3.3 DATA FLOW DIAGRAM :**

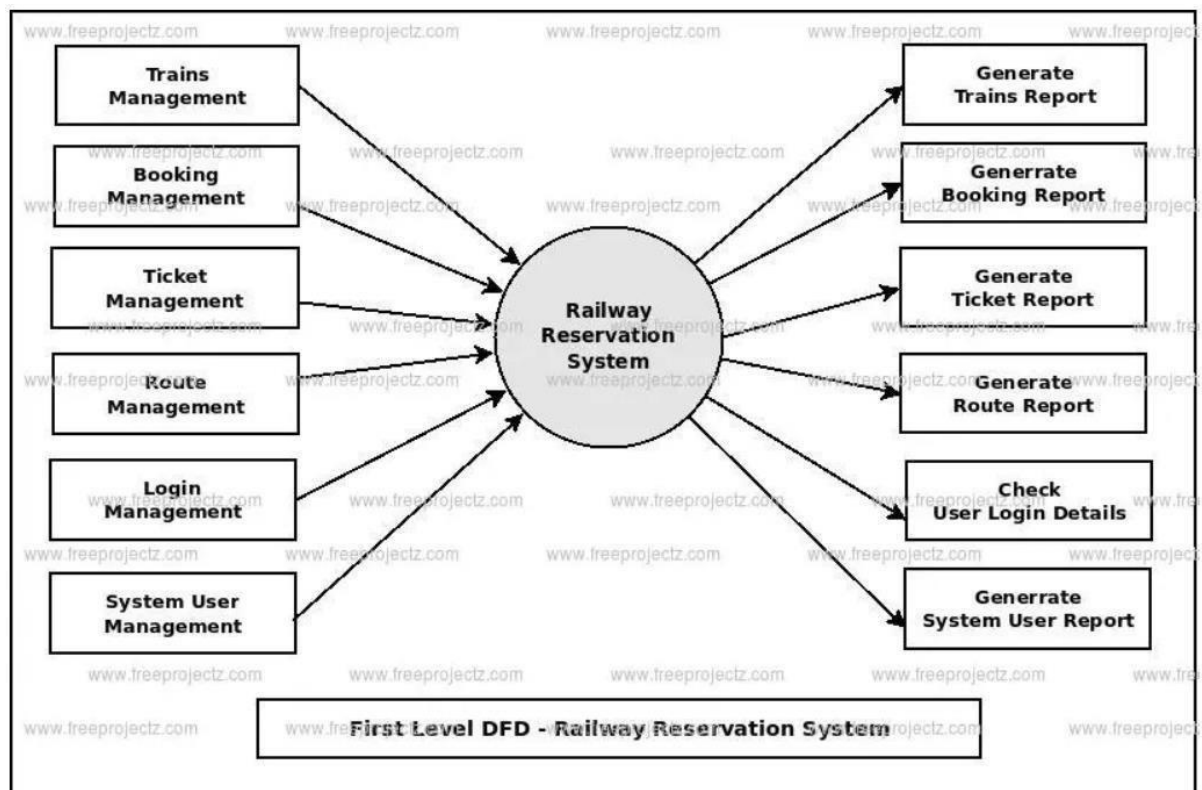
DFD is an important tool used by system analysis. A data flow diagram model, a system using external entities from which data flows through a process which transforms the data and creates output data transforms which go to other processes external entities such as files. The main merit of DFD is that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

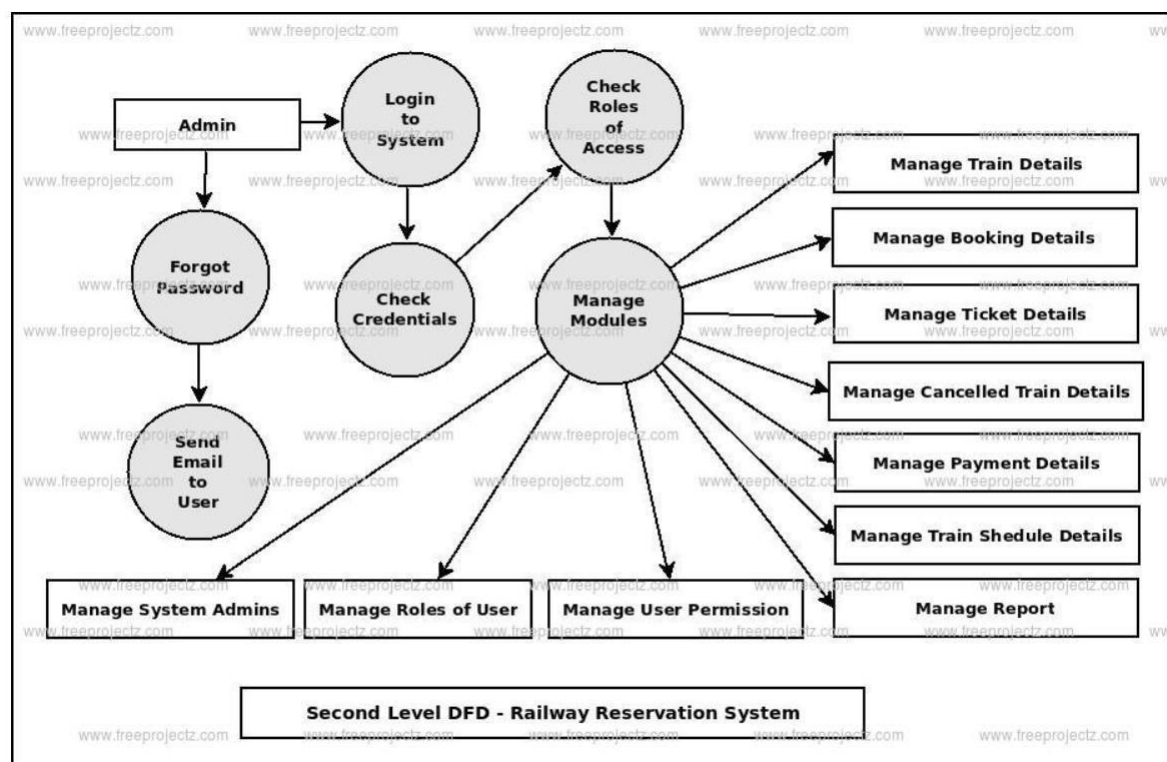
## ZERO LEVEL :



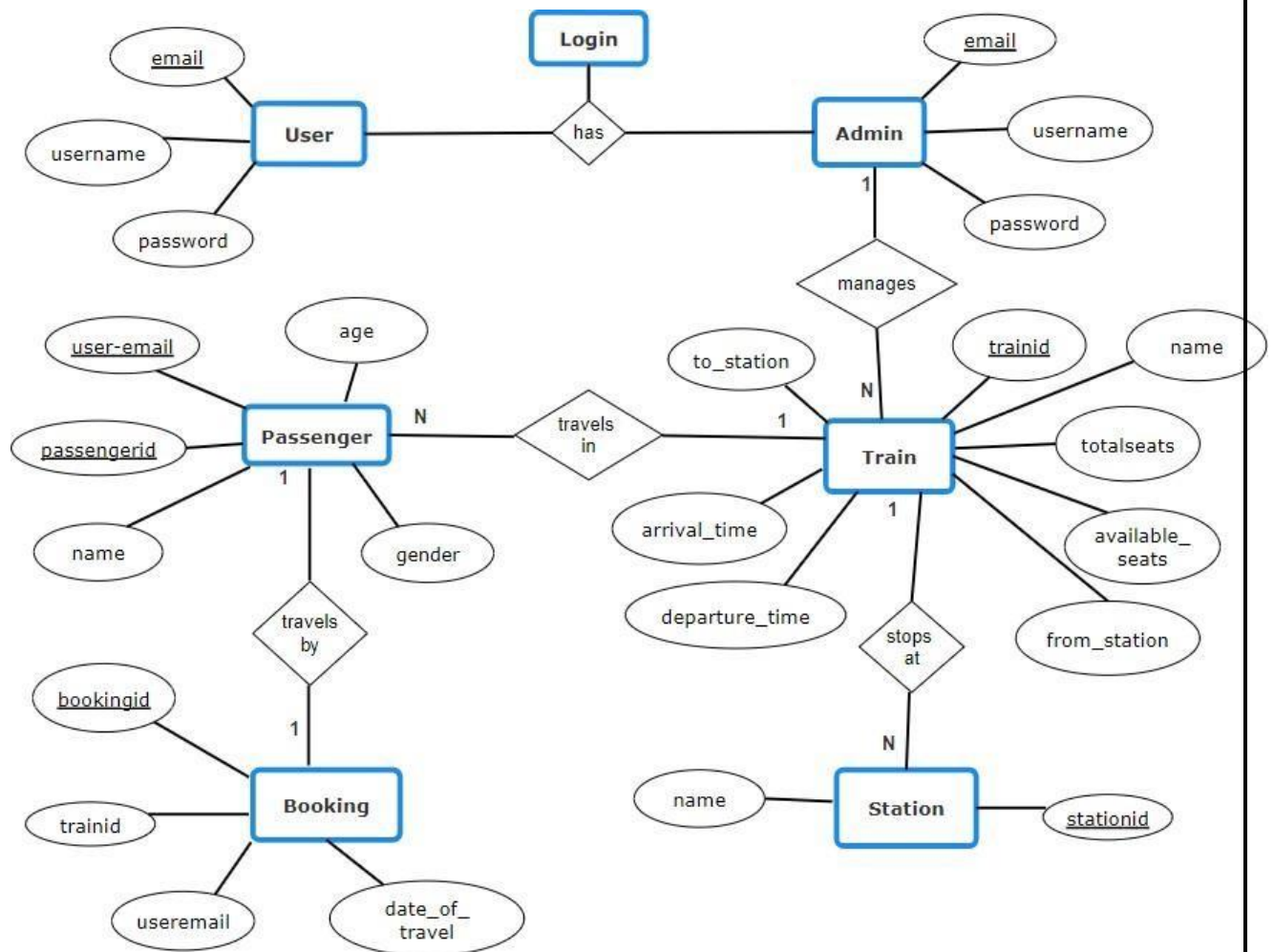
## FIRST LEVEL :



## SECOND LEVEL :




### 3.4 ER DIAGRAM :



### 3.5 NORMALIZATION :

### RAW DATABASE :

Column Name	Data Type	Constraints
`station_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`name`	`varchar(100)`	`NOT NULL`
`user_name`	`varchar(100)`	`NOT NULL`
`user_email`	`varchar(100)`	`NOT NULL`, `PRIMARY KEY`, `UNIQUE KEY`
`user_password`	`varchar(100)`	`NOT NULL`
`admin_email`	`varchar(255)`	`NOT NULL`, `PRIMARY KEY`, `UNIQUE KEY`
`admin_name`	`varchar(100)`	`NOT NULL`, `UNIQUE KEY`
`admin_password`	`varchar(255)`	`NOT NULL`
`train_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`train_name`	`varchar(100)`	`NOT NULL`
`from_station`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`station_id`)
`to_station`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`station_id`)
`departure_time`	`time`	`NOT NULL`
`arrival_time`	`time`	`NOT NULL`
`total_seats`	`int`	 `NULL`

`from_station`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`station_id`)
`to_station`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`station_id`)
`departure_time`	`time`	`NOT NULL`
`arrival_time`	`time`	`NOT NULL`
`total_seats`	`int`	`NOT NULL`
`available_seats`	`int`	`NOT NULL`
`booking_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`booking_train_id`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`train_id`)
`date_of_travel`	`date`	`NOT NULL`
`booking_email`	`varchar(100)`	`NOT NULL`, `FOREIGN KEY` (`user_email`)
`passenger_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`booking_id`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`booking_id`)
`passenger_name`	`varchar(100)`	`DEFAULT NULL`
`passenger_age`	`int`	`DEFAULT NULL`
`passenger_sex`	`varchar(10)`	`DEFAULT NULL`



## NORMALIZED DATABASE :

`stations`

Column Name	Data Type	Constraints
`station_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`name`	`varchar(100)`	`NOT NULL`

`users`

Column Name	Data Type	Constraints
`name`	`varchar(100)`	`NOT NULL`
`email`	`varchar(100)`	`NOT NULL`, `PRIMARY KEY`, `UNIQUE KEY`
`password`	`varchar(100)`	`NOT NULL`

`admins`

Column Name	Data Type	Constraints
`email`	`varchar(255)`	`NOT NULL`, `PRIMARY KEY`, `UNIQUE KEY`
`name`	`varchar(100)`	`NOT NULL`, `UNIQUE KEY`
`password`	`varchar(255)`	`NOT NULL`

`trains`

Column Name	Data Type	Constraints
`train_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`name`	`varchar(100)`	`NOT NULL`
`from_station`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`stations`.`station_id`)
`to_station`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`stations`.`station_id`)
`departure_time`	`time`	`NOT NULL`
`arrival_time`	`time`	`NOT NULL`
`total_seats`	`int`	`NOT NULL`
`available_seats`	`int`	`NOT NULL`

`bookings`

Column Name	Data Type	Constraints
`booking_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`train_id`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`trains`.`train_id`)
`date_of_travel`	`date`	`NOT NULL`
`email`	`varchar(100)`	`NOT NULL`, `FOREIGN KEY` (`users`.`email`)

`passengers`

Column Name	Data Type	Constraints
`passenger_id`	`int`	`NOT NULL`, `AUTO_INCREMENT`, `PRIMARY KEY`
`booking_id`	`int`	`DEFAULT NULL`, `KEY`, `FOREIGN KEY` (`bookings`.`booking_id`)
`name`	`varchar(100)`	`DEFAULT NULL`
`age`	`int`	`DEFAULT NULL`
`sex`	`varchar(10)`	`DEFAULT NULL`

## 4. 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"
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-1.34-3-3-3,1.34-3,3-3,1.34,3,3Z"></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_id 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
href="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  
    [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"

```

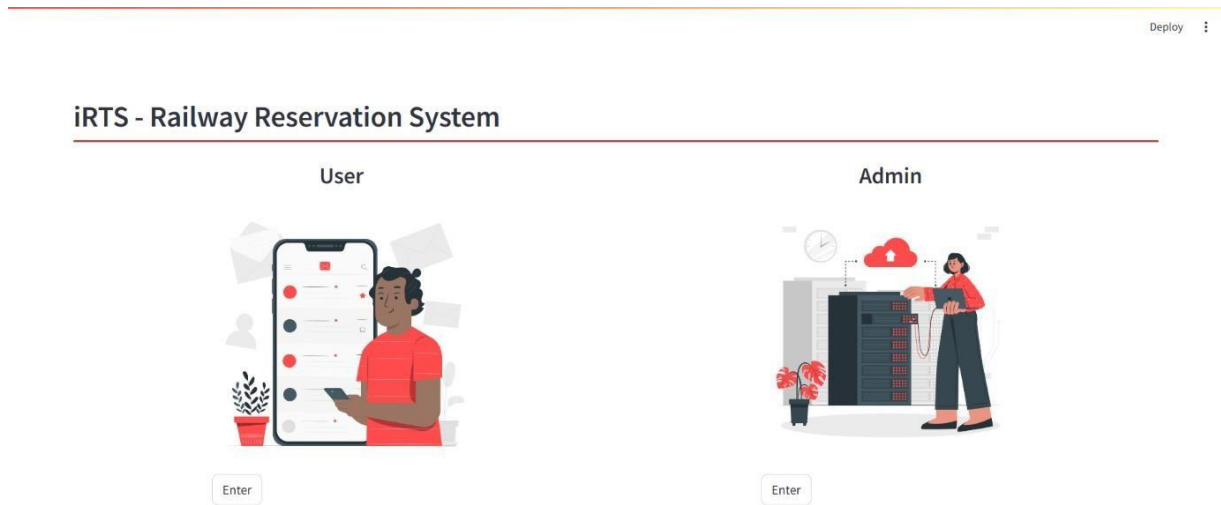
```
st.success(f"Booking ID {ticket['booking_id']} deleted.")
st.rerun() # Refresh the page to show updated tickets

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

## 5.RESULT AND DISCUSSION

### 5.1 USER DOCUMENTATION :

On boarding page :



Sign up page :

The sign up page for the iRTS - Railway Reservation System features a header with a 'Deploy' button. Below the header, there is a navigation bar with 'Login' and 'Sign Up' buttons. The 'Sign Up' button is highlighted in red. Below the navigation bar, the title 'Sign Up as user' is displayed. The form includes four input fields: 'Username', 'Email', 'Password', and 'Confirm Password'. Each field has a corresponding label and a placeholder. The 'Password' and 'Confirm Password' fields have eye icons for toggling visibility. A 'Sign Up' button is located at the bottom of the form.

## User login :

Deploy 

 **Login**

 Sign Up

### Login as user

Email

Password



Login

## Admin login :

Deploy 

### Login as admin

Email

Password



Login

Home page :

×

Deploy

User Dashboard

Main Menu

Home

Book Ticket


View Tickets

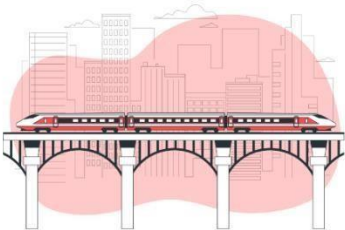
Contact Us


About Us

Logged In


Logout

 iTRS - Train Reservation System




 Available Trains


	train_id	name	from_station	to_station	departure_time	arrival_time	total_seats	available_seats
0	1	Rajdhani Express	New Delhi	Mumbai Central	0 days 07:00:00	0 days 19:00:00	500	450
1	2	Duronto Express	Mumbai Central	Chennai Central	0 days 08:00:00	0 days 20:00:00	600	550
2	3	Shatabdi Express	Chennai Central	Howrah Junction	0 days 09:00:00	0 days 21:00:00	550	500
3	4	Garib Rath	Howrah Junction	Bangalore City Junction	0 days 10:00:00	0 days 22:00:00	700	650
4	5	Jan Shatabdi	Bangalore City Junction	Secunderabad Junction	0 days 11:00:00	0 days 23:00:00	300	250
5	6	Humsafar Express	Secunderabad Junction	Ahmedabad Junction	0 days 06:00:00	0 days 18:00:00	400	350
6	7	Tejas Express	Ahmedabad Junction	Pune Junction	0 days 07:30:00	0 days 19:30:00	450	400
7	8	Antyodaya Express	Pune Junction	Kolkata Shalimar	0 days 08:30:00	0 days 20:30:00	500	450
8	9	Vande Bharat Express	Kolkata Shalimar	Jaipur Junction	0 days 09:30:00	0 days 21:30:00	600	550
9	10	Sampark Kranti Express	Jaipur Junction	New Delhi	0 days 10:30:00	0 days 22:30:00	550	500

 Book Tickets

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

 About Us

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.

 Contact Us

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

© 2024 iTRS - Train Reservation System. All rights reserved.



# Book ticket page :

✕

Deploy

User Dashboard

Main Menu

Home

Book Ticket

View Tickets

Contact Us

About Us

Logged In

Logout

Book Train Tickets

Logged in as: [kanaan@gmail.com](#)

Search Filter

From

To

Date of Travel

New Delhi

Mumbai Central

2024/05/26

Search Filter

Available Trains

	train_id	name	from_station	to_station	departure_time	arrival_time	total_seats	available_seats
0	1	Rajdhani Express	1	2	7 hours	19 hours	500	450

Enter Passenger Details

You selected the following train:

Train ID: 1, Train Name: Rajdhani Express, Departure: 0 days 07:00:00, Arrival: 0 days 19:00:00

Passenger Details

Passenger 1

Name

Age

Sex

Other

Passenger 2

Name

Age

Sex

Other

Passenger 3

Name

Age

Sex

Other

Passenger 4

Name

Age

Sex

Other

Passenger 5

Name

Age

Sex

Other

Passenger 6

Name

Age

Sex

Other

Submit

View ticket page :

×

Deploy

⋮

User Dashboard

🏠 Main Menu

🏠 Home

🎫 Book Ticket

📋 **View Tickets**

✉ Contact Us

📄 About Us

👤 **Logged In** ●

Logout

🎫

**View and Manage Your Tickets**

Welcome to your ticket dashboard!

Logged in as: [kanaan@gmail.com](mailto:kanaan@gmail.com)

🔍 Search Tickets

Enter Booking ID or Train ID

Found 1 ticket(s):

📄 Booking ID: 1

Train ID: 1

Date of Travel: 2024-05-26

Name: Kanaan

Email: [kanaan@gmail.com](mailto:kanaan@gmail.com)

Passengers:

Name: Muthu Krish	Age: 76	Sex: Male	<div>✖ Delete Passenger</div>
Name: Raj Vani	Age: 68	Sex: Female	<div>✖ Delete Passenger</div>

🗑 Delete Booking

66

## Contact us page :

×

Deploy

User Dashboard

Main Menu

Home

Book Ticket

View Tickets

Contact Us


About Us

Logged In

Logout

Contact Us

We're here to help!



Email: [support@trainbookingsystem.com](mailto:support@trainbookingsystem.com)

Phone: +1 234 567 890

Address: 123 Train St, Booking City, Country

Send Us a Message

Your Name

Your Email

Subject

Message

Send

[Facebook](#)

[Twitter](#)

[LinkedIn](#)

[Instagram](#)

67



## Manage station page :

×

Admin Dashboard

✂ Admin Menu

🏠 Home

🚂 Manage Trains

👤 **Manage Stations**


⚙ Settings

📖 Guide

👤 **Logged In** ●

Logout

Deploy ⋮



## Manage Stations

🗉 Manage Stations

👁 View

+ **Add**

🔄 Update

🗑 Delete

### Add Station

Station Name

Add

×

Admin Dashboard

✂ Admin Menu

🏠 Home

🚂 Manage Trains

👤 **Manage Stations**


⚙ Settings

📖 Guide

👤 **Logged In** ●

Logout

Deploy ⋮



## Manage Stations

🗉 Manage Stations

👁 View

+ Add

🔄 Update

🗑 **Delete**

🔗 Delete Station

Select Station to Delete

New Delhi ▼

Delete

## Manage train page :

Admin Dashboard

Admin Menu

Home

Manage Trains

Manage Stations

Settings

Guide

Logged In

Logout

Deploy

Manage Trains

View

Add

Update

Delete

Add Train

Train Name

From Station

New Delhi

To Station

New Delhi

Departure Time

19:32

Arrival Time

19:32

Total Seats

1

Available Seats

0

Add

Admin Dashboard

Admin Menu

Home

Manage Trains

Manage Stations

Settings

Guide

Logged In

Logout

Deploy

Manage Trains

View

Add

Update

Delete

Delete Train

Select Train to Delete

Rajdhani Express

Delete

## 6. TESTING

### 6.1 Unit Testing

Unit testing 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

### 6.2 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.

### 6.3 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.

### 6.4 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. This testing 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-normalization-explained-in-simple-english/>
- Database Design Documentation: <https://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.

